Πανεπιστήμιο Πειραιώς

Τμήμα Ψηφιακών Συστημάτων

Μεταπτυχιακή Διπλωματική Εργασία για το

Π.Μ.Σ. Ασφάλειας Ψηφιακών Συστημάτων

Θέμα:

Ανάπτυξη Ασφαλούς Λογισμικού - Έλεγχος Ασφάλειας Κώδικα (Java Frameworks Oriented)

Επιβλέποντες Καθηγητές:

Κωνσταντίνος Λαμπρινουδάκης

Συντάκτης:	Email:	Αριθμός Μητρώου:
Θεόδωρος Μαλαχιάς	tmalachias@ssl-unipi.gr	1919

Απρίλιος, 2021

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον καθηγητή κ. Λαμπρινουδάκη Κωνσταντίνο και τον Ταγματάρχη κ. Βάσιο Γεώργιο για την βοήθεια και επίβλεψη τους κατά τη διάρκεια εκπόνησης της μεταπτυχιακής διπλωματικής μου εργασίας.

0.0 Περίληψη

Η παρούσα διπλωματική έχει ως στόχο την παροχή οδηγιών σε μία ομάδα που αναπτύσσει λογισμικό για μία εφαρμογή web σε γλώσσα Java. Στο πρώτο κεφάλαιο αναλύονται οι διαφορετικές αρχιτεκτονικές που υπάρχουν για την ασφαλή ανάπτυξη εφαρμογών web. Επίσης παρέχονται συμβουλές για την ενίσχυση αυτού του στόχου. Στο κύριο μέρος της διπλωματικής και επόμενο κεφάλαιο αναλύονται διεξοδικά και με παραδείγματα καλές πρακτικές ασφαλούς προγραμματισμού σε Java. Σε κάθε κατηγορία πρακτικών αναλύεται ξεχωριστά πώς εφαρμόζεται σε διάφορα frameworks, η κάθε πρακτική, όπως το JSF, το Spring και το Struts με περισσότερη έμφαση στο JSF καθώς υπάρχει ευρύτερη χρήση του, αλλά και βιβλιογραφία. Στο τελευταίο κεφάλαιο αναλύεται το Static και Dynamic Security Testing (SAST & DAST) καθώς και λογισμικά ανοιχτού κώδικα που μπορεί κάποιος να χρησιμοποιήσει για να να διεξάγει SAST και DAST.

1.0 Εισαγωγή

Η τεράστια αύξηση των ψηφιακών υπηρεσιών είναι γεγονός. Ειδικά κατά τη διάρκεια της πανδημίας του COVID-19 οι περισσότερες επιχειρήσεις έψαξαν διέξοδο στον ψηφιακό κόσμο. Είναι εμφανές ότι η ασφάλεια στον παγκόσμιο ιστό είναι υψίστης σημασίας για τους καταναλωτές, αλλά και για τις ίδιες τις επιχειρήσεις. Επίσης, η εύρεση πληροφοριών πλέον για το οτιδήποτε στο διαδίκτυο, έχει και ως φυσικό επακόλουθο την αύξηση της εγκληματικότητας στον κυβερνοχώρο. Με τα παραπάνω δεδομένα και έχοντας κατα νου την κοινωνική ευθύνη ή και καθήκον δημιουργήθηκε η παρούσα διπλωματική. Όπως αναφέρεται ρητά και παρακάτω είναι σημαντικό η ασφάλεια να ξεκινάει από τα πρώτα στάδια ανάπτυξης μιας εφαρμογής web, καθώς μειώνεται δραστικά το κόστος διόρθωσης και ανακατασκευής, αλλά αυξάνεται ταυτόχρονα και η ολική ασφάλεια. Η γλώσσα για την οποία αναλύεται ο ασφαλής προγραμματισμός και οι τακτικές ασφαλούς σχεδιασμού είναι η Java, η οποία χρησιμοποιείται ευρέως σε εφαρμογές web και όχι μόνο.

2.0 Ασφαλής Σχεδιασμός Εφαρμογών και Αρχιτεκτονική

Οι Web εφαρμογές δημιουργούν ένα μεγάλο πλήθος ανησυχιών στους δημιουργούς τους. Όταν μια εφαρμογή “χτίζεται” από την αρχή με βάση την ασφάλεια είναι πιο ασφαλής και πιο ανθεκτική σε επιθέσεις. Εφαρμόζοντας ασφαλής τακτικές αρχιτεκτονικής και σχεδιασμού μαζί με πολιτικές ασφάλειας κατα τον αρχικό σχεδιασμό, δημιουργούνται επιτυχημένες εφαρμογές που δεν υστερούν σε ασφάλεια. Σε αυτό το κεφάλαιο θα γίνει περιγραφή και αναφορά σε αρχιτεκτονικές ασφάλειας και στρατηγικές σχεδιασμού που πρέπει να ακολουθούνται κατα τη διάρκεια όλων των φάσεων σχεδιασμού.

2.1 Software Development Life Cycle (SDLC)

Το Software Development Life Cycle (SDLC) αναφέρεται στην μεθοδολογία που εφαρμόζεται στην ανάπτυξη ενός προϊόντος λογισμικού. Το SDLC είναι ένα αναπόσπαστο κομμάτι του κύκλου ανάπτυξης ενός συστήματος. Υπάρχουν πολλά διαφορετικά μοντέλα που μπορούν να χρησιμοποιηθούν στον κύκλο ανάπτυξης ενός λογισμικού. Κάθε μοντέλο περιέχει μία συγκεκριμένη προσέγγιση, με την οποία αντιμετωπίζει διάφορες ενέργειες που γίνονται στην διαδικασία της ανάπτυξης. Κάποια παραδείγματα τέτοιων μοντέλων είναι το waterfall,το spiral, το iterative, το agile, το radical application και τα code and fix μοντέλα. Το πιο συχνά χρησιμοποιούμενο διεθνές πρότυπο για την ανάπτυξη λογισμικού είναι το ISO/IEC 12207, στο οποίο ο στόχος του SDLC είναι να αναπτύξει ένα πρότυπο που έχει την ιδιότητα να αντιμετωπίσει εργασίες που συμπεριλαμβάνονται στην ανάπτυξη και στην συντήρηση του λογισμικού. Γενικά, το SDLC έχει 5 απαραίτητες φάσεις σε όλα τα μοντέλα. Αυτές είναι:

- ❏ Απαιτήσεις
- ❏ Σχεδιασμός
- ❏ Εφαρμογή
- ❏ Έλεγχος
- ❏ Παραγωγή

Στην φάση των Απαιτήσεων, συλλέγονται οι απαιτήσεις με βάση τις ανάγκες της επιχείρησης. Η φάση του Σχεδιασμού περιγράφει τη δομή και τη σύνθεση του νέου συστήματος σαν μία συλλογή από ενότητες και υποσυστήματα. Η φάση της Εφαρμογής είναι μια φάση προγραμματισμού στην οποία ο Σχεδιασμός μετατρέπεται σε πραγματικότητα. Η φάση του Ελέγχου χρησιμοποιείται για για επαληθευτεί η λειτουργικότητα της εφαρμογής που δημιουργήθηκε. Τέλος, η τελική φάση της Παραγωγής προετοιμάζει το σύστημα για τους τελικούς χρήστες.

Βλέπουμε λοιπόν ότι το SDLC ξεκινάει με μία συγκεκριμένη ανάπτυξη που βασίζεται σε ολόκληρο το σύστημα που έχει σχεδιαστεί και εφαρμοστεί. Στην ανάπτυξη λογισμικού, χρειάζονται ασφαλείς διαδικασίες κατά την φάση της ανάπτυξης για να παραχθεί ασφαλές λογισμικό. Γι’ αυτό, η ασφάλεια κατα τη διάρκεια των φάσεων της ανάπτυξης και της ασφάλειας του λογισμικού σχετίζονται και είναι απαραίτητα για την ολική ασφάλεια.

2.2 Σχεδιασμός και Αρχιτεκτονική

Η ασφάλεια σε επίπεδο σχεδιασμού είναι ένα κρίσιμο καθήκον και πρέπει να προτιμάται στα αρχικά στάδια του κύκλου ανάπτυξης του λογισμικού, για τη δημιουργία μιας “στιβαρής” εφαρμογής. Η ασφάλεια που ενσωματώνεται από τη φάση του σχεδιασμού βοηθάει περισσότερο να βρεθούν σχετικές ευπάθειες. Η κατάλληλη διακυβέρνηση ασφάλειας κατα τη διάρκεια της φάσης σχεδιασμού βοηθάει περισσότερο να βρεθούν ελαττώματα ασφαλείας στα αρχικά στάδια της ανάπτυξης. Ο ασφαλής σχεδιασμός μιας εφαρμογής βασίζεται σε προϋποθέσεις ασφαλείας που έχουν αναγνωριστεί κατα τη διάρκεια των προϋποθέσεων ασφαλείας της SDLC φάσης.

Στόχοι της διαδικασίας ασφαλούς σχεδιασμού

- ❏ Οι στόχοι της διαδικασίας ασφαλούς σχεδιασμού εξαρτώνται κυρίως από την απόδοση των προγραμματιστών της εφαρμογής.

- ❏ Οι προγραμματιστές θα πρέπει να κατανοήσουν λεπτομερώς τις απειλές και να καθορίσουν επίσης τις τεχνικές που θα αντιμετωπίσουν αυτές τις απειλές.

- ❏ Οι προγραμματιστές θα πρέπει να σχεδιάσουν ασφαλή αρχιτεκτονική που μετριάζει όλες τις πιθανές απειλές.

- ❏ Οι προγραμματιστές πρέπει να εφαρμόζουν αρχές ασφαλούς σχεδιασμού για να προσθέσουν ασφάλεια κατά τον προγραμματισμο.

Ενέργειες Ασφαλούς Σχεδιασμού

Οι ενέργειες που πρέπει να εκτελούνται κατά τον ασφαλή σχεδιασμό εξαρτώνται από τέσσερα βήματα:

I. **Προδιαγραφές Απαιτήσεων Ασφάλειας:** Σχεδιασμός της εφαρμογής ακολουθώντας τις προδιαγραφές ασφαλείας που ορίζονται στη φάση απαιτήσεων ασφαλείας.

II. **Αρχές ασφαλούς σχεδιασμού:** Ορισμός ασφαλών προτύπων προγραμματισμού που πρέπει να εφαρμόζονται κατά τη φάση ανάπτυξης.

III. **Μοντελοποίηση απειλών:** Οι απειλές πρέπει να ανακαλυφθούν μέσω της διαδικασίας μοντελοποίησης απειλών.

IV. **Αρχιτεκτονική ασφαλούς εφαρμογής:** Σχεδιασμός ασφαλούς αρχιτεκτονικής εφαρμογής.

2.2.1 Αρχές Ασφαλούς Σχεδιασμού

Οι αρχές ασφαλούς σχεδιασμού είναι το σύνολο των πρακτικών που σχεδιάστηκαν για να ακολουθούν τους κανόνες ασφαλείας κατά τη φάση ανάπτυξης. Οι αρχές ασφαλείας παρέχουν οδηγίες για τη λήψη ασφαλούς απόφασης σχετικά με την αρχιτεκτονική. Παρέχουν επίσης μεθοδολογίες που βοηθούν στην εξάλειψη και τον περιορισμό των ατελειών σε επίπεδο αρχιτεκτονικής εντός της εφαρμογής. Οι αρχές ασφαλούς σχεδιασμού αποτελούνται από έναν αριθμό αρχών που χρησιμοποιούνται για την αποτροπή κοινών τρωτών σημείων ασφαλείας:

Ασφάλεια μέσω της αφάνειας - Security Through Obscurity (STO)

- ❏ Το STO εμποδίζει τους ανεπιθύμητους χρήστες να έχουν πρόσβαση σε ευαίσθητα δεδομένα και πληροφορίες.

- ❏ Τα συστήματα STO διαθέτουν σημαντικά ελαττώματα, αλλά ο σχεδιαστής πιστεύει ότι ο εισβολέας δεν θα εντοπίσει αυτά τα ελαττώματα.

- ❏ Πρέπει να χρησιμοποιείται μόνο σε περιπτώσεις όπου υπάρχουν και άλλα επίπεδα ασφαλείας όπως οι περιορισμοί με βάση την IP, two factor authentication (2FA), TCP Wrappers και firewalling.[1]

Ασφάλεια του πιο αδύναμου συνδέσμου

- ❏ Οι επιτιθέμενοι στοχεύουν έναν ασθενέστερο σύνδεσμο στο σύστημα καθώς είναι πιο εύκολο να του επιτεθούν.

- ❏ Αντιμετώπιση όλων των πιθανών κινδύνων και τρωτών σημείων ασφαλείας πριν ξεκινήσει η ανάπτυξη.[2]

Χρήση της αρχής Least Privilege

Η χρήση των αρχών του ελάχιστου προνόμου μπορεί να βοηθήσει στην άμυνα ενάντια σε πολλές επιθέσεις με:

---

[1] "Security Through Obscurity - SecurityTrails." https://securitytrails.com/blog/security-through-obscurity.

[2] "Securing Enterprise Web Applications at the Source - OWASP ...." https://owasp.org/www-pdf-archive/Securing_Enterprise_Web_Applications_at_the_Source.pdf. Σελ 19-20

- ❏ Εκχώρηση δικαιωμάτων μόνο σε όσους χρειάζονται προνόμια για να εκτελέσουν συγκεκριμένη εργασία.
- ❏ Εκτέλεση και εγκατάσταση εφαρμογών χειροκίνητα.
- ❏ Σύνταξη εφαρμογών για χρήστες που δεν έχουν δικαιώματα διαχειριστή.[3]

Ασφαλής από προεπιλογή

- ❏ Ορισμένα λογισμικά ή εφαρμογές έχουν προεπιλεγμένα χαρακτηριστικά ασφαλείας, ο χρήστης πρέπει να πάρει άδεια για μείωση της ασφάλειας.[4]

Αποτυχία με ασφάλεια

- ❏ Σχεδιασμός μηχανισμού ασφαλείας, με τέτοιο τρόπο, ώστε μία αποτυχία του συστήματος, που ενεργοποιεί κάποια εξαίρεση, να ακολουθεί τον ίδιο τρόπο εκτέλεσης με μία διαδικασία που είναι απαγορευμένη (Unauthorized).
- ❏ Προγραμματισμός με σωστή λογική, ώστε αν ενεργοποιηθεί κάποια εξαίρεση να μην αποφεύγει κάποιο μηχανισμό ασφαλείας.

```
isAdmin = true;
try {
  codeWhichMayFail();
  isAdmin = isUserInRole( “Administrator” );
}
catch (Exception ex)
{
  log.write(ex.toString());
}
```

```
isAdmin = false;
try {
  codeWhichMayFail();
  isAdmin = isUserInrole( "Administrator" );
}
catch (Exception ex)
{
  log.write(ex.toString());
}
```

Όπως φαίνεται στα παραπάνω code snippets το αριστερό θα έχει security risk, αφού παίρνει από προεπιλογή, ότι ο χρήστης είναι admin. Έτσι αν υπάρξει exception θα παραμείνει true. Ενώ στο δεξί αυτό διορθώνεται αρχικοποιώντας το isAdmin με false.[5]

Εφαρμογή άμυνας σε βάθος

- ❏ Εφαρμογή μηχανισμών ασφαλείας σε διαφορετικά επίπεδα εφαρμογής, συμπεριλαμβανομένου του επιπέδου δικτύου, του επιπέδου πυρήνα, του φυσικού επιπέδου και το συστήματος αρχείων.[6]

Η είσοδος του χρήστη δεν είναι αξιόπιστη

- ❏ Προστασία της εφαρμογής από όλες τις εισόδους χρήστη, επειδή οποιαδήποτε είσοδος χρήστη ενδέχεται να έχει κακόβουλο σκοπό.
- ❏ Εφαρμογή μηχανισμού ασφαλείας σε όλες τις εισόδους του χρήστη για αποφυγή κακόβουλης δραστηριότητας.[7]

Μείωση της επιφάνειας επίθεσης

- ❏ Η επιφάνεια επίθεσης της εφαρμογής πρέπει να ελαχιστοποιηθεί, μέσω του αριθμού των σημείων εισόδου στην εφαρμογή.

---

[3] "Securing Enterprise Web Applications at the Source - OWASP ...." https://owasp.org/www-pdf-archive/Securing_Enterprise_Web_Applications_at_the_Source.pdf. Σελ 21-22

[4] "Securing Enterprise Web Applications at the Source - OWASP ...." https://owasp.org/www-pdf-archive/Securing_Enterprise_Web_Applications_at_the_Source.pdf. Σελ 137-138

[5] "Fail Securely | OWASP." https://owasp.org/www-community/Fail_securely.

[6] "Defence in depth and how it applies to web applications ...." https://www.acunetix.com/websitesecurity/defence-in-depth-and-how-it-applies-to-web-applications/.

[7] "The Dangers of Trusting User Input - enisa - europa.eu." 1 Ιουν. 2016, https://www.enisa.europa.eu/publications/info-notes/the-dangers-of-trusting-user-input.

- ❏ Απενεργοποίηση των περιττών δυνατοτήτων, πρωτοκόλλων, θυρών για μείωση των συνολικών κινδύνων.[8]

Ενεργοποίηση ελέγχου και καταγραφής

- ❏ Η εφαρμογή πρέπει να καταγράφει τα συμβάντα που σχετίζονται με την ασφάλεια.

- ❏ Ο έλεγχος και η καταγραφή επιτρέπει τη συλλογή πληροφοριών σχετικά με επιθέσεις.[9]

Διατήρηση Απλής Ασφάλειας

- ❏ Ο σχεδιασμός ασφαλείας πρέπει να είναι απλός, επειδή είναι εύκολο να ανακαλύψεις τα λάθη στην απλή αρχιτεκτονική και διαμόρφωση.

- ❏ Στην πολύπλοκη αρχιτεκτονική, η απαίτηση μηχανισμού ασφαλείας αυξάνεται επίσης για να φτάσει στο κατάλληλο επίπεδο ασφάλειας.[10](Kiss Principle)

Διαχωρισμός καθηκόντων

- ❏ Ο βασικός έλεγχος απάτης βασίζεται στον διαχωρισμό των καθηκόντων.

- ❏ Συγκεκριμένοι ρόλοι έχουν διαφορετικά επίπεδα εμπιστοσύνης από τους κανονικούς χρήστες. Συγκεκριμένα, οι διαχειριστές διαφέρουν από τους κανονικούς χρήστες. Γενικά, οι διαχειριστές δεν πρέπει να είναι χρήστες της εφαρμογής.[11]

Διόρθωση Ζητημάτων Ασφαλείας Σωστά

- ❏ Όταν εντοπιστεί ζήτημα ασφαλείας, είναι σημαντικό να αναπτυχθεί μια δοκιμή για αυτό και να ελεγχθεί η βασική αιτία του ζητήματος.

- ❏ Όταν χρησιμοποιούνται μοτίβα σχεδιασμού, είναι πιθανό το ζήτημα της ασφάλειας να είναι ευρέως διαδεδομένο σε όλες τις βάσεις κώδικα, επομένως είναι απαραίτητη η ανάπτυξη της σωστής επιδιόρθωσης χωρίς να χρειάζεται να γίνει σε κάθε σημείο που επαναλαμβάνεται ο κώδικας, αλλά σε ένα και να ενημερώνονται όλα.[12]

Εφαρμογή Ασφάλειας κατά τη Φάση της Σχεδίασης

- ❏ Η ενσωμάτωση μηχανισμών ασφαλείας από την αρχή της σχεδίασης της εφαρμογής βοηθάει στο να προληφθούν πολλοί κίνδυνοι από νωρίς.

- ❏ Επίσης το κόστος για απόσυρση ή δημιουργία νέων μηχανισμών κατά τη λειτουργία της εφαρμογής μεγαλώνουν κατά πολύ το κόστος.[13]

Προστασία ευαίσθητων δεδομένων

- ❏ Αποφυγή hard-coded απόρρητα δεδομένα όπως ο κωδικός πρόσβασης κατά τη διάρκεια της αυθεντικοποίησης.

---

[8] "Reducing your attack surface | Digital Shadows." 9 Απρ. 2019, https://www.digitalshadows.com/blog-and-research/reducing-your-attack-surface/.

[9] "Logging - OWASP Cheat Sheet Series." https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html.

[10] "Defence in depth and how it applies to web applications ...." https://www.acunetix.com/websitesecurity/defence-in-depth-and-how-it-applies-to-web-applications/. (KISS Principle)

[11] "Separation of duties - OWASP - Linux." 7 Απρ. 2009, https://www.linuxsecrets.com/owasp-wiki/index.php/Separation_of_duties.html. Πρόσβαση στις 9 Σεπ. 2020.

[12] "Security by Design Principles according to OWASP." 27 Μαρ. 2018, https://blog.threatpress.com/security-design-principles-owasp/. Πρόσβαση στις 9 Σεπ. 2020.

[13] "Promote Security throughout your Software Design phase ...." 21 Μαΐ. 2015, https://www.advantio.com/blog/secure-software-development-life-cycle-design-phase. Πρόσβαση στις 9 Σεπ. 2020.

- ❏ Διαβιβασμός μόνο κρυπτογραφημένων δεδομένων.[14]

Χειρισμός εξαιρέσεων

- ❏ Ο χειρισμός εξαιρέσεων συμβαίνει όταν οι συνθήκες σφάλματος διαταράσσουν τη ροή της εκτέλεσης προγραμματισμού.

- ❏ Ο κατάλληλος χειρισμός εξαιρέσεων προάγει τον κατάλληλο χειρισμό σφαλμάτων. Πρέπει να υπάρχει καταγραφή των εξαιρέσεων, έτσι ώστε να παίρνουν αρκετές πληροφορίες για αυτές η υποστήριξη και οι ομάδες ασφάλειας. [15]

Ασφαλής διαχείριση μνήμης

- ❏ Εξασφάλιση ορίων μνήμης στο μήκος των μεταβλητών εισόδου, πινάκων και ορισμάτων για την αποφυγή επιθέσεων υπερχείλισης buffer.

Προστασία μνήμης ή μυστικών αποθήκευσης

- ❏ Προστατέψτε τη μνήμη κρυπτογραφώντας μυστικά.

- ❏ Κρυπτογράφηση μυστικών με χρήση κατάλληλων μεθόδων κρυπτογράφησης.

Ανοχή σε σφάλματα

- ❏ Το λογισμικό πρέπει να σχεδιαστεί εφαρμόζοντας μια τέτοια στρατηγική που θα πρέπει να μπορεί να λειτουργεί παρουσία βλαβών.[16]

Ανίχνευση βλαβών

- ❏ Εντοπισμός σφαλμάτων συστήματος και δημιουργία σχετικών αντιδράσεων.

- ❏ Οθόνες συστήματος, οθόνες ασφαλείας, ενσωματωμένος έλεγχος, έλεγχος βρόχου είναι παραδείγματα ανίχνευσης σφαλμάτων.

Αφαίρεση βλαβών

- ❏ Τα σφάλματα πρέπει να αφαιρούνται κατά τη φάση σχεδιασμού.

- ❏ Η ανίχνευση σφαλμάτων, η επαλήθευση μέσω επιθεώρησης, οι ενσωματωμένες δοκιμές, οι λειτουργίες διόρθωσης είναι παραδείγματα αφαίρεσης σφαλμάτων.

Αποφυγή βλαβών

- ❏ Αποφυγή σφαλμάτων που μπορεί να οδηγήσουν σε σφάλματα συστήματος κατά τη φάση ανάπτυξης.

- ❏ Ο αμυντικός προγραμματισμός, η ελαχιστοποίηση σφαλμάτων κατά τη διαδικασία σχεδιασμού, η ελαχιστοποίηση του κρίσιμου κώδικα ασφαλείας με χρήση κατάλληλων τεχνικών SDLC, είναι κάποια παραδείγματα αποφυγής σφαλμάτων. [17]

Χαλαρή σύζευξη

---

[14] "OWASP TOP 10: Sensitive Data Exposure | Detectify Blog." 1 Ιουλ. 2016, https://blog.detectify.com/2016/07/01/owasp-top-10-sensitive-data-exposure-6/. Πρόσβαση στις 9 Σεπ. 2020.

[15] "C10: Handle all Errors and Exceptions | OWASP." https://owasp.org/www-project-proactive-controls/v3/en/c10-errors-exceptions. Πρόσβαση στις 11 Σεπ. 2020.

[16] "Denial of Service - OWASP Cheat Sheet Series." https://cheatsheetseries.owasp.org/cheatsheets/Denial_of_Service_Cheat_Sheet.html. Πρόσβαση στις 11 Σεπ. 2020.

[17] "1. (Miscel) What is the difference between fault avoidance and ...." https://www.csm.ornl.gov/~sheldon/cs422/solns/Ch1-9StudyQsKey.pdf. Πρόσβαση στις 11 Σεπ. 2020.

- ❏ Διαδικασίες που δεν εξαρτώνται από τη λειτουργία άλλων διαδικασιών έχουν χαλαρή σύζευξη.
- ❏ Η χαλαρή σύζευξη καθορίζει τη σχέση μεταξύ δύο συστημάτων.
- ❏ Η χαλαρή σύζευξη εφαρμόζεται για υπηρεσίες διαδικτύου ή αρχιτεκτονικές προσανατολισμένες στην υπηρεσία στις οποίες οι λεπτομέρειες εφαρμογής είναι κρυφές. [18]

Υψηλή συνοχή

- ❏ Διαδικασίες που εκτελούν μία μόνο λειτουργία.
- ❏ Η συνοχή είναι μια μέθοδος αναγνώρισης του εύρους των πολύ συσχετιζόμενων λειτουργιών στον πηγαίο κώδικα μιας μεμονωμένης ενότητας.
- ❏ Ο κωδικός υψηλής συνοχής αυξάνει την αξιοπιστία και την επαναχρησιμοποίηση του κώδικα χωρίς να επηρεάζει την πολυπλοκότητα. [19]

Διαχείριση αλλαγών και έλεγχος έκδοσης

- ❏ Η διαχείριση αλλαγών ελέγχει το κόστος ολοκλήρωσης και ανάπτυξης.
- ❏ Οι απαιτούμενες αλλαγές στον κώδικα ενδέχεται να δημιουργήσουν ευπάθειες.
- ❏ Οι ενδιαφερόμενοι επιτρέπουν την γνώση επικείμενων αλλαγών. [20]

2.3 Σχεδιασμός Ασφαλούς Αρχιτεκτονικής Εφαρμογών

Ο σχεδιασμός της αρχιτεκτονικής των εφαρμογών web περιλαμβάνει τρία στοιχεία που πρέπει να ληφθούν υπόψη, ο web εξυπηρετητής, ο εξυπηρετητής εφαρμογής και ο εξυπηρετητής της βάσης δεδομένων. Θα πρέπει να ενσωματωθεί ασφάλεια σε όλα τα στοιχεία το ίδιο ώστε να προληφθεί η παραβίαση του συστήματος. Η κατάλληλη χρήση της αρχής “άμυνας σε βάθος” κατα τη φάση της αρχιτεκτονικής συμβάλλει στη δημιουργία μιας στιβαρής και ασφαλής εφαρμογής.

Μηχανισμοί Ασφαλείας ανα στοιχείο:

- ❏ Web εξυπηρετητής: Πρέπει να ενσωματωθούν μηχανισμοί ασφαλείας όπως επικύρωση εισόδου (input validation), εξουσιοδότηση χρήστη (user authorization), ασφαλής εξαιρέσεις (secure exceptions) και ασφαλής παραμετροποίηση (secure configuration).
- ❏ Εξυπηρετητής εφαρμογής: Οι δράσεις ασφάλειας πρέπει να συμπεριλαμβάνουν αυθεντικοποίηση και εξουσιοδότηση χρηστών, έλεγχος ασφαλείας και καταγραφή συμβάντων, καθώς και ασφαλής συναλλαγές.
- ❏ Εξυπηρετητής Βάσης Δεδομένων: Αποθήκευση ευαίσθητων πληροφοριών με χρήση κρυπτογράφησης μέσω δυνατών και σύγχρονων κρυπτογραφικών αλγορίθμων. [21]

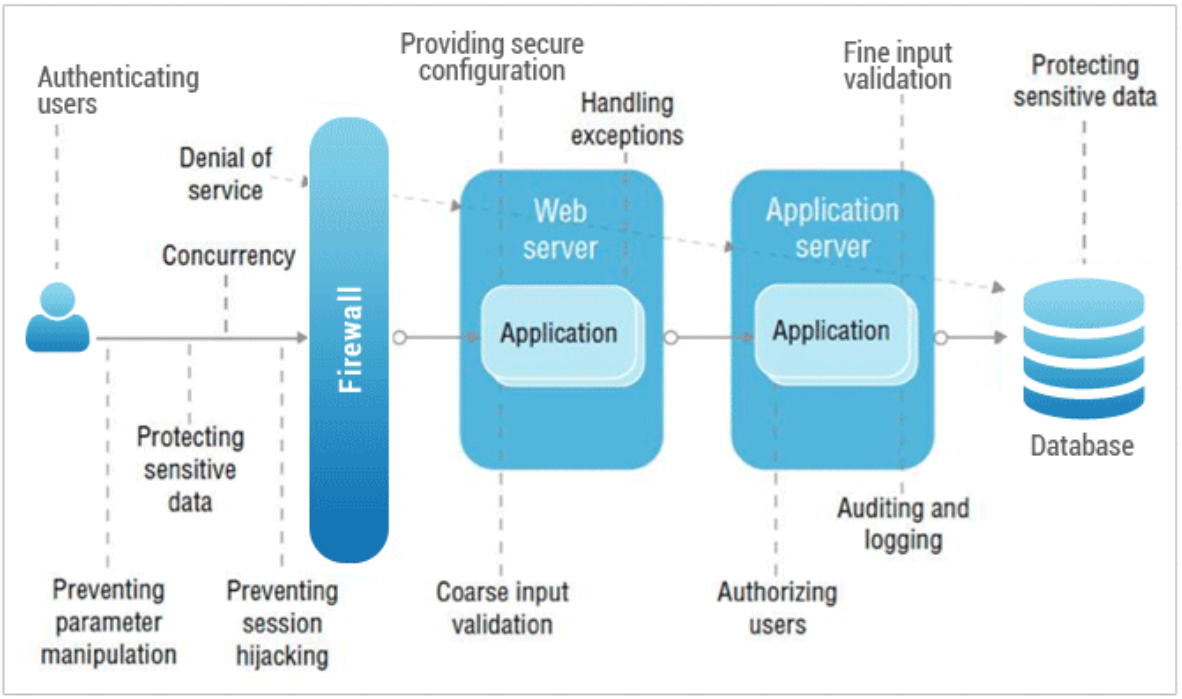
---

[18] "Why is loose coupling between services so ... - Ben Morris." 14 Απρ. 2019, https://www.ben-morris.com/why-is-loose-coupling-between-services-so-important/. Πρόσβαση στις 11 Σεπ. 2020.

[19] "Low Coupling, High Cohesion. The key to creating ... - Medium." 17 Σεπ. 2018, https://medium.com/clarityhub/low-coupling-high-cohesion-3610e35ac4a6. Πρόσβαση στις 11 Σεπ. 2020.

[20] "The OWASP Foundation OWASP Technology and Business ...." 28 Σεπ. 2010, https://owasp.org/www-pdf-archive//Technology_and_Business_Risk_Management_How_Application_Security_Fits_In.pdf. Πρόσβαση στις 11 Σεπ. 2020.

[21] "Security in Software Development and Infrastructure System ...." 26 Νοε. 2018, https://medium.com/cermati-tech/security-in-software-development-and-infrastructure-system-design-7b675c2323fc. Πρόσβαση στις 12 Σεπ. 2020.



SECURE ARCHITECTURE

2.4 Διαδικασία Ανάπτυξης Ασφαλούς Λογισμικού

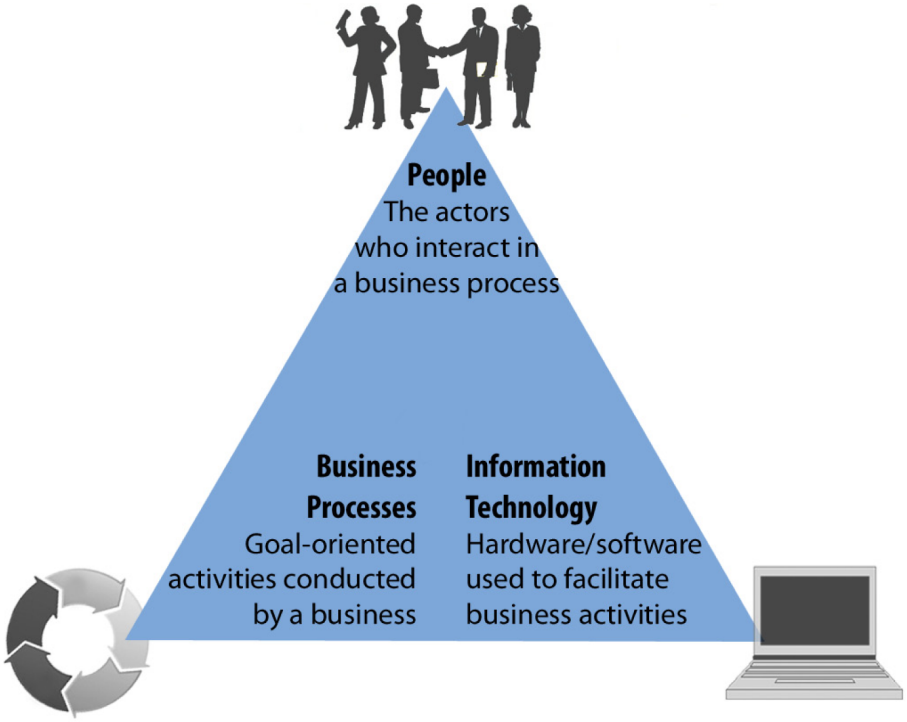
Ο καλύτερος τρόπος για να αποφευχθούν όλοι οι τύποι ευπαθειών είναι σχεδιάζοντας και υλοποιώντας χαρακτηριστικά ασφαλείας σε κάθε κύκλο ανάπτυξης. Η ασφαλής ανάπτυξη λογισμικού είναι μία μέθοδος που διαβεβαιώνει ότι η διαδικασία ανάπτυξης εφαρμογής που είναι σε εξέλιξη είναι τόσο ασφαλής όσο αναμενόταν. Η ασφαλής ανάπτυξη λογισμικού συμπεριλαμβάνει τη χρήση πολλών διαδικασιών, που είναι οι απαιτήσεις, ο σχεδιασμός, η ανάπτυξη, ο έλεγχος, η ανάπτυξη και η συντήρηση.

Τα βασικά πλεονεκτήματα της συγκεκριμένης πρότασης Ασφαλούς SDLC είναι:

- ❏ Αρκετά ασφαλές λογισμικό, διότι η ασφάλεια είναι ένας συνεχής αγώνας.
- ❏ Επίγνωση από τους επενδυτές σχετικά με θέματα ασφάλειας.
- ❏ Ανίχνευση ευπαθειών στο σύστημα από νωρίς.
- ❏ Μείωση κόστους λόγω της ανίχνευσης από νωρίς και λύσης των προβλημάτων.
- ❏ Μεγάλη μείωση των ουσιαστικών κινδύνων του συστήματος.

2.4.1 Τρίγωνο SDLC

Ο κύκλος ζωής ανάπτυξης λογισμικού (SDLC) είναι βασισμένος στους ανθρώπους, τις διαδικασίες και την τεχνολογία που συμμετέχουν όλα μαζί σε κάθε φάση του SDLC.



Στους ανθρώπους περιλαμβάνονται η κεντρική ομάδα ασφάλειας και η ομάδα ανάπτυξης, όπου η ομάδα ανάπτυξης κατευθύνει την διαδικασία και την ενημερώνει ενω η ομάδα ασφάλειας πραγματοποιεί τις εργασίες σχετικές με την ασφάλεια. Όσον αφορά την τεχνολογία, περιλαμβάνει εργαλεία που βοηθούν στην εύρεση ευπαθειών στον πηγαίο κώδικα ή την εύρεση ευπαθειών σε ενεργό παράδειγμα εφαρμογής. Το SDLC είναι μία εφικτή δομή που ταιριάζει τις ενέργειες ασφαλείας σε οποιαδήποτε μεθοδολογία, όπως π.χ. waterfall, agile ή DevOps. Η βασική προσέγγιση του SDLC περιέχει την φάση απαιτήσεων, τη φάση σχεδιασμού, την φάση ανάπτυξης, την φάση ελέγχου, την φάση εφαρμογής και τέλος τη φάση συντήρησης.[22]

Η φάση Απαιτήσεων

Η διαδικασία ορισμού των απαιτήσεων βασίζεται στις ανάγκες του τελικού χρήστη, εφόσον ο αναλυτής του project έχει λάβει feedback από τον τελικό χρήστη. Η ανατροφοδότηση θα δείξει τις βασικές λειτουργίες που θα χρησιμοποιηθούν για τον τελικό στόχο του project. Σε αυτή τη φάση θα οριστούν επίσης και θα συμπεριληφθούν οι καλύτερες πρακτικές ασφαλείας για τις συγκεκριμένες συνθήκες. Αυτές οι πρακτικές είναι το αποτέλεσμα βιομηχανικών προτύπων ή λύσεις από παλιά προβλήματα.

Οι απαιτήσεις ορίζουν τις λειτουργικές προϋποθέσεις ασφάλειας στην εφαρμογή και σε όλες τις ενέργειες στο SDLC. Χρησιμοποιούνται σαν ένα σημείο εκτέλεσης που επιβεβαιώνει ότι όλες οι πτυχές έχουν εξεταστεί πλήρως. Κατά τη φάση των απαιτήσεων, είναι ζωτικής σημασίας να εξεταστεί ένα σχέδιο ασφαλείας που θα ενσωματωθεί με βάση τους στόχους. Για παράδειγμα, η δημιουργία απαιτήσεων για τη λίστα ελέγχου πρόσβασης και τον τύπο αυθεντικοποίησης και απαιτήσεων ταυτότητας που απαιτούνται και τους διαφορετικούς ρόλους.

Η φάση του Σχεδιασμού

Ο σχεδιασμός είναι η δεύτερη φάση της ενσωμάτωσης χαρακτηριστικών ασφαλείας στον κύκλο ζωής ανάπτυξης λογισμικού. Σε αυτή τη φάση, η απαίτηση έχει μετατραπεί σε πραγματικό δομικό σχέδιο. Η φάση σχεδιασμού αντιμετωπίζεται με δύο τρόπους, ο ένας είναι η λογική μορφή και ο άλλος είναι η φυσική μορφή. Η λογική μορφή είναι η θεωρητική όψη για το πώς υποτίθεται ότι λειτουργεί το σύστημα και η φυσική μορφή είναι το πραγματικό φυσικό συστατικό του συστήματος. Γίνεται από προγραμματιστές λογισμικού, χρησιμοποιώντας διαγράμματα ροής και use cases. Η φάση σχεδιασμού του ασφαλούς SDLC αποτελείται από ενέργειες που πραγματοποιούνται πριν από τη σύνταξη κώδικα. Αφορά τον σχεδιασμό μιας αρχιτεκτονικής και στη συνέχεια την εύρεση προβλημάτων.

---

[22] "Information Systems in Your Life - GitHub Pages." https://saylordotorg.github.io/text_business-information-systems-design-an-app-for-that/s05-information-systems-in-your-li.html.

Για τον σωστό σχεδιασμό ασφαλούς SDLC, η μοντελοποίηση απειλών είναι το καλύτερο. Η μοντελοποίηση απειλών είναι η μεθοδολογία για την ανάλυση της εφαρμογής σε σχέση με επιθέσεις και τον περιορισμό αυτών των μελλοντικών επιθέσεων στο σχεδιασμό πριν από τη σύνταξη του κώδικα.

Η φάση της Ανάπτυξης

Κατά τη φάση ανάπτυξης, η ασφάλεια πρέπει να λαμβάνεται υπόψη με τις απαιτήσεις του συστήματος. Η ομάδα ασφαλείας θα πρέπει να διασφαλίσει ότι οι απαιτούμενες τεχνολογίες ασφαλείας και οι διαδικασίες είναι σωστά ενσωματωμένες και έτοιμες για δοκιμή. Αυτή η φάση είναι υπεύθυνη για τον έλεγχο των συστημάτων πληροφοριών. Η ασφάλεια ενσωματώνεται με τη σύνταξη ασφαλούς κώδικα. Αυτό μπορεί να επιτυγχάνεται από προγραμματιστές χρησιμοποιώντας ένα συνδυασμό προτύπων και αυτοματοποιημένων εργαλείων.

Ακολουθώντας τα πρότυπα, ένα συγκεκριμένο SDLC καθορίζει έναν ασφαλή οδηγό προγραμματισμού. Τα εργαλεία υλοποίησης περιέχουν λογισμικά SAST και DAST. Το Static Application Security Testing (SAST) ενεργεί σαν ορθογραφικός έλεγχος για τον κώδικα, προσδιορίζοντας πιθανές ευπάθειες στον πηγαίο κώδικα. Το Dynamic Application Security Testing (DAST) εξετάζει την εφαρμογή κατά τον χρόνο εκτέλεσης.

Η φάση της Δοκιμής

Κατά τη διάρκεια της φάσης δοκιμής, ολόκληρο το σύστημα είναι έτοιμο και δοκιμάζεται για σφάλματα. Οι προγραμματιστές πρέπει να είναι αρκετά έμπειροι και να χρησιμοποιήσουν διάφορα εργαλεία ελέγχου για να διασφαλιστεί η ασφάλεια και η ακεραιότητα των δεδομένων κατά τη φάση δοκιμής του κύκλου ζωής ανάπτυξης λογισμικού.

Οι αναγνωρισμένες δραστηριότητες δοκιμής περιλαμβάνουν λειτουργικά σχέδια δοκιμών ασφαλείας, σάρωση ευπαθειών και δοκιμή διείσδυσης. Η σάρωση ευπαθειών χρησιμοποιεί βιομηχανικού προτύπου εργαλεία για τον προσδιορισμό ευπαθειών σε επίπεδο συστήματος που υπάρχουν στην εφαρμογή.

Οι δοκιμές διείσδυσης περιλαμβάνουν ειδικούς που προσπαθούν να ξεπεράσουν αμυντικούς μηχανισμούς και να παραβιάσουν την εφαρμογή.

Η φάση της Εφαρμογής

Η φάση εφαρμογής διασφαλίζει ότι οι απαιτήσεις λογισμικού έχουν εκπληρωθεί, ελεγχθεί και το λογισμικό αποδίδει, όπως αναμενόταν. Εάν εντοπιστεί κάποιο ελάττωμα, οι προγραμματιστές το επιλύουν και θα κυκλοφορήσουν μια νέα έκδοση του λογισμικού και στη συνέχεια επαναλαμβάνουν τη φάση ελέγχου. Ο κύκλος συνεχίζεται έως ότου επιλυθούν όλα τα ελαττώματα, και το λογισμικό είναι έτοιμο για το περιβάλλον παραγωγής.

Κατά τη φάση εφαρμογής ενός ασφαλούς SDLC, αλληλεπιδρούν όλα τα στοιχεία της πλατφόρμας μεταξύ τους. Η πλατφόρμα στην οποία λειτουργεί η εφαρμογή μπορεί να έχει εκμεταλεύσιμα προβλήματα, και έτσι η ασφάλεια της πλατφόρμας δεν μπορεί να αγνοηθεί έως ότου η εφαρμογή γίνει ασφαλής. Η πλατφόρμα γίνεται ασφαλής με το κλείσιμο των ανεπιθύμητων υπηρεσιών, τη λειτουργία της εφαρμογής με τα ελάχιστα δυνατά δικαιώματα ανα χρήστη και διαβεβαιώνοντας ότι υπάρχουν άμυνες όπως IDS, τείχος προστασίας κ.λπ.

Η φάση της Συντήρησης

Η συντήρηση είναι η πιο σημαντική φάση του κύκλου ζωής ανάπτυξης, αφού οι απειλές για την ασφάλεια συνεχώς αναπτύσσονται. Το λογισμικό εφαρμογών απαιτεί τακτική συντήρηση και ενημέρωση για να παρακολουθούνται οι αλλαγές στις τεχνολογίες, ενοποίηση με νέα εργαλεία και ανάπτυξη νέων ευπαθειών. Όταν μια εφαρμογή κάνει οποιεσδήποτε αλλαγές, θα πρέπει να εφαρμόσει έλεγχο κώδικα για να διασφαλιστεί ότι η αλλαγή δεν έχει εισάγει νέες ευπάθειες στον κώδικα και το λογισμικό παραμένει ασφαλές.

Η συνεχής παρακολούθηση και η εφαρμογή πολλών συστημάτων πρόληψης εισβολής είναι απαραίτητα για την ακεραιότητα του συστήματος και των δεδομένων, τα οποία υποβάλλονται για επεξεργασία στο σύστημα. Η ομάδα συντήρησης πρέπει να εκτελεί συνεχώς δοκιμές διείσδυσης και να ελέγχει τα αρχεία καταγραφής και αναφορών.[23]

[23] "What are the Software Development Life Cycle (SDLC) phases?." 23 Αυγ. 2017, https://www.linkedin.com/pulse/what-software-development-life-cycle-sdlc-phases-private-limited.

2.5 Συμβουλές Ανάπτυξης

Όταν η εφαρμογή βρίσκεται στη φάση ανάπτυξης, οι επιχειρησιακές πολιτικές ασφαλείας και οι διαδικασίες συγκρίνονται με την υποδομή της εφαρμογής για να διαπιστωθεί αν έχουν εκπληρωθεί οι απαιτήσεις. Μερικές φορές η εφαρμογή που αναπτύχθηκε και το περιβάλλον που υπάρχει ως στόχος να χρησιμοποιηθεί η εφαρμογή δεν ταιριάζουν το ένα με το άλλο σε σχέση με τις πολιτικές ασφαλείας.

Η φάση της εφαρμογής είναι η μεγαλύτερη πρόκληση. Η διαδικασία μπορεί να είναι περίπλοκη και χωρισμένη μεταξύ διαφορετικών στοιχείων που αλληλεπιδρούν, όταν η τελικη εφαρμογή μεταφέρεται στο στάδιο της παραγωγής. Έχοντας κάποιες σκέψεις στο νου κατα τη διάρκεια της φάσης εφαρμογής μπορούν να αποφευχθούν πολλά προβλήματα. Αυτές είναι:

Απλή Εγκατάσταση
Τα αρχεία και οι κατάλογοι που εγκαθίστανται θα πρέπει να είναι μικρά και συμπιεσμένα. Αρχεία που δεν χρειάζονται, δεν πρέπει να εγκαθίστανται.

Αυτοματοποίηση
Η ανάπτυξη λογισμικού πρέπει να είναι απαλλαγμένη από ανθρώπινα σφάλματα. Υπάρχουν διάφορα εργαλεία που μπορούν να βοηθήσουν στην ανάπτυξη, χρησιμοποιώντας έναν βασικό διακομιστή build όπως το Maven ή το Bamboo για την αυτοματοποίηση της διαδικασία ή ακόμα και ένα απλό script που αντιγράφει τα αρχεία στο δίκτυο, μπορεί να είναι πολύ χρήσιμα.

Ελαχιστοποίηση Αλλαγών
Όταν κάτι στην εφαρμογή σταματάει να λειτουργεί σωστά, σημαίνει ότι έγινε κάποια αλλαγή. Όταν οι προγραμματιστές προσθέτουν κάποιες αλλαγές μέσω κάποιας αναβάθμισης λογισμικού, είναι πιο εύκολο να αναιρεθούν αυτές οι αλλαγές ή να αναγνωριστεί η ακριβής αιτία του προβλήματος.

Διαγραφή Πρώτα
Πριν την ανάπτυξη νέων χαρακτηριστικών ή πακέτων στην εφαρμογή, πρέπει πρώτα να διαγραφούν τα ήδη υπάρχοντα. Αν είναι αναγκαίο πρέπει να γίνει κάποιο αντίγραφο ασφάλειας και μετα διαγραφή του αντιγράφου εφόσον τα νέα χαρακτηριστικά λειτουργούν κανονικά.

Στρατηγική Επαναφοράς
Το να υπάρχει μια στρατηγική επαναφοράς σημαίνει ότι υπάρχει πλάνο να επανέλθει η εφαρμογή σε προηγούμενο στάδιο, αν υπάρξει κάποιο ανεπανόρθωτο πρόβλημα. Συμπεριλαμβάνει συγγραφή και δοκιμή scripts που είναι υπεύθυνα για απεγκατάσταση χαρακτηριστικών.

Ενημέρωση Ενδιαφερόμενων
Πρέπει να γίνεται ενημέρωση των χρηστών και των υπόλοιπων μελών της ομάδας για επερχόμενες αλλαγές. Είναι αναγκαίο οι προγραμματιστές να αφιερώνουν χρόνο στη συγγραφή και ενημέρωση του documentation, καθώς και να εκπαιδεύσουν το προσωπικό για τα νέα χαρακτηριστικά.

Τοπολογίες Εφαρμογής
Μία εφαρμογή Web μπορεί να είναι είτε τοπική είτε απομακρυσμένη. Αν είναι απομακρυσμένη πρέπει να γίνουν κάποιες ενέργειες όσον αφορά την ασφάλεια, από τη φάση του σχεδιασμού. Θα πρέπει να ασφαλιστεί το δίκτυο μεταξύ των servers για να αποφευχθεί τυχόν υποκλοπή και για να υπάρχει περισσότερη ιδιωτικότητα και ακεραιότητα στα ευαίσθητα δεδομένα. Επίσης θα ήταν χρήσιμο να εξεταστεί η ροή δεδομένων και λογαριασμών που χρειάζονται για αυθεντικοποίηση στο δίκτυο κατα την απομακρυσμένη σύνδεση των εξυπηρετητών. Αυτό είναι εφικτό με τη χρήση της αρχής ελάχιστων προνομίων και την εφαρμογή κατάλληλα ρυθμισμένων τειχών προστασίας.

Πολιτικές και Διαδικασίες Ασφάλειας
Οι πολιτικες και οι διαδικασίες ασφάλειας είναι σχεδιασμένες ώστε να ακολουθούνται κανόνες, ως προς το τι επιτρέπεται να κάνει η εφαρμογή και τι άδειες δίνονται στο χρήστη να κάνει. Η πολιτική ασφάλειας πρέπει να είναι διαθέσιμη, ώστε να υπάρχει σωστή στρατηγική προστασίας της εφαρμογής. Γενικά πρέπει να υπάρχουν βήματα για την αναγνώριση ευθυνών, την πρόβλεψη κινδύνων και

τον ορισμό μεθοδολογιών πρόληψης και αντιμετώπισης. Τέλος ορίζει τους κανόνες για την διασφάλιση υψηλής διαθεσιμότητας της εφαρμογής και μείωσης των αδυναμιών.

3.0 Καλές Πρακτικές Ασφαλούς Προγραμματισμού

3.1 Αξιολόγηση Εισόδου

Η αξιολόγηση εισόδου αφορά κάθε είσοδο στην εφαρμογή από έναν χρήστη ή μία άλλη εφαρμογή. Στόχος αυτής της αξιολόγησης είναι η αποφυγή αποθήκευσης ακατάλληλων δεδομένων στην βάση δεδομένων, καθώς και η εκτέλεση script κατα τη διάρκεια υποβολής κάποιας φόρμας. Παρόλα αυτά οι εφαρμογές συνήθως χρειάζονται κάποιου είδους είσοδο από το χρήστη για να παρέχουν δυναμικές λειτουργίες. Το να γίνει αντιληπτό ότι η είσοδος ενός χρήστη έχει κακόβουλους σκοπούς είναι μία αρκετά δύσκολη διαδικασία. Η καλύτερη λύση είναι να υπάρχει συνεχής έλεγχος και αξιολόγηση όλων των εισόδων στην εφαρμογή. Η είσοδος δεδομένων από μη έμπιστες πηγές θα πρέπει να αξιολογείται για την αποφυγή διαρροής μνήμης, έκθεσης του συστήματος και της βάσης δεδομένων.

Υπάρχουν δύο προσεγγίσεις, αυτή της blacklist αξιολόγησης και της whitelist αξιολόγησης. Η blacklist αξιολόγηση έχει ως σκοπό να αποκλείσει κάποιους χαρακτήρες όπως την απόστροφο (‘) ή τα tags (<>). Όμως μερικές φορές αυτός ο τύπος φιλτραρίσματος δεν λειτουργεί σωστά. Για παράδειγμα, αν κάποιος χρήστης θέλει να καταχωρήσει ένα όνομα όπως το *John O’Groats,* δεν θα είναι σε θέση να το καταχωρήσει.

Από την άλλη η whitelist αξιολόγηση είναι αρκετά πιο κατάλληλη για πεδία εισαγωγής δεδομένων. Με αυτόν τον τρόπο ο χρήστης δεν θα μπορεί να αποφύγει τους περιορισμούς που του ορίζει ο προγραμματιστής, δίνοντας ακριβώς αυτό που πρέπει να καταχωρήσει. Παρακάτω υπάρχει ένα παράδειγμα κώδικα με χρήση whitelist τεχνικής στην Java, με κανονικές εκφράσεις.

```
private static final Pattern zipPattern = Pattern.compile("^\d{5}(-\d{4})?$");

public void doPost( HttpServletRequest request, HttpServletResponse response) {
  try {
      String zipCode = request.getParameter( "zip" );
      if ( !zipPattern.matcher( zipCode ).matches()  {
          throw new YourValidationException( "Improper zipcode format." );
      }
      // do what you want here, after its been validated ..
  } catch(YourValidationException e ) {
      response.sendError( response.SC_BAD_REQUEST, e.getMessage() );
  }
}
```

Αυτός ο κώδικας αξιολογεί την παράμετρο “zip”.[24]

3.1.1 Κίνδυνοι λόγω ακατάλληλης αξιολόγησης εισόδου

Η αξιολόγηση εισόδου είναι αναγκαία για οποιαδήποτε εφαρμογή λογισμικού, λόγω κινδύνων κακόβουλης εισόδου σε διαφορετικά επίπεδα εκτέλεσης. Μία εφαρμογή που δεν αξιολογεί τα δεδομένα εισόδου είναι ευάλωτη σε κινδύνους που είναι κρίσιμοι για την ασφάλεια της εφαρμογής. Τέτοιοι κίνδυνοι παραβιάζουν άμεσα τις τρεις βασικές αρχές της ιδιωτικότητας, δηλαδή την Διαθεσιμότητα την Εμπιστευτικότητα και την Ακεραιότητα. Η σωστή αξιολόγηση των εισόδων από το χρήστη είναι ικανή να προστατεύσει την εφαρμογή από επιθέσεις όπως:

Στόχος	**Τύπος Επίθεσης**
Browsers	HTML Splitting, XFS, XSS

[24] "Input Validation - OWASP Cheat Sheet Series." https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html. Πρόσβαση στις 20 Σεπ. 2020.

Data Repositories	SQL Injection, LDAP Injection
Server Attacks	XPath Injection, XXE Injection, XML Injection
Application, Server and OS Attacks	Buffer Overflows, File Uploads

3.1.2 Τεχνικές Αξιολόγησης Δεδομένων

Οι τεχνικές αξιολόγησης δεδομένων περιλαμβάνουν διαφορετικές μεθόδους, για να αποτρέψουν τους επιτιθέμενους να εκμεταλλευτούν ευπάθειες. Οι τεχνικές αυτές είναι:

1. **Range Validation**

Η Range Validation είναι κατάλληλη για αριθμητικά δεδομένα και τιμές, καθώς και ημερομηνίες. Αρχικά αποφασίζεται ένα εύρος τιμών και μετά ακολουθούν δοκιμές με πραγματικά δεδομένα.[25]

2. **Lookup Validation**

Αυτή η τεχνική είναι κατάλληλη όταν υπάρχει ένα σύνολο τιμών για μία τιμή. Οι είσοδοι από τον χρήστη θα συγκριθούν με τις τιμές από την λίστα για να εξακριβωθεί η καταλληλότητα τους.[26]

3. **Masked Input Validation**

Σε αυτή τη μέθοδο αξιολόγησης εισόδου, επιτρέπεται συγκεκριμένη μορφή εισόδου που ταιριάζουν σε μία “μασκα”. Για παράδειγμα ένας τηλεφωνικός αριθμός με συγκεκριμένο πλήθος αριθμών ή κάποιος ταχυδρομικός κώδικας.[27]

Κάποιες βασικές στρατηγικές που συμπεριλαμβάνουν τις τεχνικές που αναφέρθηκαν είναι:

- ❏ Κωδικοποίηση των γνωστών “κακών”: Κωδικοποίηση συμβόλων που γνωρίζουμε ότι μπορούν να χρησιμοποιηθούν για κακόβουλο σκοπό.
- ❏ Άρνηση των γνωστών “κακών”: Απαγόρευση γνωστών κακόβουλων συμβόλων.
- ❏ Αποδοχή Ακριβούς Αντιστοίχισης: Αποδοχή πεπερασμένων εισόδων που περιλαμβάνονται σε λίστα.
- ❏ Γνωστών “καλών”: Αν δεν υπάρχει διαθέσιμη λίστα, χρήση ήδη γνωστής προσέγγισης.

3.1.2.1 Αξιολόγηση Εισόδου με χρήση OpenSource Frameworks και APIs

Τα Java Frameworks επιτρέπουν τον έλεγχο της συμπεριφοράς του χρήστη και βοηθούν στην εξοικονόμηση χρόνου τερματίζοντας την δημιουργία δοκιμαστικών scripts. Τα Java APIs φιλοξενούν ένα σύνολο από κλάσεις στο περιβάλλον ανάπτυξης της java. Επιτρέπει στους προγραμματιστές να εκτελούν κοινές εργασίες, όπως δικτύωση, μεταφορά αρχείων και δομών δεδομένων.

Είναι επιτακτική η ανάγκη να υπάρχει αξιολόγηση εισόδου. Κάθε εφαρμογή πρέπει να είναι σχεδιασμένη με τέτοιο τρόπο, ώστε να προστατεύεται από κακόβουλες εισόδους. Κάθε προσπάθεια να εισαχθεί κακόβουλη είσοδος πρεπει να ανιχνεύεται και να απορρίπτεται από την εφαρμογή.

Υπάρχουν διάφορες βιβλιοθήκες και frameworks που χρησιμοποιούνται ευρεία για την αξιολόγηση εισόδων στη Java. Κάποια από αυτά είναι:

---

[25] "Data Validation - Teach Computer Science." https://teachcomputerscience.com/validation/. Πρόσβαση στις 20 Σεπ. 2020.

[26] "Validation Using a Lookup Table | Importing and ... - Flylib.com." https://flylib.com/books/en/2.305.1/validation_using_a_lookup_table.html. Πρόσβαση στις 21 Σεπ. 2020.

[27] "Input Mask Format - Formidable Forms." 9 Ιουλ. 2020, https://formidableforms.com/knowledgebase/format/. Πρόσβαση στις 21 Σεπ. 2020.

- ❏ Commons Validator: Υποστηρίζει αξιολόγηση και server side και client side.
- ❏ Spring-modules-validation: Περιέχει πολλά εργαλεία και πρόσθετα για να επεκτείνουν τις δυνατότητες του Spring Framework.
- ❏ OVal: Επαληθεύει οποιουδήποτε τύπου Java object, με έλεγχο των παραμέτρων κατά την κλήση του constructor.
- ❏ iScreen: Αξιολογεί Java Objects για να επιβεβαιώσει ότι είναι έγκυρα σύμφωνα με κάποιον ορισμό, συνήθως μέσω διαμόρφωσης.
- ❏ jReform: Είναι μία βιβλιοθήκη για την επεξεργασία και την αξιολόγηση HTML φορμών, στοχεύοντας στην απλοποίηση της ανάπτυξης ιστοσελίδων μέσω αυτοματοποιημένων αξιολογήσεων εισόδου.
- ❏ JValidate: Framework που υποστηρίζει την αξιολόγηση εισόδου σε εκδόσεις Java 1.4 και πάνω. Οι περιορισμοί της εισόδου μπορούν να γίνουν με annotations, xdoclet tags ή προγραμματιστικά. Ενσωμάτωση σε web-frameworks όπως το JSF(Java Server Faces) και Wicket.
- ❏ JValidations: Είναι ένα Framework στο οποίο η αξιολόγηση γίνεται από το ίδιο το object, χωρίς καμία εξωτερική κλάση.[28]

3.1.2.2 Servlet Filters

Μία άλλη μέθοδος για αξιολόγηση της εισόδου ενός χρήστη είναι τα Servlet Filters, τα οποία είναι αποτελεσματικά και χρειάζονται μικρή παραμετροποίηση. Είναι στοιχεία της Java που λειτουργούν ως plugins που μπορούν να ανακόψουν ένα request που λαμβάνει η εφαρμογή, αλλά και ένα response προτού αποσταλεί από την εφαρμογή. Η αξιολόγηση εισόδου είναι μια συμπληρωματική μορφή αξιολόγησης, αφού μία εφαρμογή πρέπει να έχει παραπάνω από έναν μηχανισμό ασφαλείας σε περίπτωση που κάποιος δεν λειτουργήσει σωστά.

Ένα φίλτρο είναι ένα object που εκτελεί διαδικασίες φιλτραρίσματος κατά την αίτηση μιας πηγής ή κατά την απάντηση από μία πηγή. Μπορούν αυτά τα φίλτρα να χρησιμοποιηθούν για διάφορους σκοπούς, όπως για καταγραφή συμβάντων, συμπίεση δεδομένων, αυθεντικοποίηση, κωδικοποίηση ή και μετατροπή εικόνας.Παρακάτω θα χρησιμοποιηθεί ένα τέτοιο φίλτρο για την αξιολόγηση δεδομένων εισόδου. Η αξιολόγηση γίνεται πριν την επεξεργασία ενός request. Επίσης φιλτράρονται τα μηνύματα σφαλμάτων και αποθηκεύονται. Για να χρησιμοποιηθεί ένα φίλτρο πρέπει πρώτα να οριστεί και μετά να γίνει “Map” με κάποιο URL[29]. Ένα παράδειγμα “filter mapping” με χρήση του @webFilter annotation βρίσκεται παρακάτω. Όταν εφαρμόζεται το @webFilter σε ένα URL, το φίλτρο καλείται πριν την κλήση του url. Αν η είσοδος αποτύχει να περάσει το φίλτρο, τότε γίνεται ανακατεύθυνση σε μία σελίδα σφάλματος που την ορίζουμε μέσω του RequestDispatcher. Η αξιολόγηση της εισόδου γίνεται μέσω της κλάσης validationFilter.

```
@WebFilter(filterName = "ValidationFilter", urlPatterns = {"/agenda"})
```

Το @WebFilter annotation ορίζει ένα servlet filter. Το φίλτρο εφαρμόζεται στο URL. Σε αυτή την περίπτωση, καλείται πριν το agenda servlet.

```
public class ValidationFilter implements Filter {
```

Ένα φίλτρο web κάνει το φιλτράρισμα στην διεπαφή.

```
@Override
public void doFilter(ServletRequest request, ServletResponse response,
```

[28] "Open Source Validation Frameworks - Java-Source.net." https://java-source.com/open-source/validation. Πρόσβαση στις 21 Σεπ. 2020.

[29] "Define and Map Filters - Rogue Wave - Documentation." https://docs.roguewave.com/en/hydraexpress/4.3.0/html/rwsfexpservletug/4-8.html. Πρόσβαση στις 22 Σεπ. 2020.

```
        FilterChain chain)
        throws IOException, ServletException {
...
}
```

Όλη η εργασία γίνεται ουσιαστικά στην doFilter() μέθοδο.

```
String userName = request.getParameter("username");
String email = request.getParameter("email");
```

Με την getParameter() μέθοδο παίρνουμε τα δεδομένα που στάλθηκαν από την HTML φόρμα.

```
boolean valid = EmailValidator.getInstance().isValid(email);
```

Με χρήση του Apache Commons Validator's EmailValidator ελέγχουμε την εγκυρότητα του email.

```
if (userName == null || "".equals(userName)
        || email == null || "".equals(email)) {

    request.setAttribute("errMsg", "One or both fields are empty");

    RequestDispatcher rd = request.getRequestDispatcher(erpg);
    rd.include(request, response);

} else if (!valid) {

    request.setAttribute("errMsg", "Email format not valid");
    RequestDispatcher rd = request.getRequestDispatcher(erpg);
    rd.include(request, response);
} else {

    chain.doFilter(request, response);
}
```

Αν τα δεδομένα αποτύχουν να περάσουν την αξιολόγηση, η επεξεργασία περνάει στην σελίδα σφάλματος μέσω του RequestDispatcher. Διαφορετικά, το request συνεχίζει το ταξίδι του στο servlet προορισμό.[30]

3.1.3 Struts 2 Framework

Το Struts 2 Framework χρησιμοποιείται για την ανάπτυξη MVC (Model View Controller) web εφαρμογές. Είναι ο συνδυασμός του webwork framework και του Struts 1 Framework.

Struts 2 Validation

Για να αποφύγουμε λάθος τιμές εισόδου, πρέπει να γίνεται αξιολόγηση σε φόρμες που ο χρήστης εισάγει δεδομένα. Για παράδειγμα αν ο χρήστης βάζει ως ημερομηνία το Θοδωρής θα πρέπει να εμφανίζει μήνυμα σφάλματος ότι έχει βάλει λάθος ημερομηνία. Υπάρχουν τρεις τρόποι για να διεκπαιρεώσουμε αξιολόγηση εισόδου στο Struts 2:

[30] "Java validation filter - validate data with Java filter - ZetCode." 6 Ιουλ. 2020, http://zetcode.com/java/validationfilter/. Πρόσβαση στις 22 Σεπ. 2020.

1) Με Custom αξιολόγηση: Πρέπει να ενσωματωθεί το Validateable interface (ή να επεκτείνουμε την κλάση ActionSupport) και να ενσωματώσουμε την μέθοδο validate.
2) Με built-in validators (Bundled Αξιολόγηση): Το Struts 2 παρέχει πολλά προκαθορισμένα bundles που μπορούν να χρησιμοποιηθούν για να διεξαχθεί αξιολόγηση εισόδου.

Αυτοί είναι:

- ❏ requiredstring validator
- ❏ stringlength validator
- ❏ email validator
- ❏ date validator
- ❏ int validator
- ❏ double validator
- ❏ url validator
- ❏ regex validator

3) Με Ajax Validation: Αν δεν είναι επιθυμητή η ανανέωση της σελίδας, μπορεί να χρησιμοποιηθεί ο jsonValidation διαμεσολαβητής για να διεξάγουμε αξιολόγηση με ajax.

3.1.3.1 Custom Αξιολόγηση

Στο Struts 2 μπορούμε να εισάγουμε τον δικό μας custom αξιολογητή ενσωματώνοντας το Validateable interface στην action κλάση. Επίσης για τον έλεγχο σφάλματος υπάρχει ο αναχαιτιστής ροής εργασίας (Workflow interceptor) ο οποίος δεν είναι ο ίδιος υπεύθυνος για την διεξαγωγή αξιολόγησης. Η είσοδος του χρήστη είναι η προκαθορισμένη παράμετρος του που ορίζει το αποτέλεσμα που θα προκληθεί για την πράξη ή για το σφάλμα πεδίου. Υπάρχει στο defaultStack οπότε δεν χρειάζεται να το ορίσουμε ξεχωριστά.

Το Validateable interface πρέπει να ενσωματωθεί για να εφαρμοστεί η λογική της αξιολόγησης στην action κλάση. Έχει μόνο μία μέθοδο την validate() που πρέπει να γίνει override στην action κλάση για να οριστεί η λογική της αξιολόγησης.

```
public void validate();
```

Επίσης το ValidationAware interface δέχεται τα field level ή τα action class level μηνύματα σφαλμάτων. Τα field level μηνύματα φυλάσσονται στο Map και τα Action class level μηνύματα φυλάσσονται στο collection. Η εισαγωγή κάποιου μηνύματος σφάλματος ενσωματώνεται μέσω της action κλάσης.

Βήματα για να διεξαχθεί Custom αξιολόγηση

1. Δημιουργία της φόρμας που θα γίνει η λήψη της εισόδου από τον χρήστη
2. Ορισμός της λογικής αξιολόγησης στην κλάση action επεκτείνοντας την ActionSupport κλάση και overriding την μέθοδο validate.
3. Ορισμός αποτελέσματος για το μήνυμα σφάλματος από την είσοδο ονόματος στο struts.xml αρχείο**.**

Παράδειγμα custom αξιολόγησης

Δημιουργία 4 σελίδων:

1. index.jsp για την είσοδο από τον χρήστη
2. RegisterAction.java για τορ ορισμο της λογικής αξιολόγησης
3. struts.xml για τον ορισμό του αποτελέσματος και της δράσης
4. welcome.jsp για την προβολή

1) Δημιουργία index.jsp για την είσοδο

Αυτή η jsp σελίδα δημιουργεί μία φόρμα με χρήση των struts UI tags. Λαμβάνει το όνομα, τον κωδικό και το email id από το χρήστη.

index.jsp

```
<%@ taglib uri="/struts-tags" prefix="s" %>
```

```
<s:form action="register">
<s:textfield name="name" label="Name"></s:textfield>
<s:password name="password" label="Password"></s:password>
<s:submit value="register"></s:submit>
</s:form>
```

2) Δημιουργία action κλάσης

Η action κλάση κληρονομεί την ActionSupport κλάση και κάνει override την μέθοδο αξιολόγησης για να ορίσει την λογική αξιολόγησης.

RegisterAction.java

```
package com.javatpoint;
import com.opensymphony.xwork2.ActionSupport;
public class RegisterAction extends ActionSupport{
private String name,password;
public void validate() {
    if(name.length()<1)
        addFieldError("name","Name can't be blank");
    if(password.length()<6)
        addFieldError("password","Password must be greater than 5");
}

//getters and setters

public String execute(){
//perform business logic here
    return "success";
}
}
```

3) Ορισμός ένα αποτέλεσμα εισόδου στο struts.xml

Το struts.xml αρχείο ορίζει ένα επιπλέον αποτέλεσμα, που θα προκληθεί αν κάποιο μήνυμα σφάλματος βρεθεί στην κλάση action.

struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts
 Configuration 2.1//EN" "http://struts.apache.org/dtds/struts-2.1.dtd">
<struts>
<package name="default" extends="struts-default">
<action name="register" class="com.javatpoint.RegisterAction">
<result>welcome.jsp</result>
<result name="input">index.jsp</result>
</action>
</package>
</struts>
```

4) Δημιουργία στοιχείου προβολής

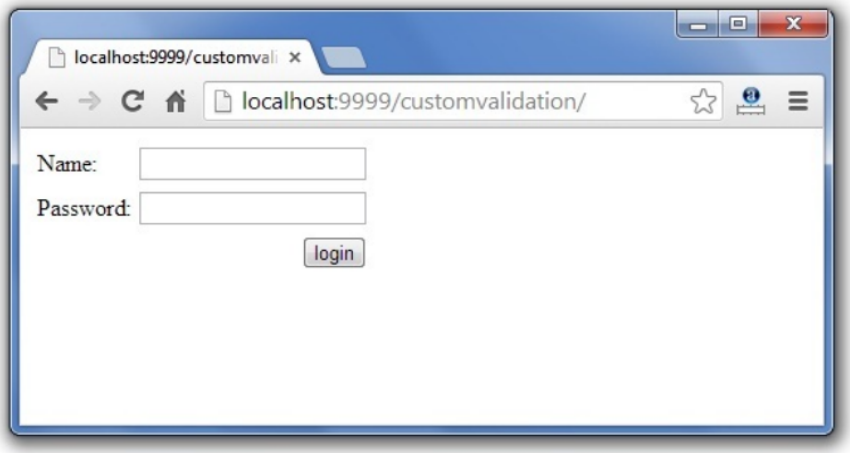
Ένα απλό jsp αρχείο για προβολή των πληροφοριών του χρήστη.

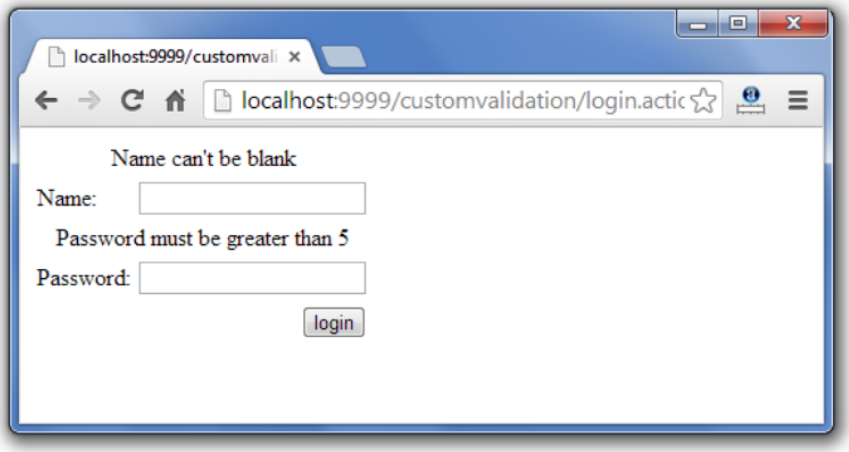
welcome.jsp

```
<%@ taglib uri="/struts-tags" prefix="s" %>
```

```
Name:<s:property value="name"/><br/>
Password:<s:property value="password"/><br/>
```

Έξοδος





Ορισμός Action Level Μήνυμα Σφάλματος

Το Action level μήνυμα σφάλματος δουλεύει για όλη την φόρμα. Υπάρχει δυνατότητα να οριστεί το μήνυμα μέσω της μεθόδου addActionError() του ValidationAware interface στην μέθοδο validate(). Για παράδειγμα:

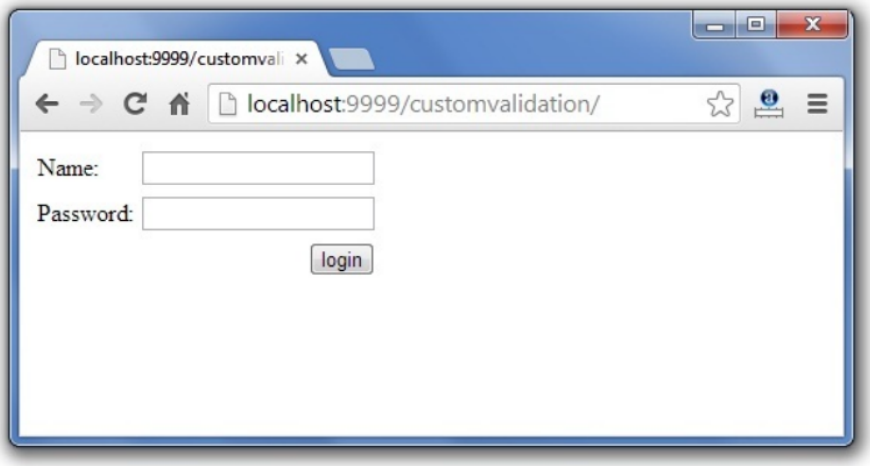
```
package com.javatpoint;
import com.opensymphony.xwork2.ActionSupport;
public class RegisterAction extends ActionSupport{
private String name,password,email;
public void validate() {
    if(name.trim().length()<1 || password.trim().length()<1){
        addActionError("Fields can't be blank");
    }
}
//getters and setters
public String execute(){
    return "success";
}
}
```

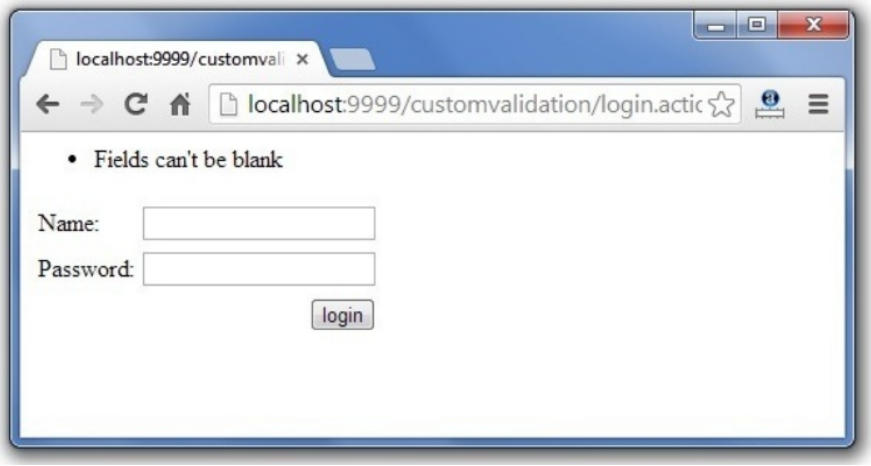
Τώρα πρέπει να χρησιμοποιηθεί το tag actionerror στο index.jsp αρχείο για να εμφανιστεί το action level μήνυμα σφάλματος.

index.jsp

```
<%@ taglib uri="/struts-tags" prefix="s" %>
<s:actionerror/>
<s:form action="register">
<s:textfield name="name" label="Name"></s:textfield>
<s:password name="password" label="Password"></s:password>
<s:textfield name="email" label="Email Id"></s:textfield>
<s:submit value="register"></s:submit>
</s:form>
```

Έξοδος





[31]

3.1.3.2 Bundled Αξιολόγηση

Το Struts 2 framework αξιολόγησης προσφέρει πολλούς built-in αξιολογητές για email,string,int, double, url, date κλπ. αξιολόγηση. Οπότε για αυτά δεν χρειάζεται να ενσωματώσουμε επιπλέον αξιολόγηση. Για επιπλέον αξιολόγηση μπορούμε να χρησιμοποιήσουμε

[31] "Struts 2 Custom Validation - Workflow Interceptor - javatpoint." https://www.javatpoint.com/struts-2-custom-validation-workflow-interceptor. Πρόσβαση στις 24 Σεπ. 2020.

τον αξιολογητή regex που θα περιγραφει παρακάτω.

Validation Interceptor
Διεξάγει αξιολόγηση με βάση τους ορισμένους κανόνες αξιολόγησης και προσθέτει field-level και action-level μηνύματα σφάλματος. Λειτουργεί σε σύνδεση με τον workflow interceptor για να εμφανίσει τα μηνύματα σφάλματος. Δεν υπάρχει κάποια παράμετρος για να οριστεί αυτός ο interceptor. Πλεονεκτεί στο γεγονός ότι βοηθάει στην γρήγορη ανάπτυξη, επειδή δεν χρειάζεται να οριστούν κοινοί αξιολογητές όπως email, date, string length κλπ. Υπάρχουν δύο τρόποι για να χρησιμοποιηθούν οι bundled αξιολογητές:

1. Plain-Validator (non-field validator) Syntax
2. Field-Validator Syntax

Ο Plain Validator μπορει να χρησιμοποιηθεί για action level αξιολόγηση. Σε τέτοια περίπτωση, ένας μόνο αξιολογητής μπορεί να εφαρμοστεί σε πολλά πεδία. Το μειονέκτημα αυτής της προσέγγισης είναι ότι δεν μπορούμε να εφαρμόσουμε πολλούς αξιολογητές σε ένα πεδίο. Παρακάτω παρατίθεται ένα παράδειγμα:

```
<validators>
    <!-- Plain-Validator Syntax -->
    <validator type="requiredstring">
        <param name="fieldName">username</param>
        <param name="trim">true</param>
        <message>username is required</message>
    </validator>
</validators>
```

Ο Field Validator μπορεί να εφαρμοστεί για field level αξιολόγηση. Σε τέτοια περίπτωση πολλαπλοί αξιολογητές μπορούν να εφαρμοστούν σε ένα πεδίο. Για παράδειγμα, μπορούμε να εφαρμόσουμε και τον requiredstring αξιολογητή και τον αξιολογητή email ταυτόχρονα στο πεδίο email. Επιπλέον, ένα πεδίο μπορεί να εμφανίσει διαφορετικά μηνύματα. Το μειονέκτημα αυτής της προσέγγισης όμως είναι ότι δεν μπορούμε να εφαρμόσουμε κοινό αξιολογητή σε πολλά πεδία όπως τον plain validator. Για παράδειγμα:

```
<validators>
    <!-- Field-Validator Syntax -->
    <field name="username">
          <field-validator type="requiredstring">
            <param name="trim">true</param>
            <message>username is required</message>
       </field-validator>
    </field>
</validators>
```

Συνιστάται η χρήση του **Field Validator** μεταξύ των δύο, διότι παρέχει μεγαλύτερη ευελιξία.

Validators

i) requiredstring validator

Αυτός ο αξιολογητής ελέγχει αν το πεδίο είναι null ή κενό. Αφαιρεί τα κενά από το πεδίο και μετά ελέγχει αν το μήκος είναι μεγαλύτερο του 0. Υπάρχουν δύο παράμετροι ορισμένες για τον requiredstring validator.

- fieldName: ορίζει το όνομα του πεδίου που θα αξιολογηθεί. Απαιτείται μόνο στον Plain Validator.
- trim: Αφαιρεί τα κενά από τα πεδία. Είναι από προεπιλογή true δηλαδή ενεργοποιημένο από προεπιλογή.

Παράδειγμα με Plain Validator και Field Level Validator:

```
<validators>
    <!-- Plain-Validator Syntax -->
    <validator type="requiredstring">
```

```
        <param name="fieldName">username</param>
        <param name="trim">true</param>
        <message>username is required</message>
    </validator>
</validators>
```

```
<validators>
    <!-- Field-Validator Syntax -->
    <field name="username">
          <field-validator type="requiredstring">
            <param name="trim">true</param>
            <message>username is required</message>
       </field-validator>
    </field>
</validators>
```

ii) stringlength validator

Ο stringlength validator ορίζει τι μήκος πρέπει να έχει η συμβολοσειρά που δίνεται. Μπορεί να χρησιμοποιηθεί για username, password κλπ. Αφαιρεί τα κενά από προεπιλογή και μετά ελέγχει, αν το μήκος είναι το επιτρεπτό. Αυτός ο αξιολογητής έχει 4 παραμέτρους:

- fieldName: ορίζει το όνομα του πεδίου που θα αξιολογηθεί. Απαιτείται μόνο στον Plain Validator.
- minLength: ορίζει το ελάχιστο μήκος. Αγνοείται από προεπιλογή.
- maxLength: ορίζει το μέγιστο μήκος. Αγνοείται από προεπιλογή.
- trim: Αφαιρεί τα κενά από τα πεδία. Είναι από προεπιλογή true δηλαδή ενεργοποιημένο από προεπιλογή.

Παράδειγμα με Plain Validator και Field Level Validator:

```
<validators>
   <!-- Plain Validator Syntax -->
    <validator type="stringlength">
         <param name="fieldName">password</param>
             <param name="minLength">6</param>
             <param name="maxLength">10</param>
             <param name="trim">true</param>
             <message>Password must be between 6 to 10 characters long</message>
         </validator>
</validators>
```

```
<validators>
    <!-- Field-Validator Syntax -->
    <field name="password">
          <field-validator type="stringlength">
          <param name="minLength">6</param>
          <param name="maxLength">10</param>
          <param name="trim">true</param>
          <message>Password must be between 6 to 10 characters long</message>
       </field-validator>
    </field>
</validators>
```

iii) email validator

Ο email validator ελέγχει, αν το δοσμένο πεδίο περιέχει έγκυρη διεύθυνση email. Λειτουργεί μόνο αν το πεδίο δεν είναι κενό. Αυτός ο αξιολογητής έχει 1 παράμετρο:

- fieldName: ορίζει το όνομα του πεδίου που θα αξιολογηθεί. Απαιτείται μόνο στον Plain Validator.

Παράδειγμα με Plain Validator και Field Level Validator:

```
<validators>
   <!-- Plain Validator Syntax -->
    <validator type="email">
         <param name="fieldName">email</param>
             <message>Please enter a valid email address</message>
         </validator>
</validators>
```

```
<validators>
    <!-- Field-Validator Syntax -->
    <field name="email">
          <field-validator type="email">
          <message>Please enter a valid email address.</message>
       </field-validator>
    </field>
</validators>
```

iv) date validator

Ο date validator ελέγχει, αν η δοσμένη ημερομηνία είναι μεταξύ του ορισμένου εύρους. Αυτός ο αξιολογητής έχει 3 παραμέτρους:

- fieldName: ορίζει το όνομα του πεδίου που θα αξιολογηθεί. Απαιτείται μόνο στον Plain Validator.
- min: ορίζει το ελάχιστο του εύρους. Αγνοείται από προεπιλογή.
- max: ορίζει το μέγιστο του εύρους. Αγνοείται από προεπιλογή.

Παράδειγμα με Plain Validator και Field Level Validator:

```
<validators>
  <!-- Plain Validator syntax -->
        <validator type="date">
      <param name="fieldName">dob</param>
          <param name="min">01/01/1980</param>
          <param name="max">01/01/2010</param>
          <message>Date of Birth must be within ${min} and ${max}</message>
    </validator>
</validators>
```

```
<validators>
   <!-- Field Validator Syntax -->
      <field name="dob">
        <field-validator type="date">
            <param name="min">01/01/1980</param>
              <param name="max">01/01/2010</param>
```

```
            <message>Date of Birth must be within ${min} and ${max}</message>
        </field>
      </field>
</validators>
```

v) int validator

Ο int validator ελέγχει αν ο δοσμένος αριθμός είναι εντός των επιτρεπτών ορίων.
Αυτός ο αξιολογητής έχει 3 παραμέτρους:

- fieldName: ορίζει το όνομα του πεδίου που θα αξιολογηθεί. Απαιτείται μόνο στον Plain Validator.
- min: ορίζει το ελάχιστο του εύρους. Αγνοείται από προεπιλογή.
- max: ορίζει το μέγιστο του εύρους. Αγνοείται από προεπιλογή.

Παράδειγμα με Plain Validator και Field Level Validator:

```
<validators>
 <!-- Plain Validator Syntax -->
          <validator type="int">
              <param name="fieldName">age</param>
              <param name="min">16</param>
              <param name="max">50</param>
              <message>Age must be between ${min} and ${max}</message>
          </validator>
</validators>
```

```
<validators>
    <!-- Field Validator Syntax -->
          <field name="age">
              <field-validator type="int">
                  <param name="min">16</param>
                  <param name="max">50</param>
                  <message>Age must be between ${min} and ${max}</message>
              </field-validator>
          </field>
</validators>
```

vi) double validator

Ο double validator ελέγχει ότι ο δοσμένος αριθμός κινητής υποδιαστολής είναι εντός του προκαθορισμένου εύρους. Μπορεί να χρησιμοποιηθεί για κάποια τιμή προϊόντος κλπ. Αυτός ο αξιολογητής έχει 3 παραμέτρους:

- fieldName: ορίζει το όνομα του πεδίου που θα αξιολογηθεί. Απαιτείται μόνο στον Plain Validator.
- minInclusive: ορίζει την ελάχιστη inclusive τιμή. Αγνοείται από προεπιλογή.
- maxInclusive: ορίζει την μέγιστη inclusive τιμή. Αγνοείται από προεπιλογή.
- minExclusive: ορίζει την ελάχιστη exclusive τιμή. Αγνοείται από προεπιλογή.
- maxExclusive: ορίζει την μέγιστη exclusive τιμή. Αγνοείται από προεπιλογή.

Παράδειγμα με Plain Validator και Field Level Validator:

```
<validators>
 <!-- Plain Validator Syntax -->
         <validator type="double">
             <param name="fieldName">price</param>
             <param name="minInclusive">100.0</param>
             <param name="maxInclusive">10000.0</param>
             <message>Price must be between ${minInclusive} and ${maxInclusive}</message>
         </validator>
</validators>
```

```
<validators>
    <!-- Field Validator Syntax -->
         <field name="price">
             <field-validator type="double">
                <param name="minInclusive">100.0</param>
                <param name="maxInclusive">10000.0</param>
                <message>Price must be between ${minInclusive} and ${maxInclusive}</message>
         </field-validator>
         </field>
</validators>
```

vii) url validator

Ο url validator ελέγχει ότι η δοσμένη τιμή είναι συμβολοσειρά και ότι είναι έγκυρο url. Μπορεί να χρησιμοποιηθεί σε url ιστοσελίδας κλπ. Αυτός ο αξιολογητής έχει 1 παράμετρο:

- fieldName: ορίζει το όνομα του πεδίου που θα αξιολογηθεί. Απαιτείται μόνο στον Plain Validator.

Παράδειγμα με Plain Validator και Field Level Validator:

```
<validators>
 <!-- Plain Validator Syntax -->
         <validator type="url">
             <param name="fieldName">website</param>
             <message>Invalid website url</message>
         </validator>
</validators>
```

```
<validators>
  <!-- Field Validator Syntax -->
  <field-validator type="url">
         <field name="website">
             <message>Invalid website url</message>
         </field>
</field-validator>
</validators>
```

viii) regex validator

Ο regex validator ελέγχει αν η δοσμένη συμβολοσειρά είναι έγκυρη με βάση κάποια συγκεκριμένη κανονική έκφραση. Μπορεί να χρησιμοποιηθεί για passwords, security keys κλπ. Αυτός ο αξιολογητής έχει 4 παραμέτρους:

- fieldName: ορίζει το όνομα του πεδίου που θα αξιολογηθεί. Απαιτείται μόνο στον Plain Validator.
- expression: ορίζει την κανονική έκφραση
- caseSensitive: ορίζει αν η έκφραση πρέπει να αντιστοιχίζεται και με case sensitive τρόπο. Είναι true από προεπιλογή.
- trim: ορίζει αν θα αφαιρεθούν τα κενά από την τιμή πριν την αντιστοίχιση. Είναι true από προεπιλογή.

Παράδειγμα με Plain Validator και Field Level Validator:

```
<validators>
 <!-- Plain Validator Syntax -->
          <validator type="regex">
              <param name="fieldName">data</param>
              <param name="expression">[A-Z,a-z,0-9]{5}</param>
          <message>data must be alpha numeric of 5 digits</message>
          </validator>
</validators>
```

```
<validators>
    <!-- Field Validator Syntax -->
         <field name="data">
        <field-validator type="regex">
          <param name="expression">[A-Z,a-z,0-9]{5}</param>
          <message>data must be alpha numeric of 5 digits</message>
        </field-validator>
    </field>
</validators>
```

[32]

3.1.3.3 Ajax Αξιολόγηση

Το Struts 2 προσφέρει υποστήριξη σε αξιολόγηση με ajax. Σε αυτή την περίπτωση, η σελίδα δεν θα ανανεωθεί, οπότε θα ανέβει η απόδοση. Γίνεται “σιωπηλά” με χρήση της javascript, δηλαδή για αξιολόγηση client side. Για να χρησιμοποιηθεί η AJAX αξιολόγηση, χρειάζεται να προστεθεί το dojo plugin στο project.

jsonValidation Interceptor

Η AJAX αξιολόγηση διεξάγεται από τον jsonValidation Interceptor. Δεν βρίσκεται μέσα στην default stack, οπότε πρέπει να τον ορίσουμε ξεχωριστά. Δεν κάνει κάποια αξιολόγηση από μόνο του, γι’ αυτό πρέπει να χρησιμοποιηθεί με validation interceptor. Βρίσκεται στο jsonValidationWorkflowStack, που περιλαμβάνει το jsonvalidation, το validation και τους workflow interceptors και την basicstack.
Τα βήματα για να γίνει AJAX αξιολόγηση είναι:

1. Δημιουργία της φόρμας που θα γίνει λήψη της εισόδου του χρήστη.
2. Κληρονόμηση της ActionSupport κλάσης στην action.
3. Ορισμός της αξιολόγησης στο validation.xml αρχείο.
4. Ορισμός result name input για το σφάλμα μηνύματος και καταχώρηση του jsonValidationWorkflowStack στο struts.xml αρχείο.

Παράδειγμα

1. index.jsp για την είσοδο του χρήστη.
2. Register.java για τη λογική της επιχείρισης.

---

[32] "Struts 2 Validation by bundled validators - javatpoint." https://www.javatpoint.com/struts-2-validation-by-bundled-validators. Πρόσβαση στις 25 Σεπ. 2020.

3. Register-validation.xml για την χρήση των bundled validators.
4. struts.xml για τον ορισμό των interceptors και αποτελεσμάτων της ενέργειας.
5. welcome.jsp για την προβολή του στοιχείου.

1) Δημιουργία index.jsp για την είσοδο
Αυτή η jsp σελίδα δημιουργεί μία φόρμα με χρήση των struts UI tags. Λαμβάνει το name, το password και το email id από τον χρήστη.

index.jsp

```
<%@ taglib  uri="/struts-tags" prefix="s"%>
<%@ taglib uri="/struts-dojo-tags" prefix="d"%>
<html>
<head>
<d:head/>
</head>
<body>
<marquee>Registration Form.....</marquee>

<s:form action="register">
<s:textfield name="name" label="Username"></s:textfield>
<s:textfield name="email" label="Email ID"></s:textfield>
<s:password name="password" label="Password"></s:password>
<d:submit validate="true" type="image" src="register-now.jpg">
</d:submit>
</s:form>

</body>
</html>
```

2) Δημιουργία της action κλάσης

Αυτή η action κλάση κληρονομεί την ActionSupport κλάση και κάνει override την μέθοδο εκτέλεσης.

RegisterAction.java

```
package mypack;

import com.opensymphony.xwork2.ActionSupport;
public class Register extends ActionSupport{
private String name,password,email;

//setters and getters

public String execute(){
    return "success";
}
}
```

3) Δημιουργία του αρχείου αξιολόγησης
Εδω, χρησιμοποιούμε τους bundled validators για να διεξάγουμε την αξιολόγηση.

Register-validation.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE validators PUBLIC
  "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
  "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">

    <validators>

    <field name="name">
    <field-validator type="requiredstring">
    <message>Name can't be blank</message>
    </field-validator>
    </field>

    <field name="email">
    <field-validator type="requiredstring">
    <message>Email ID can't be blank</message>
    </field-validator>
    <field-validator type="email">
    <message>Please enter a valid email ID</message>
    </field-validator>
    </field>

    <field name="password">
    <field-validator type="requiredstring">
    <message>Password can't be blank</message>
    </field-validator>
    <field-validator type="stringlength">
    <param name="minLength">5</param>
    <param name="maxLength">10</param>
    <message>Password can't be less than 5 or greater than 10</message>
    </field-validator>
    </field>
    </validators>
```

4) Δημιουργία του struts.xml

Αυτό το xml αρχείο ορίζει ένα επιπλέον αποτέλεσμα από το name input, και έναν interceptor jsonValdiatorWorkflowStack.

struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts
 Configuration 2.1//EN" "http://struts.apache.org/dtds/struts-2.1.dtd">
<struts>

<package name="a" extends="struts-default">

<action name="register" class="mypack.Register">
<interceptor-ref name="jsonValidationWorkflowStack"></interceptor-ref>
```

```
<result name="success">/welcome.jsp</result>
<result name="input">/index.jsp</result>
</action>

</package>
</struts>
```

5) Δημιουργία στοιχείου προβολής

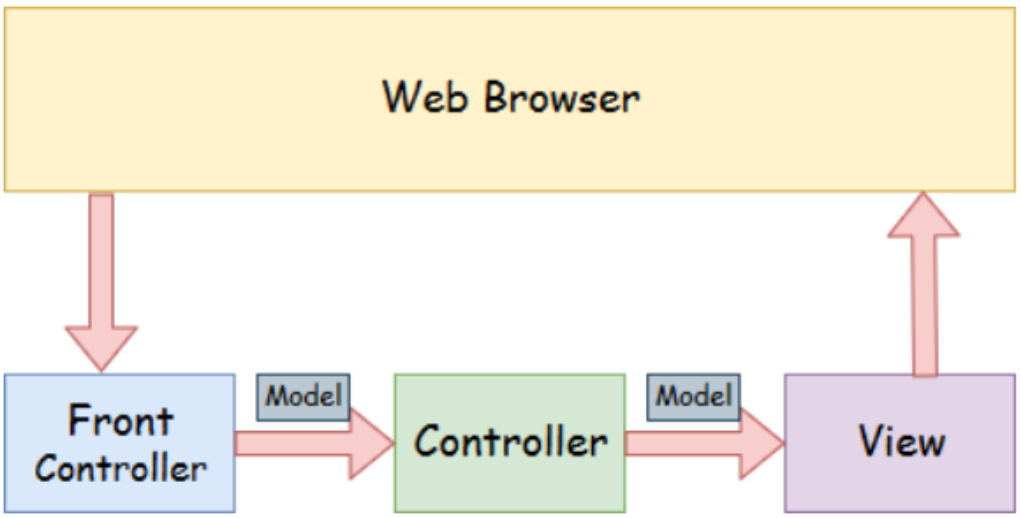
Και τέλος ένα απλό jsp αρχείο που εμφανίζει τις πληροφορίες του χρήστη.

welcome.jsp

```
<%@ taglib uri="/struts-tags" prefix="s" %>

Welcome, <s:property value="name"/>
```

3.1.4 Spring Framework

Το Spring είναι ένα ελαφρύ framework που παρέχει υποστήριξη σε πολλά άλλα frameworks όπως το Struts, Hibernate, JSF κλπ. Γενικά framework, με την ευρύτερη έννοια, είναι μία δομή όπου βρίσκει κάποιος λύση για διάφορα τεχνικά προβλήματα. Ένα Spring MVC είναι ένα Java framework που χρησιμοποιείται για την κατασκευή web εφαρμογών. Ακολουθεί το Model-View-Controller πρότυπο σχεδιασμού. Παρέχει μία κομψή λύση με τη βοήθεια του DispatcherServlet. Εδώ, το DispatcherServlet είναι μία κλάση που λαμβάνει το επερχόμενο request και το χαρτογραφεί στον σωστό πόρο όπως τους controllers, στα models και στα views.



- ➔ Model: Ένα μοντέλο περιέχει τα δεδομένα της εφαρμογής. Τα δεδομένα μπορεί να είναι ένα απλό αντικείμενο ή μία συλλογή από αντικείμενα.
- ➔ Controller: Ένας Controller περιέχει τη λογική της επιχείρησης μιας εφαρμογής.
- ➔ View: Το view αναπαριστά τις πληροφορίες που δίνονται σε κάποιο συγκεκριμένο format. Γενικά, γίνεται χρήση JSP+JSTL για την δημιουργία ενός view. Βέβαια το spring υποστηρίζει και άλλες τεχνολογίες view όπως ο Apache, Velocity, Thymeleaf και FreeMaker.

➔ Front Controller: Στο Spring Web MVC, η DispatcherServlet κλάση λειτουργεί σαν front controller. Είναι υπεύθυνο να διαχειρίζεται την ροή της Spring MVC εφαρμογής. [33]

3.1.4.1 Spring MVC Validation

Το Spring MVC Validation χρησιμοποιείται για να περιορίσει την είσοδο που δίνει ο χρήστης. Για να αξιολογήσει την είσοδο του χρήστη, το Spring έκδοσης 4 και πάνω υποστηρίζει και χρησιμοποιεί το Bean Validator API. Μπορεί να αξιολογήσει και server-side και client side εφαρμογές. Το Bean Validation API είναι ένα στοιχείο της Java που χρησιμοποιείται για να εφαρμόσει περιορισμούς στο object μοντέλο μέσω annotations. Εδώ, υπάρχει δυνατότητα να αξιολογηθεί το μήκος, ένα νούμερο, μια κανονική εκφραση κλπ. Εκτός από αυτό, γίνεται επίσης να εφαρμοστεί custom αξιολόγηση. Το Bean Validation API πρέπει να ενσωματωθεί για να λειτουργήσει. Γι'αυτό χρησιμοποιεί τον Hibernate Validator. Παρακάτω παρατίθενται μερικά συχνά χρησιμοποιούμενα annotations.

@NotNull: Ορίζει ότι η τιμή δεν μπορεί να είναι null.
@Min: Ορίζει ότι ο αριθμός πρέπει να είναι ίσος ή μεγαλύτερος από κάποια ορισμένη τιμή.
@Max: Ορίζει ότι ο αριθμός πρέπει να είναι ίσος ή μικρότερος από κάποια ορισμένη τιμή.
@Size: Ορίζει ότι το μέγεθος πρέπει να είναι ίσο με μια ορισμένη τιμή.
@Pattern: Ορίζει ότι η ακολουθία ακολουθεί μία ορισμένη κανονική έκφραση.

Για την καλύτερη κατανόηση της διαδικασίας παρακάτω παρατίθεται μία απλή φόρμα που περιέχει τα πεδία εισαγωγής δεδομένων. Χρησιμοποιείται το σύμβολο (*) για να υποδείξει ποια πεδία είναι υποχρεωτικά. Αν δεν περιέχουν τιμή παράγεται σφάλμα.

1) Προσθήκη dependencies στο pom.xml αρχείο.

pom.xml

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.1.1.RELEASE</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-jasper -->
<dependency>
    <groupId>org.apache.tomcat</groupId>
    <artifactId>tomcat-jasper</artifactId>
    <version>9.0.12</version>
</dependency>
    <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>3.0-alpha-1</version>
</dependency>
<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
 <!-- https://mvnrepository.com/artifact/org.hibernate.validator/hibernate-validator -->
<dependency>
    <groupId>org.hibernate.validator</groupId>
```

[33] "Spring MVC Tutorial - javatpoint." https://www.javatpoint.com/spring-mvc-tutorial. Πρόσβαση στις 26 Σεπ. 2020.

```
    <artifactId>hibernate-validator</artifactId>
    <version>6.0.13.Final</version>
</dependency>
```

2) Δημιουργία bean κλάσης

Employee.java

```
package com.javatpoint;
import javax.validation.constraints.Size;

public class Employee {

    private String name;
    @Size(min=1,message="required")
    private String pass;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPass() {
        return pass;
    }
    public void setPass(String pass) {
        this.pass = pass;
    }
}
```

3) Δημιουργία controller κλάσης

Στην κλάση controller:

- Το @Valid annotation εφαρμόζει κανόνες αξιολόγησης στο δοσμένο αντικείμενο.
- Το BindingResult interface περιέχει το αποτέλεσμα της αξιολόγησης.

```
package com.javatpoint;

import javax.validation.Valid;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class EmployeeController {

    @RequestMapping("/hello")
    public String display(Model m)
    {
        m.addAttribute("emp", new Employee());
        return "viewpage";
    }
```

```
    @RequestMapping("/helloagain")
    public String submitForm( @Valid @ModelAttribute("emp") Employee e, BindingResult br)
    {
        if(br.hasErrors())
        {
            return "viewpage";
        }
        else
        {
        return "final";
        }
    }
}
```

4) Παροχή καταχώρισης του controller στο web.xml αρχείο.

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
id="WebApp_ID" version="3.0">
  <display-name>SpringMVC</display-name>
   <servlet>
    <servlet-name>spring</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
</web-app>
```

5) Ορισμός του bean στο xml αρχείο

spring-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd">
    <!-- Provide support for component scanning -->
    <context:component-scan base-package="com.javatpoint" />
    <!--Provide support for conversion, formatting and validation -->
```

```
    <mvc:annotation-driven/>
    <!-- Define Spring MVC view resolver -->
     <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"></property>
        <property name="suffix" value=".jsp"></property>
     </bean>
</beans>
```

6) Δημιουργια της ζητούμενης σελίδας

index.jsp

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<html>
<body>
<a href="hello">Click here...</a>
</body>
</html>
```

7) Δημιουργία των υπόλοιπων στοιχείων προβολής

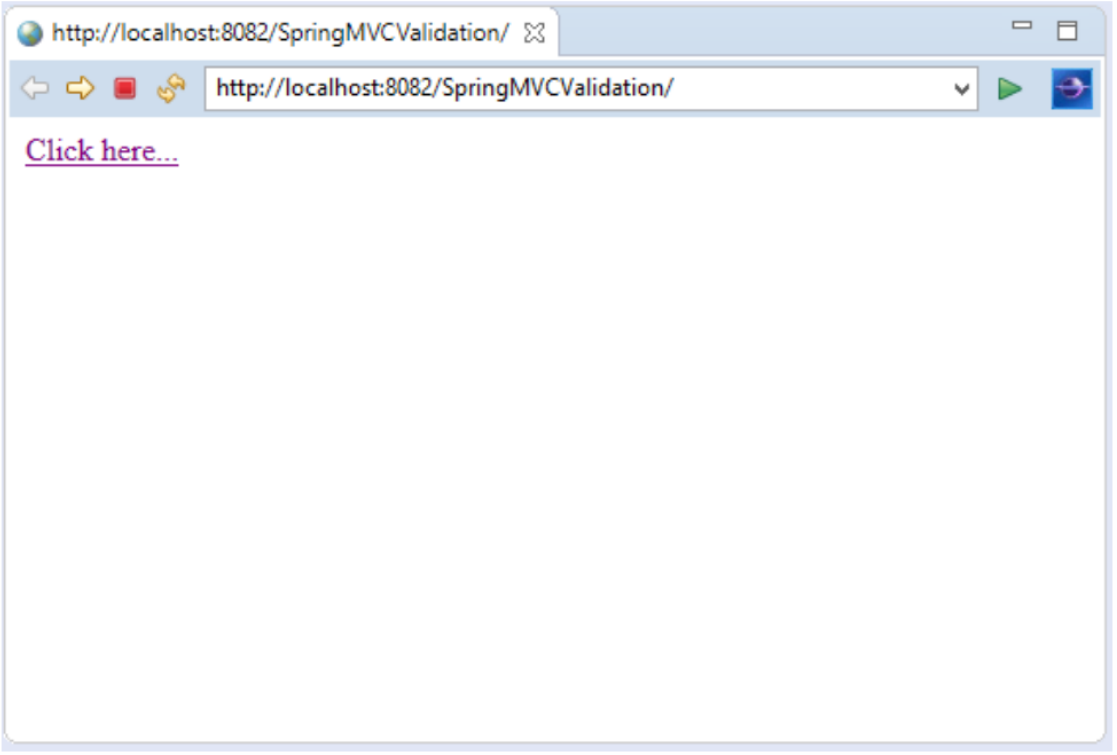
viewpage.jsp

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<html>
<head>
<style>
.error{color:red}
</style>
</head>
<body>
<form:form action="helloagain" modelAttribute="emp">
Username: <form:input path="name"/> <br><br>
Password(*): <form:password path="pass"/>
<form:errors path="pass" cssClass="error"/><br><br>
<input type="submit" value="submit">
</form:form>
</body>
</html>
```

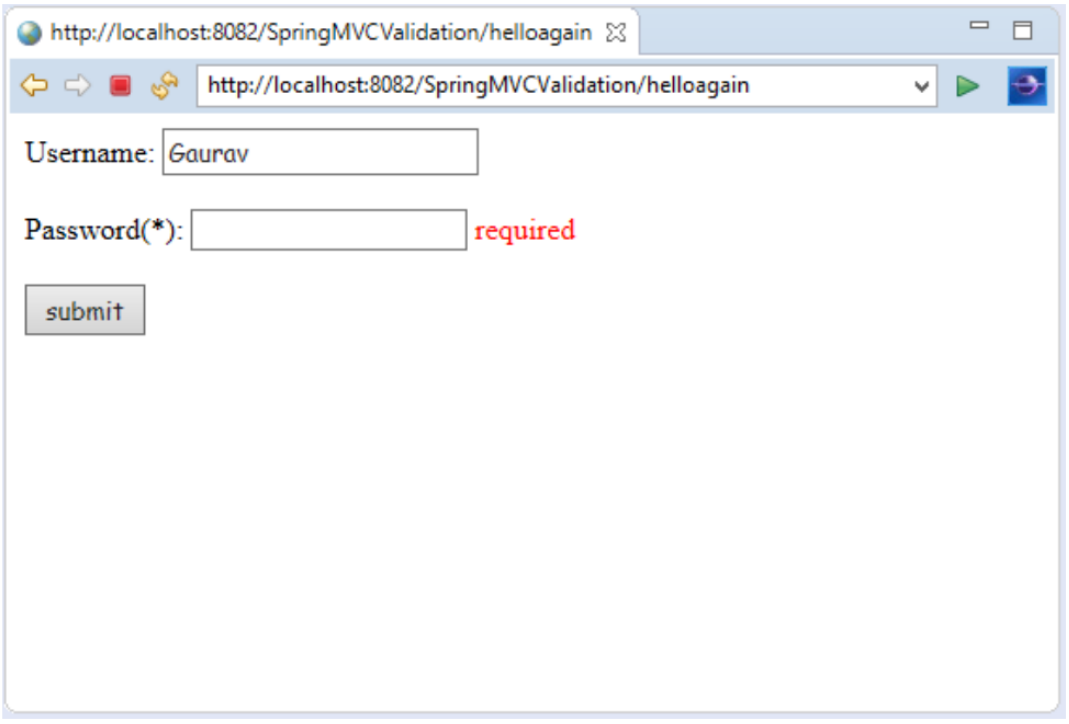
final.jsp

```
<html>
<body>
Username: ${emp.name} <br><br>
Password: ${emp.pass}
</body>
</html>
```

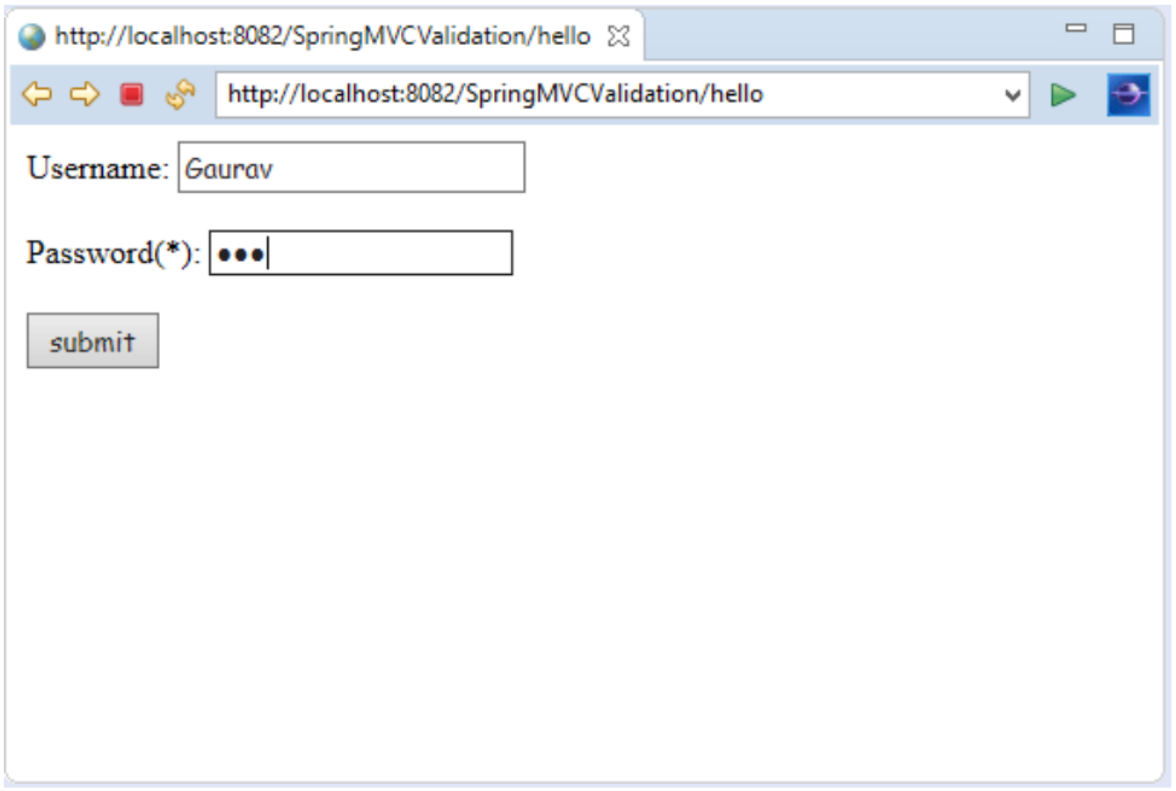
Έξοδος

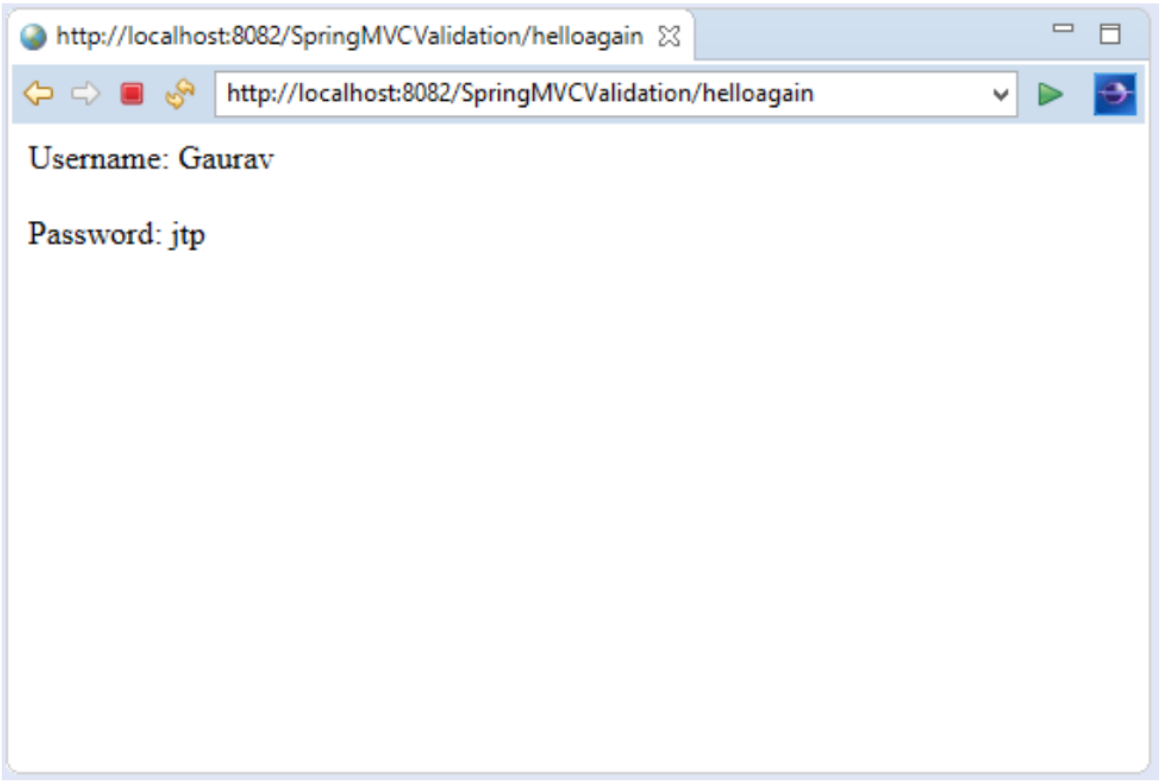


Αν η παρακάτω φόρμα καταχωρηθεί χωρίς κωδικό…



Αν καταχωρηθεί με κωδικό…





3.1.4.2 Αξιολόγηση με κανονικές εκφράσεις

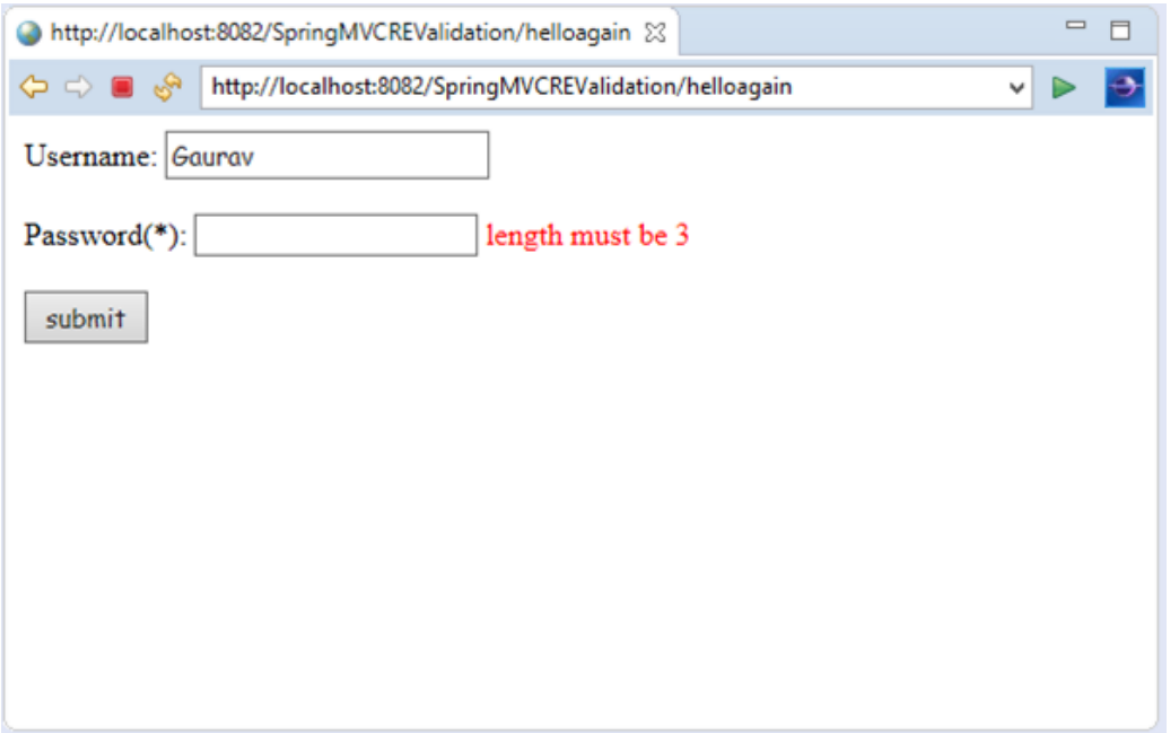
Το Spring MVC Validation επιτρέπει την αξιολόγηση της εισόδου του χρήστη σύμφωνα με μία κανονική έκφραση. Το @Pattern annotation χρησιμοποιείται για την επίτευξη αξιολόγησης με κανονικές εκφράσεις. Η κανονική έκφραση γράφεται στο regexp attribute

και περνιέται με το annotation. Η αλλαγή με το προηγούμενο παράδειγμα για να επιτευχθεί η αξιολόγηση σύμφωνα με μία κανονική έκφραση είναι μόνο στο Employee.java αρχείο, δηλαδή στην bean class.

Employee.java

```
package com.javatpoint;

import javax.validation.constraints.Pattern;
public class Employee {

    private String name;
    @Pattern(regexp="^[a-zA-Z0-9]{3}",message="length must be 3")
    private String pass;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPass() {
        return pass;
    }
    public void setPass(String pass) {
        this.pass = pass;
    }
}
```

Έξοδος



3.1.4.2 Αξιολόγηση αριθμών

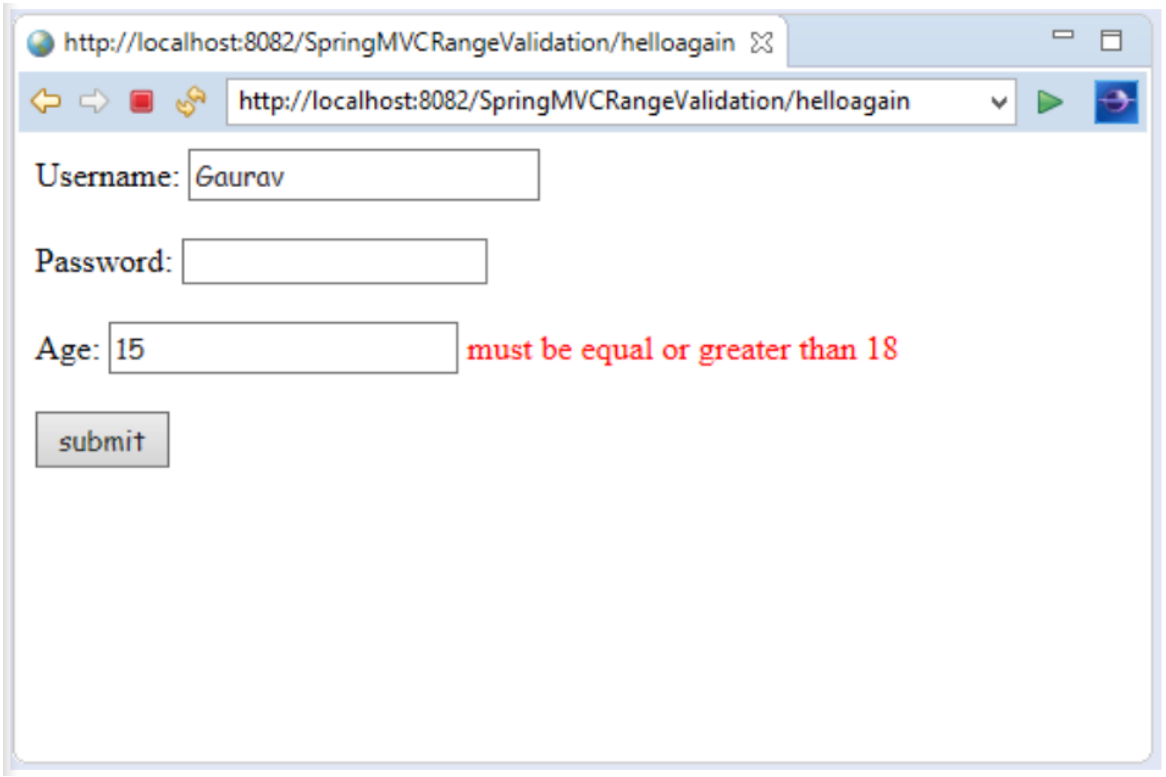
Στο Spring MVC Validation, μπορεί να αξιολογηθεί η είσοδος ενός χρήστη με βάση ένα εύρος αριθμών. Τα 2 annotations που χρησιμοποιούνται για την αξιολόγηση αριθμών είναι:

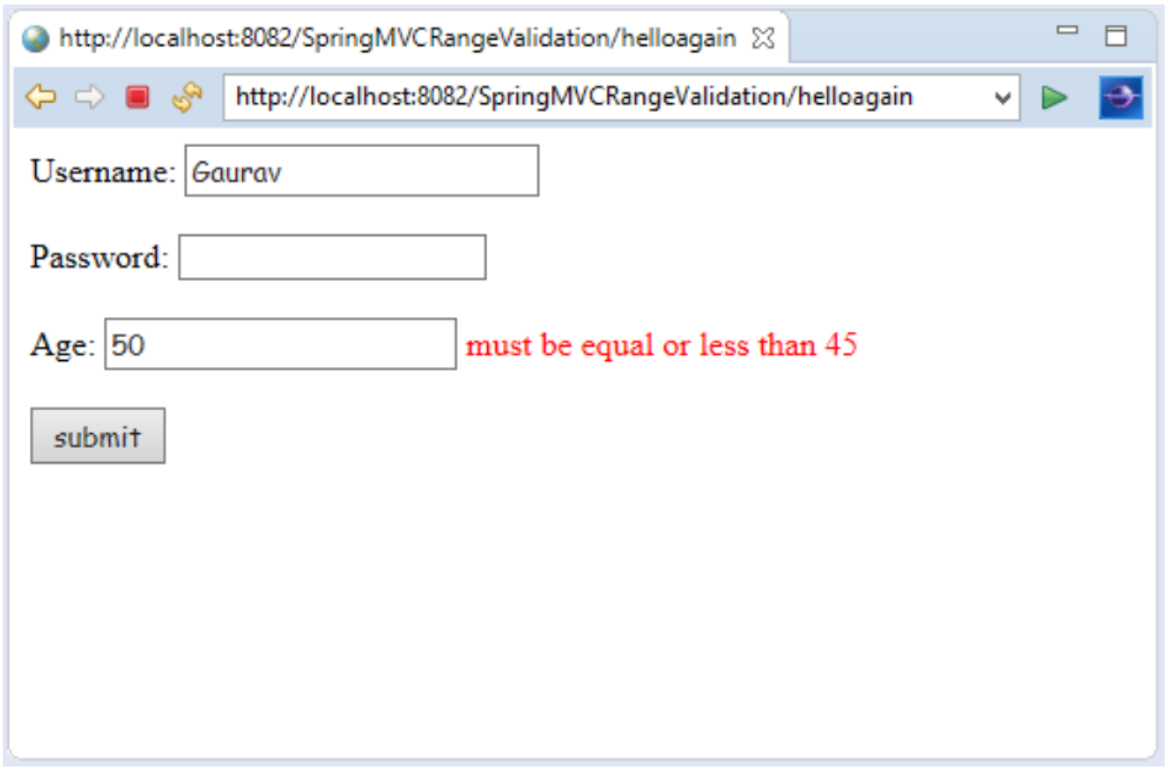
- @Min annotation - Απαιτείται να περαστεί ένας ακέραιος αριθμός με το @Min annotation. Η είσοδος του χρήστη πρέπει να είναι ίση ή μεγαλύτερη από αυτή την τιμή.
- @Max annotation - Απαιτείται να περαστεί ένας ακέραιος αριθμός με το @Max annotation. Η είσοδος του χρήστη πρέπει να είναι ίση ή μικρότερη από αυτή την τιμή.

Employee.java

```
package com.javatpoint;

import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.Size;

public class Employee {

    private String name;
    @Size(min=1,message="required")
    private String pass;

    @Min(value=18, message="must be equal or greater than 18")
    @Max(value=45, message="must be equal or less than 45")
    private int age;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPass() {
        return pass;
    }
    public void setPass(String pass) {
        this.pass = pass;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }

}
```

Έξοδος





3.1.4.3 Custom Αξιολόγηση

Το Spring MVC framework επιτρέπει την διεξαγωγή custom αξιολόγησης. Ο ορισμός των annotations γίνεται από τον προγραμματιστή. Έτσι, μπορεί να γίνει αξιολόγηση με βάση την λογική της επιχείρησης. Στο παρακάτω παράδειγμα θα γίνει χρήση και ορισμένων από πριν annotations, αλλά και custom annotations, για την αξιολόγηση της εισόδου του χρήστη.

1) Προσθήκη dependencies στο pom.xml αρχείο.

pom.xml

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.1.1.RELEASE</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-jasper -->
<dependency>
    <groupId>org.apache.tomcat</groupId>
    <artifactId>tomcat-jasper</artifactId>
    <version>9.0.12</version>
</dependency>
    <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>3.0-alpha-1</version>
</dependency>
<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
 <!-- https://mvnrepository.com/artifact/org.hibernate.validator/hibernate-validator -->
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>6.0.13.Final</version>
</dependency>
```

2) Δημιουργία της κλάσης bean

Employee.java

```
package com.javatpoint;

import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import com.javatpoint.customvalidation.Password;

public class Employee {
    private String name;
    //Custom annotation
    @Password
    private String password;
    //Predefined annotation
    @Min(value=18, message="must be equal or greater than 18")
    @Max(value=45, message="must be equal or less than 45")
    private int age;

    public String getName() {
        return name;
```

```
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

3) Δημιουργία της controller κλασης

EmployeeController.java

```
package com.javatpoint;

import javax.validation.Valid;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class EmployeeController {

    @RequestMapping("/hello")
    public String showForm(Model theModel) {

        theModel.addAttribute("emp", new Employee());

        return "viewpage";
    }

    @RequestMapping("/helloagain")
    public String processForm(
            @Valid @ModelAttribute("emp") Employee emp,
            BindingResult br) {

        if (br.hasErrors()) {
            return "viewpage";
```

```
        }
        else {
            return "final";
        }
    }
}
```

4) Δημιουργία του validator annotation

Password.java

```
package com.javatpoint.customvalidation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

import javax.validation.Constraint;
import javax.validation.Payload;

@Constraint(validatedBy = PasswordConstraintValidator.class)
@Target( { ElementType.METHOD, ElementType.FIELD } )
@Retention(RetentionPolicy.RUNTIME)
public @interface Password {
    //error message
        public String message() default "must contain jtp";
    //represents group of constraints
        public Class<?>[] groups() default {};
    //represents additional information about annotation
        public Class<? extends Payload>[] payload() default {};
}
```

5) Δημιουργία της validator κλάσης

PasswordConstraintValidator.java

```
package com.javatpoint.customvalidation;

import javax.validation.ConstraintValidator;
import javax.validation.ConstraintValidatorContext;

public class PasswordConstraintValidator implements ConstraintValidator<Password,String> {

    public boolean isValid(String s, ConstraintValidatorContext cvc) {

        boolean result=s.contains("jtp");
        return result;
    }
}
```

6) Καταχώρηση του controller στο web.xml αρχείο

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
id="WebApp_ID" version="3.0">
  <display-name>SpringMVC</display-name>
   <servlet>
    <servlet-name>spring</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
</web-app>
```

7) Ορισμός του bean στο xml αρχείο

spring-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd">
    <!-- Provide support for component scanning -->
    <context:component-scan base-package="com.javatpoint" />
    <!--Provide support for conversion, formatting and validation -->
    <mvc:annotation-driven/>
    <!-- Define Spring MVC view resolver -->
     <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"></property>
        <property name="suffix" value=".jsp"></property>
     </bean>
</beans>
```

8) Δημιουργία της ζητουμενης σελιδας

```
<html>
<body>
<a href="hello">Click here...</a>
</body>
</html>
```

9) Δημιουργία των υπόλοιπων στοιχείων προβολής

viewpage.jsp

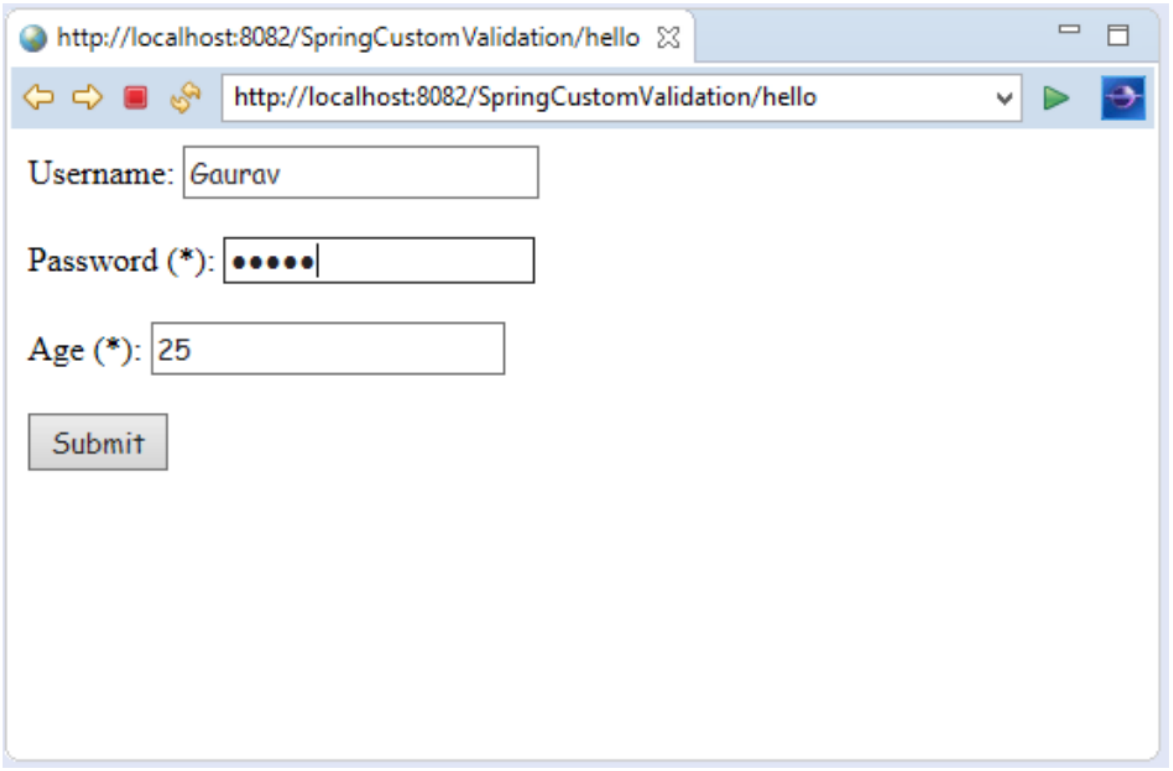
```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<html>
<head>
    <style>
        .error {color:red}
    </style>
</head>
<body>
    <form:form action="helloagain" modelAttribute="emp">
        Username: <form:input path="name" />
        <br><br>

        Password (*): <form:password path="password" />
        <form:errors path="password" cssClass="error" />
        <br><br>

        Age (*): <form:input path="age" />
        <form:errors path="age" cssClass="error" />
        <br><br>
        <input type="submit" value="Submit" />
    </form:form>
</body>
</html>
```

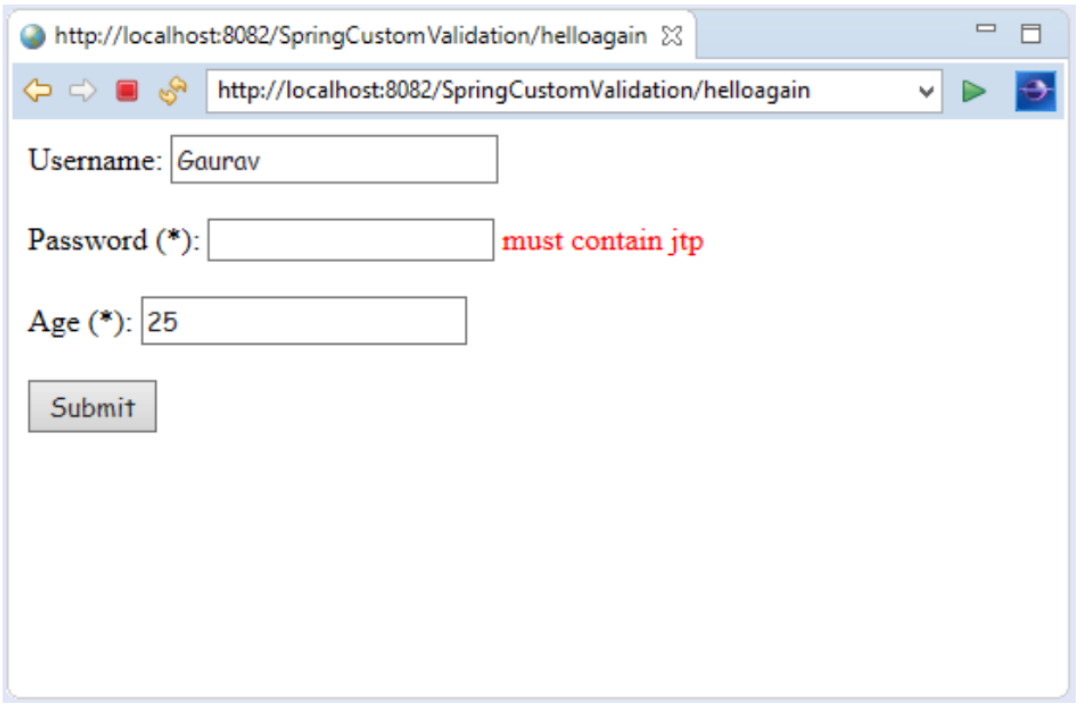
final.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
<body>
Username: ${emp.name}<br><br>
Password: ${emp.password}<br><br>
Age: ${emp.age}
<br><br>
</body>
</html>
```

Έξοδος

Είσοδος κωδικού χωρίς την ακολουθία “jtp”





3.1.5 JSF Framework

Το JSF (Java Server Faces) είναι ένα Java Framework server-side για ανάπτυξη εφαρμογών διαδικτύου. Προσφέρει ένα καλά ορισμένο μοντέλο προγραμματισμού και αποτελείται από πλούσιο API και tag βιβλιοθήκες. Το JSF API προσφέρει στοιχεία (inputText, commandButton κλπ.) και βοηθάει στην διαχείριση τους. Επίσης προσφέρει server-side αξιολόγηση, μετατροπή δεδομένων, ορισμό πλοήγησης σελίδας, επεκτασιμότητα κλπ. Οι JSF Tag βιβλιοθήκες χρησιμοποιούνται για την προσθήκη στοιχείων στις ιστοσελίδες και σύνδεση στοιχείων με στοιχεία στον server.

JSF Validation
Η JavaServer Faces τεχνολογία προσφέρει ένα σύνολο standard κλάσεων και σχετικών tags που μπορούν να χρησιμοποιηθούν για την αξιολόγηση δεδομένων.

Κλάση Αξιολόγησης	**Tag**	**Λειτουργία**
BeanValidator	validateBean	Εγγραφή bean validator σε στοιχείο
DoubleRangeValidator	validateDoubleRange	Έλεγχος αν ή τοπική τιμή ενός στοιχείου είναι εντός συγκεκριμένου εύρους ή όχι. Η τιμή πρέπει να είναι κινητής υποδιαστολής ή μετατρέψιμο σε αριθμό κινητής υποδιαστολής.
LengthValidator	validateLength	Έλεγχος αν το μήκος της τοπικής τιμής ενός στοιχείου είναι εντός συγκεκριμένου εύρους ή οχι. Η τιμή πρέπει να είναι μία java.lang.String.
LongRangeValidator	validateLongRange	Χρησιμοποιείται για να ελέγξει εάν η τοπική τιμή ενός στοιχείου βρίσκεται εντός ενός συγκεκριμένου εύρους ή όχι. Η τιμή πρέπει να είναι οποιοσδήποτε αριθμητικός τύπος ή συμβολοσειρά που μπορεί να μετατραπεί σε long.
RegexValidator	validateRegex	Χρησιμοποιείται για να ελέγξει αν η τοπική τιμή ενός στοιχείου αντιστοιχεί σε μια κανονική έκφραση από το πακέτο java.util.regex ή όχι.
RequiredValidator	validateRequired	Χρησιμοποιείται για να διασφαλιστεί ότι η τοπική τιμή δεν είναι κενή σε ένα στοιχείο EditableValueHolder.

3.1.5.1 <f:validateBean>

Χρησιμοποιείται για την καταχώρηση ενός bean αξιολογητή στο στοιχείο. Για την αξιολόγηση ενός bean μοντέλου, πρέπει να οριστεί η context παράμετρος στο αρχείο περιγραφής ανάπτυξης ιστού web.xml.

Περιορισμοί αξιολογητή Bean
Το JSF παρέχει περιορισμούς αξιολόγησης για το μοντέλο bean στη μορφή annotations. Μπορεί να τοποθετηθεί annotation σε πεδίο, σε μέθοδο ή σε κλάση ενός JavaBeans στοιχείου, όπως ένα managed bean. Επίσης το JSF παρέχει τη δυνατότητα δημιουργίας

custom ή ορισμένους από το χρήστη περιορισμούς. Οι built-in περιορισμοί είναι διαθέσιμοι στο πακέτο javax.validation.constraints και φαίνονται στο ακόλουθο πίνακα.

Περιορισμός	Περιγραφή	Παράδειγμα
@NotNull	Χρησιμοποιείται για να οριστεί ο not null περιορισμός στην τιμή του πεδίου ή της ιδιότητας.	@NotNull String username;
@Null	Χρησιμοποιείται για να οριστεί ο null περιορισμός στην τιμή του πεδίου ή της ιδιότητας.	@Null String unusedString;
@Size	Χρησιμοποιείται για τον ορισμό μεγέθους ενός πεδίου ή μιας ιδιότητας. Το μέγεθος του πεδίου ή της ιδιότητας αξιολογείται και πρέπει να αντιστοιχεί στα ορισμένα όρια. Πρέπει να γίνει χρήση ενός από τα προαιρετικά max ή min στοιχεία για τον ορισμό των ορίων.	@Size(min=2, max=240) String briefMessage;
@Digits	Χρησιμοποιείται για τον ορισμό περιορισμών όπου η τιμή του πεδίου ή της ιδιότητας πρέπει να είναι ένας αριθμός εντός ενός ορισμένου εύρους. Το στοιχείο του ακεραίου ορίζει τον μέγιστο αριθμό ακέραιων ψηφίων για τον αριθμό, στοιχείο κλάσματος ορίζει το μέγιστο κλασματικό αριθμό ψηφίων για τον αριθμό.	@Digits(integer=6, fraction=2) BigDecimal price;
@DecimalMin	Αυτός ο περιορισμός ορίζει ότι η τιμή του πεδίου ή της ιδιότητας πρέπει να είναι δεκαδική τιμή μεγαλύτερη ή ίση με την τιμή του στοιχείου.	@DecimalMin(“5.00”) BigDecimal discount;
@DecimalMax	Χρησιμοποιείται για να καθορίσει ότι η τιμή του πεδίου ή της ιδιότητας πρέπει να είναι μια δεκαδική τιμή μικρότερη ή ίση με τον αριθμό στο στοιχείο τιμής.	@DecimalMax(“30.00”) BigDecimal discount;
@Max	Χρησιμοποιείται για τον καθορισμό της τιμής του πεδίου ή της ιδιότητας που πρέπει να είναι ακέραια τιμή μικρότερη ή ίση με τον αριθμό στο στοιχείο τιμής.	@Max(10) int quantity;
@Min	Χρησιμοποιείται για τον καθορισμό της τιμής του πεδίου ή της ιδιότητας που πρέπει να είναι ακέραια τιμή μεγαλύτερη ή ίση με τον αριθμό στο στοιχείο τιμής.	@Min(5) int quantity;
@Pattern	Χρησιμοποιείται για τον ορισμό pattern που πρέπει να ταιριάζει με την κανονική έκφραση που ορίζεται στο στοιχείο regexp.	@Pattern(regexp="\\(\\d{3}\\)\\d{3}-\\d{4}") String phoneNumber;
@Past	Χρησιμοποιείται για τον καθορισμό της τιμής του πεδίου ή της ιδιότητας που πρέπει να είναι ημερομηνία στο παρελθόν.	@Past Date birthday;
@Future	Χρησιμοποιείται για τον καθορισμό της τιμής του πεδίου ή της ιδιότητας που	@Future Date eventDate;

	πρέπει να είναι ημερομηνία στο μέλλον	
@AssertTrue	Χρησιμοποιείται για τον καθορισμό της τιμής του πεδίου ή της ιδιότητας που πρέπει να ισχύει.	@AssertTrue boolean isActive;
@AssertFalse	Χρησιμοποιείται για τον καθορισμό της τιμής του πεδίου ή της ιδιότητας που πρέπει να είναι ψευδής.	@AssertFalse boolean isUnsupported;

web.xml

```
<context-param>
<param-name>
javax.faces.INTERPRET_EMPTY_STRING_SUBMITTED_VALUES_AS_NULL
</param-name>
<param-value>true</param-value>
</context-param>
```

Αυτή η τιμή παραμέτρου επιτρέπει στην εφαρμογή JavaServer Faces να αντιμετωπίζει τις κενές συμβολοσειρές ως null.

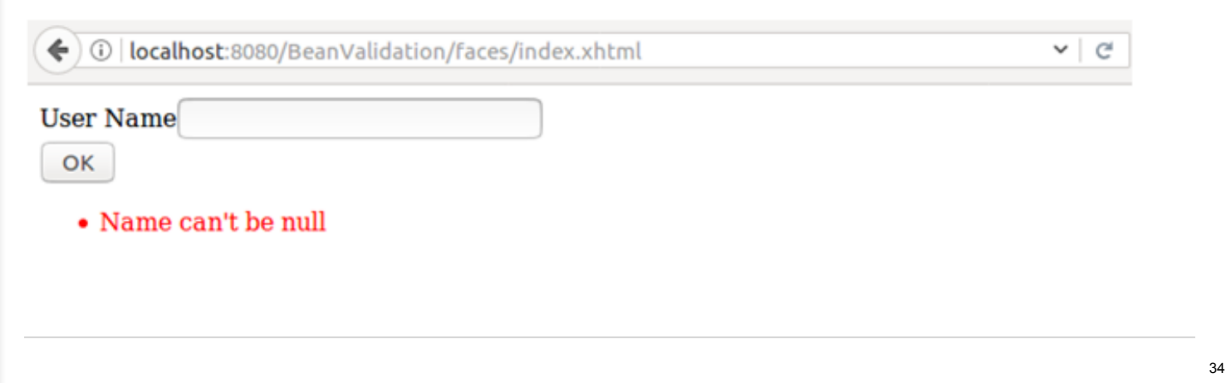
```
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import javax.validation.constraints.NotNull;
@ManagedBean
@RequestScoped
public class User{

@NotNull(message = "Name can't be null")
String name;

public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
}
```

index.html

```
<h:form id="form">
<h:outputLabel for="username">User Name</h:outputLabel>
<h:inputText id="name-id" value="#{user.name}">
<f:validateBean/>
</h:inputText><br/>
<h:commandButton value="OK" action="response.xhtml"></h:commandButton>
</h:form>
```

Έξοδος

Όταν γίνεται καταχώρηση της φόρμας αξιολογείται για nonull.



[34]

3.1.5.2 <f:validateDoubleRange>

Αυτό το tag χρησιμοποιείται για να ελέγξει αν η τιμή ενός πεδίου εισαγωγής βρίσκεται εντός συγκεκριμένου εύρους ή όχι. Η τιμή πρέπει να είναι κινητής υποδιαστολής ή μετατρέψιμη σε κινητής υποδιαστολής τιμή.

TagAttributes

Attribute	Description
minimum	Χρησιμοποιείται για τον καθορισμό ελάχιστης τιμής για αυτό το στοιχείο.
maximum	Χρησιμοποιείται για τον καθορισμό της μέγιστης τιμής για αυτό το στοιχείο.

Παράδειγμα χρήσης

Στο παρακάτω παράδειγμα, αξιολογείται η είσοδος του χρήστη που είναι ένας αριθμός κινητής υποδιαστολής, αν είναι εντός το καθορισμένου εύρους. Αυτό το πρόγραμμα θα εμφανίσει ένα μήνυμα σφάλματος, αν η είσοδος δεν περάσει την αξιολόγηση.

index.html

```
<h:outputLabel for="amount">Enter Amount </h:outputLabel>
<h:inputText id="name-id" value="#{user.amount}" validatorMessage="Please enter amount between 1000.50 and
5000.99">
<f:validateDoubleRange minimum="1000.50" maximum="5000.99"/>
</h:inputText><br/><br/>
<h:commandButton value="Submit" action="response.xhtml"></h:commandButton>
</h:form>
```

[34] "JSF f:validateBean - javatpoint." https://www.javatpoint.com/jsf-validatebean. Πρόσβαση στις 3 Οκτ. 2020.

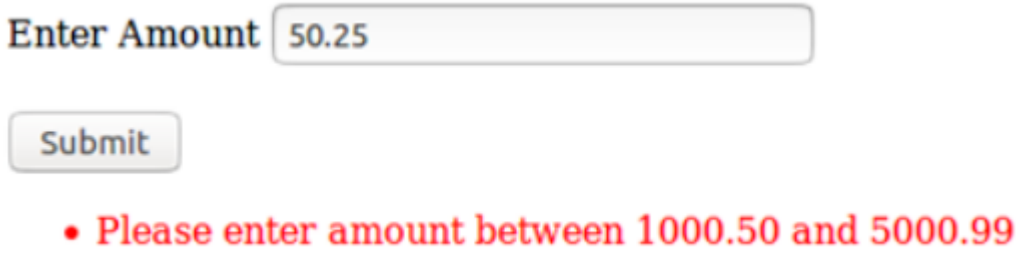
User.java

```
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
@ManagedBean
@RequestScoped
public class User{
double amount;
public double getAmount() {
return amount;
}

public void setAmount(double amount) {
this.amount = amount;
}
}
```

response.xhtml

```
<h:body>
<f:view locale="fr">
Amount entered by you: <h:outputText id = "user-name-id" value="#{user.amount}">
</h:outputText>
</f:view>
</h:body>
```

Εξοδος



3.1.5.3 <f:validateLength>

Χρησιμοποιείται για να ελέγξει εάν το μήκος της τιμής ενός στοιχείου βρίσκεται εντός ενός συγκεκριμένου εύρους ή όχι. Η τιμή πρέπει να είναι java.lang.String.

TagAttributes

Attribute	Description
minimum	Χρησιμοποιείται για τον καθορισμό ελάχιστου μήκους για το στοιχείο.
maximum	Χρησιμοποιείται για τον καθορισμό μέγιστου μήκους για το

	εξάρτημα.

index.html

```
<h:form id="user-form">
<h:outputLabel for="name">User Name</h:outputLabel><br/>
<h:inputText id="user-name" value="#{user.name}">
<f:validateLength  minimum="5"  maximum="10"/>
</h:inputText><br/>
<h:commandButton value="OK" action="response.xhtml"></h:commandButton>
</h:form>
```

User.java

```
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import javax.validation.constraints.NotNull;
@ManagedBean
@RequestScoped
public class User{
@NotNull
String name;
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
}
```

Έξοδος

3.1.5.4 <f:validateLongRange>

Χρησιμοποιείται για να ελέγξει εάν η τοπική τιμή ενός στοιχείου βρίσκεται εντός ενός συγκεκριμένου εύρους ή όχι. Η τιμή πρέπει να είναι οποιοσδήποτε αριθμητικός τύπος ή συμβολοσειρά που μπορεί να μετατραπεί σε long.

TagAttributes

Attribute	Description
minimum	Χρησιμοποιείται για τον καθορισμό ελάχιστου μήκους για το στοιχείο.
maximum	Χρησιμοποιείται για τον καθορισμό μέγιστου μήκους για το εξάρτημα.

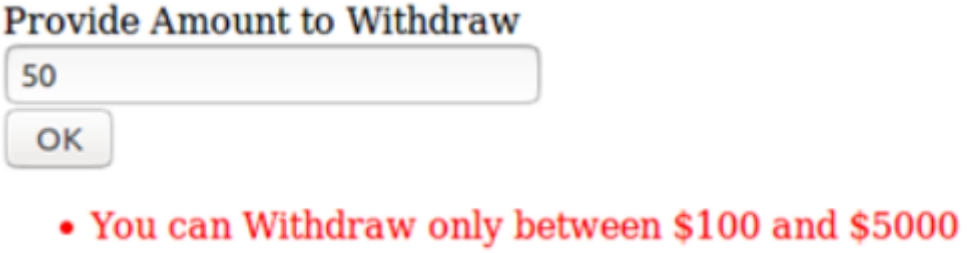
Παράδειγμα
index.html

```
<h:form id="user-form">
<h:outputLabel for="name">Provide Amount to Withdraw  </h:outputLabel><br/>
<h:inputText  id="age"  value="#{user.amount}"  validatorMessage="You  can  Withdraw  only  between  $100  and
$5000">
<f:validateLongRange minimum="100" maximum="5000" />
</h:inputText><br/>
<h:commandButton value="OK" action="response.xhtml"></h:commandButton>
</h:form>
```

User.java

```
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
@ManagedBean
@RequestScoped
public class User{
int amount;
public int getAmount() {
return amount;
}
public void setAmount(int amount) {
this.amount = amount;
}
}
```

Έξοδος



3.1.5.5 <f:validateRegex>

Χρησιμοποιείται για να ελέγξει αν η τοπική τιμή ενός στοιχείου αντιστοιχεί σε μια κανονική έκφραση από το πακέτο java.util.regex ή όχι.

TagAttributes

Attribute	Description
pattern	Χρησιμοποιείται για τον καθορισμό ενός κανονικού μοτίβου έκφρασης για το στοιχείο. Είναι ένα απαιτούμενο χαρακτηριστικό.

Παράδειγμα

index.html

```
<h:form id="form">
<h:outputLabel for="username">User Name</h:outputLabel>
<h:inputText id="name-id" value="#{user.name}" validatorMessage="Your name can have only Alphabets">
<f:validateRegex pattern="^([a-zA-Z]+(.)?[\s]*)$" />
</h:inputText><br/>
<h:commandButton value="OK" action="response.xhtml"></h:commandButton>
</h:form>
```

User.java

```
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
@ManagedBean
@RequestScoped
public class User{
String name;
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
}
```

'Εξοδος

3.1.5.6 <f:validateRequired>

Χρησιμοποιείται για να διασφαλιστεί ότι η τοπική τιμή δεν είναι κενή σε ένα στοιχείο EditableValueHolder.

TagAttributes

Attribute	Description
binding	Χρησιμοποιείται για τη δέσμευση ενός ValueExpression που αξιολογείται σε ένα instance του RequiredValidator.
for	Αυτό το χαρακτηριστικό χρησιμοποιείται για την αναφορά στην τιμή ενός από τα εκτεθειμένα συνημμένα αντικείμενα εντός του σύνθετου συστατικού εντός του οποίου είναι τοποθετημένη αυτή η ετικέτα.
id	Είναι ένα μοναδικό αναγνωριστικό συστατικών.
class	Χρησιμοποιείται για την αναπαράσταση της κλάσης CSS.

Παράδειγμα

index.html

```
<h:form id="form">
<h:outputLabel for="username">User Name</h:outputLabel>
<h:inputText id="name-id" value="#{user.name}" validatorMessage="User name is required">
<f:validateRequired />
</h:inputText><br/>
<h:commandButton value="OK" action="response.xhtml"></h:commandButton>
</h:form>
```

User.java

```
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
@ManagedBean
@RequestScoped
public class User{
String name;
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
```

```
}
}
```

Έξοδος

3.2 Αυθεντικοποίηση και Εξουσιοδότηση

3.2.1 Αυθεντικοποίηση

Η διαδικασία αναγνώρισης οποιουδήποτε χρήστη ή συσκευής σε περιβάλλον πληροφορικής είναι γνωστή ως Αυθεντικοποίηση. Η επιτυχής διαδικασία αυθεντικοποίησης παρέχει ένα συγκεκριμένο επίπεδο προνομίων στον χρήστη. Ελλείψει ελέγχου ταυτότητας, είναι δύσκολο να γίνει διάκριση μεταξύ μη αξιόπιστων και αξιόπιστων οντοτήτων, το οποίο οδηγεί σε ευάλωτο περιβάλλον. Είναι δύσκολο να γίνει εξουσιοδότηση με ασφάλεια της πρόσβασης σε διαφορετικούς κόμβους του συστήματος με αυθεντικοποίηση. Ο κύριος στόχος του ελέγχου ταυτότητας είναι να επικυρώσει την πρόσβαση σε ένα προστατευμένο τμήμα της εφαρμογής web.

Υπάρχουν διάφοροι διαθέσιμοι μηχανισμοί ελέγχου ταυτότητας και χρησιμοποιούνται ανάλογα με τις απαιτήσεις ασφαλείας του συστήματος, συμπεριλαμβανομένων βασικού ελέγχου ταυτότητας, βασισμένων σε φόρμες, δύο παραγόντων και πολλαπλών παραγόντων. Ο έλεγχος ταυτότητας με έναν παράγοντα είναι η διαδικασία χρήσης μιας μεμονωμένης μεθόδου για έλεγχο ταυτότητας. Για παράδειγμα, ένα όνομα χρήστη και ένας κωδικός πρόσβασης, ο έλεγχος ταυτότητας Multifactor είναι ένα σύστημα που χρειάζεται δύο ή περισσότερες μεθόδους για τον έλεγχο ταυτότητας. Για παράδειγμα, η πρώτη μέθοδος είναι να εισαχθεί το όνομα χρήστη και ο κωδικός πρόσβασης. Εάν είναι έγκυρα, προχωρά στη δεύτερη μέθοδο ελέγχου ταυτότητας, η οποία είναι συνήθως ένα soft token από μια εφαρμογή ασφαλείας. Σε γενικές γραμμές, η απαίτηση περισσότερων μεθόδων βελτιώνει την ασφάλεια. Θα πρέπει να χρησιμοποιούνται μέθοδοι από τουλάχιστον δύο από τους τρεις διαφορετικούς παράγοντες: κάτι που γνωρίζει κάποιος όπως ένας κωδικό πρόσβασης, κάτι που έχει όπως ένα κινητό τηλέφωνο και κάτι που είναι όπως η biometrics.

3.2.1.1 Java Container Authentication

Ενώ διασφαλίζεται η εφαρμογή ιστού, πραγματοποιείται έλεγχος ταυτότητας βάσει ρόλου, δηλαδή υπάρχουν ορισμένοι ρόλοι που εκχωρούνται σε κάθε χρήστη, όπως τα δικαιώματα για έναν προγραμματιστή είναι υψηλότερα από έναν client της εφαρμογής. Σε κάθε λογαριασμό καταχωρείται ένας ρόλος για την πρόσβαση στους πόρους της εφαρμογής ιστού όπως, προγραμματιστής, διαχειριστής ή client. Οποιοσδήποτε ρόλος μπορεί να αποκτήσει πρόσβαση μόνο όταν η ταυτότητα του συγκεκριμένου ρόλου αναγνωριστεί επιτυχώς από το container. Διαφορετικά, η πρόσβαση θα απορριφθεί, με αποτέλεσμα τον κωδικό κατάστασης HTTP 401.

Μετά την επιτυχημένη αξιολόγηση του χρήστη, οι ρόλοι εκχωρούνται μέσω κάποιας από τις παρακάτω μεθόδους:

- ❏ JDBC Module
- ❏ LDAP Login Module
- ❏ Windows Login Module
- ❏ Custom JAAS Login Module

3.2.1.2 Τύποι Αυθεντικοποίησης

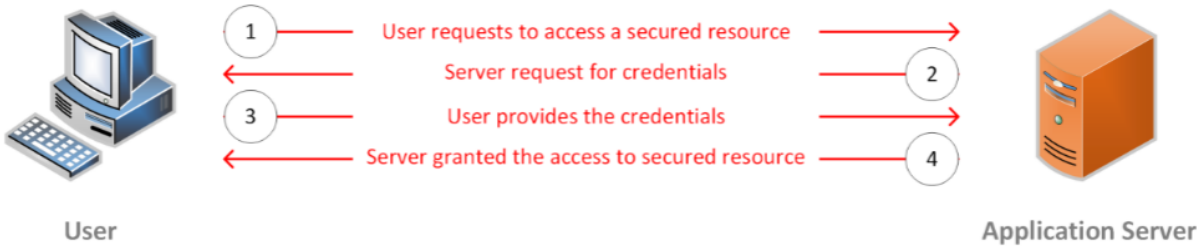
3.2.1.2.1 Βασική Αυθεντικοποίηση

Ο βασικός έλεγχος ταυτότητας χρησιμοποιεί βασική ασφάλεια HTTP. Απαιτεί ο διακομιστής να ζητήσει ένα όνομα χρήστη ή κωδικό πρόσβασης από τον πελάτη ιστού και να επικυρώσει ότι το όνομα χρήστη και ο κωδικός πρόσβασης είναι έγκυρα ή όχι, συγκρίνοντάς τα με μια βάση δεδομένων.

Όταν ορίζεται ο βασικός έλεγχος ταυτότητας, οι ενέργειες που προκύπτουν έχουν ως εξής:

1. Ένας πελάτης ζητά έναν προστατευμένο πόρο.
2. Ο διακομιστής ιστού ζητά όνομα χρήστη και κωδικό πρόσβασης.
3. Ο πελάτης στέλνει το όνομα χρήστη ή τον κωδικό πρόσβασης στον διακομιστή.
4. Ο διακομιστής επικυρώνει τον χρήστη, εάν είναι επιτυχής, επιστρέφει τον απαιτούμενο πόρο.

Ο έλεγχος ταυτότητας γίνεται μέσω SSL χρησιμοποιώντας το παράθυρο διαλόγου αντί για φόρμα HTML, χωρίς να απαιτούνται αλλαγές στις σελίδες JSP ή servlets. Το SSL / TLS πρέπει να έχει ενεργοποιηθεί ρητά αλλιώς τα διαπιστευτήρια θα μεταδοθούν σε plaintext. Είναι πιο εύκολο να εγκατασταθεί, αφού δεν χρειάζεται δημιουργία και ρύθμιση, σελίδων εισόδου και σφάλματος.



3.2.1.2.2 Form-Based Αυθεντικοποίηση

Ο έλεγχος ταυτότητας βάση φόρμας επιτρέπει στον προγραμματιστή να ελέγχει την εμφάνιση και την αίσθηση των οθονών ελέγχου ταυτότητας σύνδεσης προσαρμόζοντας την οθόνη σύνδεσης και τις σελίδες σφάλματος που παρουσιάζει ο browser στον τελικό χρήστη. Όταν δηλώνεται έλεγχος ταυτότητας βάση φόρμας, οι ενέργειες που εμφανίζονται ακολουθούν ως εξής:

1. Ένας client ζητά προστατευμένο πόρο.
2. Εάν ο client δεν έχει πιστοποιηθεί, ο διακομιστής τον ανακατευθύνει στην σελίδα σύνδεσης.
3. Ο client υποβάλλει τη φόρμα σύνδεσης στο διακομιστή Ο διακομιστής προσπαθεί να πιστοποιήσει τον χρήστη.
4. Εάν ο έλεγχος ταυτότητας επιτύχει, ο ρολος του επαληθευμένου χρήστη ελέγχεται για να επικυρωθεί ότι είναι εξουσιοδοτημένος για πρόσβαση στον πόρο. Εάν ο χρήστης έχει εξουσιοδότηση, ο διακομιστής ανακατευθύνει τον client στον πόρο.
5. Εάν ο έλεγχος ταυτότητας αποτύχει, ο client ανακατευθύνεται σε μια σελίδα σφάλματος.

Ο έλεγχος ταυτότητας βάση φόρμας είναι μια φόρμα βασισμένη σε HTML. Τα διαπιστευτήρια χρήστη και οι ρόλοι ελέγχονται για την ορθότητα τους, εάν βρεθούν λανθασμένοι, εμφανίζεται η σελίδα σφάλματος.

Το κοντέινερ χειρίζεται ολόκληρη τη διαδικασία ελέγχου ταυτότητας. Το SSL μπορεί να προστεθεί σε ένα μέρος ή σε ολόκληρη την εφαρμογή web.

Ο παρακάτω κώδικας δείχνει τον τρόπο δήλωσης ελέγχου ταυτότητας βάσει φόρμας στον deployment descriptor:

```
<login-config>
              <auth-method>FORM</auth-method>
```

```
            <realm-name>file</realm-name>
            <form-login-config>
                  <form-login-page>/logon.jsp</form-login-page>
                  <form-error-page>/logonError.jsp</form-error-page>
            </form-login-config>
</login-config>
```

3.2.1.2.3 Kerberos-Based Αυθεντικοποίηση

Το Kerberos V5 χρησιμοποιείται κυρίως ως πρωτόκολλο ασφαλείας ελέγχου ταυτότητας σε ένα domain. Υπάρχουν πολλά βήματα που πρέπει να ληφθούν για να επιτραπεί μια κλάση Java για έλεγχο ταυτότητας με Kerberos. Επαληθεύει έναν χρήστη καθώς και τον διακομιστή που παρέχει τους πόρους. Αυτό είναι γνωστό ως αμοιβαίος έλεγχος ταυτότητας.

Στην αυθεντικοποίηση με Kerberos το πιο σημαντικό είναι το useTicketCache που περιέχει την ταυτότητα του χρήστη, τον κρυπτογραφημένο κωδικό και τα κρυπτογραφημένα δεδομένα. Η διαδικασία που εκτελείται είναι αόρατη στο χρήστη, όπως και το κέντρο διαμοιρασμού κλειδιών που αποθηκεύει τις πληροφορίες λογαριασμού και τους κωδικούς των clients.[35]

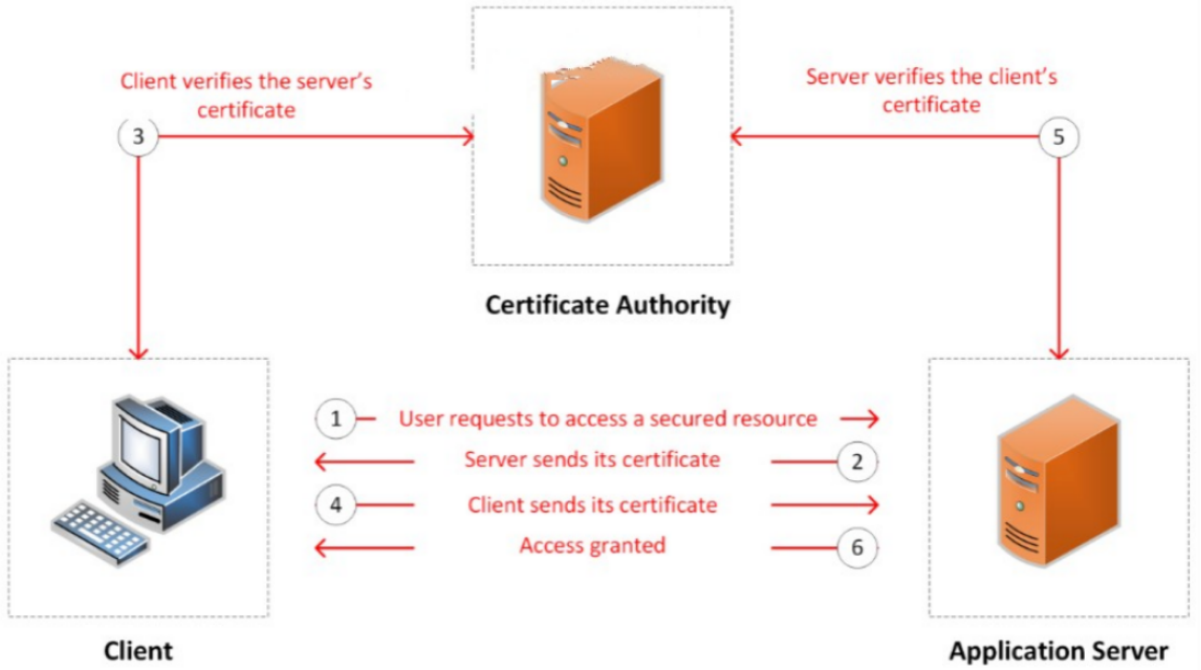
3.2.1.2.4 Client-Based Αυθεντικοποίηση

Ο έλεγχος ταυτότητας πιστοποιητικού client είναι πιο ασφαλής από τον βασικό έλεγχο ταυτότητας και τον έλεγχο ταυτότητας βάση φόρμας, επειδή αυτά τα ψηφιακά πιστοποιητικά υπογράφονται κρυπτογραφικά από τον κάτοχο. Δεν είναι εύκολη η σφυρηλάτηση ψηφιακού πιστοποιητικού. Τα πιστοποιητικά διασφαλίζουν εάν ένας χρήστης είναι νόμιμος ή όχι επικυρώνοντας το ψηφιακό πιστοποιητικό. Ο έλεγχος ταυτότητας πελάτη χρησιμοποιεί HTTP, τα προγράμματα περιήγησης και οι διακομιστές ιστού επικοινωνούν μέσω ασφαλούς κρυπτογραφημένης σύνδεσης SSL, στην οποία ο διακομιστής επικυρώνει τον πελάτη χρησιμοποιώντας το πιστοποιητικό δημοσίου κλειδιού του πελάτη. Αυτά τα ψηφιακά πιστοποιητικά υπογράφονται κρυπτογραφικά ή εκδίδονται από την Αρχή έκδοσης πιστοποιητικών (CA) όπως η VeriSign ή η Thawte. Ο client και ο server παρουσιάζουν ο ένας στον άλλον τα πιστοποιητικά τους από τα keystores τους, κατά τη διαδικασία ελέγχου ταυτότητας. Τα επικυρωμένα αντίγραφα πιστοποιητικών client και server αποθηκεύονται σε καθένα από τα keystores για ασφαλή επικοινωνία SSL.

Το SSL χρησιμοποιεί κρυπτογραφία δημόσιου κλειδιού, η οποία βασίζεται σε ζεύγη κλειδιών. Τα ζεύγη κλειδιών περιέχουν ένα δημόσιο κλειδί και ένα ιδιωτικό κλειδί. Τα δεδομένα που έχουν κρυπτογραφηθεί με ένα κλειδί μπορούν να αποκρυπτογραφηθούν μόνο με το άλλο κλειδί του ζεύγους. Ο server καθορίζει την ταυτότητα ενός επισκέπτη χρησιμοποιώντας τις ακόλουθες μεθόδους:

- ❏ Εάν το tomcatAuthentication = "false", τότε ανακτά το όνομα χρήστη και υποθέτει ότι έχει γίνει όλος ο έλεγχος ταυτότητας.
- ❏ Εάν το tomcatAuthentication = "true", το CLIENT-CERT οδηγεί στην τιμή του org.apache.Catalina.authenticator.SSLAuthenticator.

[35] "Single Sign-on Using Kerberos in Java - Oracle Help Center." https://docs.oracle.com/javase/10/security/single-sign-using-kerberos-java1.htm. Πρόσβαση στις 4 Οκτ. 2020.



Όταν χρησιμοποιείται αυτή η μέθοδος αυθεντικοποίησης, γίνονται τα παρακάτω:

1. Ο client ζητάει από το server κάποιο προστατευμένο πόρο.
2. Ο server στέλνει το πιστοποιητικό του στον client.
3. Ο client επαληθεύει το πιστοποιητικό του server από το CA.
4. Αν είναι έγκυρο, ο client στέλνει το πιστοποιητικό του στο server.
5. Ο server επαληθεύει το πιστοποιητικό του πελάτη από το CA.
6. Αν είναι έγκυρο, ο server δίνει πρόσβαση στον client.

3.2.1.3 Αδυναμίες Αυθεντικοποίησης και Πρόληψη

3.2.1.3.1 Επίθεση Ωμής Βίας

Οι επιθέσεις ωμής βίας είναι οι πιο δημοφιλείς μέθοδοι cracking password που χρησιμοποιούνται για να σπάσουν τον μηχανισμό ασφαλείας. Σε αυτήν την τεχνική, ο εισβολέας προσπαθεί όλους τους πιθανούς συνδυασμούς κωδικών πρόσβασης και PINs ασφαλείας.

Στην παραδοσιακή επίθεση brute-force, ο εισβολέας δοκιμάζει απλώς το συνδυασμό γραμμάτων και αριθμών για να δημιουργήσει διαδοχικούς κωδικούς πρόσβασης. Ωστόσο, αυτή η παραδοσιακή τεχνική θα διαρκέσει περισσότερο όταν ο κωδικός πρόσβασης είναι αρκετά μεγάλος. Αυτές οι επιθέσεις μπορεί να διαρκέσουν αρκετά λεπτά έως αρκετές ώρες, ανάλογα με το σύστημα που χρησιμοποιείται και το μήκος του κωδικού πρόσβασης. Είναι μια συνεχής μέθοδος δοκιμής και σφάλματος των προσπαθειών σύνδεσης σε μια εφαρμογή Ιστού χρησιμοποιώντας λεξικό και άλλους συνδυασμούς χαρακτήρων.

Προληπτικά μέτρα:

- ❏ Επιβολή στους χρήστες, εγγραφής με μεγάλο σε μήκος κωδικό πρόσβασης.
- ❏ Επιβολή χρήσης αλφαριθμητικών χαρακτήρων και συμβόλων στον κωδικό.
- ❏ Case sensitive κωδικός πρόσβασης.
- ❏ Κλείδωμα λογαριασμού μετά από ορισμένο αριθμό αποτυχημένων προσπαθειών.
- ❏ Επιβολή πολιτικής αλλαγής κωδικού μετά από ορισμένο χρονικό διάστημα.

3.2.1.3.2 Web-Based Enumeration επίθεση

Ο στόχος του Web-based enumeration είναι η συλλογή ενός συνόλου από usernames και passwords, αλληλεπιδρόντας ενεργά με την αυθεντικοποίηση της εφαρμογής. Αυτή η επίθεση είναι επίσης χρήσιμη για την επίθεση ωμής βίας, στην οποία ο επιτιθέμενος μπορεί να επικυρώσει αν τα credentials είναι έγκυρα ή όχι. Η ευπάθεια που πρέπει να διορθωθεί είναι η εμφάνιση μηνύματος για το αν ένα username υπάρχει στη βάση δεδομένων της εφαρμογής ή όχι. Ομοίως, μερικές φορές οι εφαρμογές εμφανίζουν για το αν το username ή το password είναι λανθασμένο.

Δύο από τις πιο κοινά σημεία που συμβαίνει user enumeration είναι το login page της εφαρμογής και η Forgot Password Page. Ο επιτιθέμενος πειραματίζεται με πολλά userID credentials και αναλύει τα responses για να βρει ποια είναι έγκυρα μέσω μηνύματα σφάλματος.

Προληπτικά μέτρα:

- ❏ Οι εφαρμογές web πρέπει να εμφανίζουν το ίδιο μήνυμα σφάλματος σε όλες τις αποτυχίες. Για παράδειγμα να εμφανίζει γενικό μήνυμα, ότι τα credentials είναι λανθασμένα.
- ❏ Ανάλυση URLs, web page titles και τα responses τους κατα την εμφάνιση σφαλμάτων και πρόληψη διαρροής περιττών δεδομένων.

3.2.1.3.3 Επίθεση αδύναμου password

Κάθε νέος εξοπλισμός διαμορφώνεται με έναν προεπιλεγμένο κωδικό πρόσβασης από τους κατασκευαστές. Συνιστάται να αλλάξετε τον προεπιλεγμένο κωδικό πρόσβασης σε ένα μοναδικό, μυστικό σύνολο χαρακτήρων. Ένας εισβολέας που χρησιμοποιεί προεπιλεγμένους κωδικούς πρόσβασης πραγματοποιώντας αναζήτηση μέσω του επίσημου ιστότοπου του κατασκευαστή συσκευών ή μέσω διαδικτυακών εργαλείων για την αναζήτηση προεπιλεγμένων κωδικών πρόσβασης μπορεί να επιχειρήσει αυτόν τον τύπο επίθεσης. Παρακάτω είναι η λίστα των διαθέσιμων εργαλείων για την αναζήτηση προεπιλεγμένου κωδικού πρόσβασης.

- ❖ https://cirt.net/
- ❖ https://default-password.info/
- ❖ http://www.passwordsdatabase.com/

Προληπτικά μέτρα:

- ❏ Δεν πρέπει να γίνεται χρήση προεπιλεγμένων κωδικών. Πρέπει να γίνεται αλλαγή όσο το δυνατόν γρηγορότερα.
- ❏ Δεν πρέπει να γίνεται χρήση μαντέψιμων κωδικών πρόσβασης.
- ❏ Πρέπει να γίνεται χρήση μεγάλων κωδικών με αριθμούς, χαρακτήρες και σύμβολα.
- ❏ Πρέπει να χρησιμοποιούνται στους κωδικού πρόσβασης και κεφαλαία και πεζά γράμματα.
- ❏ Πρέπει να υπάρχει πολιτική παλιού κωδικού.
- ❏ Αποφυγή διαρροής πληροφοριών από μηνύματα λανθασμένης αυθεντικοποίησης.
- ❏ Πρέπει να ενσωματωθεί πολιτική κλειδώματος λογαριασμού, μετά από ορισμένο πλήθος αποτυχημένων προσπαθειών αυθεντικοποίησης.

3.2.2 Εξουσιοδότηση

Η εξουσιοδότηση είναι μια συνάρτηση παροχής των δικαιωμάτων και των προνομίων στην επιτυχώς επικυρωμένη ταυτότητα για πρόσβαση σε συγκεκριμένο σύνολο πόρων. Για να περιορίσουν τη λειτουργία της εξουσιοδότησης, οι προγραμματιστές ορίζουν μια πολιτική πρόσβασης όπου διαφορετικοί χρήστες ή ρόλοι σχετίζονται με διαφορετικά επίπεδα εξουσιοδότησης προκειμένου να τους περιορίσουν να έχουν πρόσβαση στους πόρους τους.

3.2.2.1 Μοντέλο ελέγχου πρόσβασης

Κατά την εκτέλεση μιας εφαρμογής, τα δικαιώματα ελέγχονται καλώντας τη μέθοδο SecurityManager.checkPermission(), διασφαλίζοντας ότι δίνονται κατάλληλα δικαιώματα στα αντικείμενα. Κατά την επικύρωση των δικαιωμάτων, το SecurityManager.checkPermission() καλεί το AccessController.checkPermission() για να επαληθεύσει αυτά τα δικαιώματα.

Ο έλεγχος ταυτότητας και η εξουσιοδότηση διαδραματίζουν σημαντικό ρόλο στην ασφάλεια πληροφοριών και από κοινού αποτελούν τον έλεγχο πρόσβασης. Τα τρία μοντέλα ελέγχου πρόσβασης είναι:

1. Discretionary access control (DAC)
2. Mandatory access control (MAC)
3. Role-based access control (RBAC)

Discretionary access control (DAC)

Ο Discretionary access control (DAC) είναι ένας τύπος ελέγχου πρόσβασης ασφάλειας που παραχωρεί ή περιορίζει την πρόσβαση αντικειμένων μέσω μιας πολιτικής πρόσβασης που καθορίζεται από την ομάδα κατόχου ενός αντικειμένου. Τα στοιχεία ελέγχου του μηχανισμού DAC ορίζονται από την ταυτοποίηση χρήστη με παρεχόμενα διαπιστευτήρια κατά τον έλεγχο ταυτότητας, όπως όνομα χρήστη και κωδικό πρόσβασης. Το άτομο (κάτοχος) μπορεί να μεταφέρει επαληθευμένα αντικείμενα ή πρόσβαση σε πληροφορίες σε άλλους χρήστες. Με άλλα λόγια, ο κάτοχος καθορίζει τα δικαιώματα πρόσβασης αντικειμένων.

Ένα τυπικό παράδειγμα DAC είναι το filemode του Unix, το οποίο καθορίζει τα δικαιώματα ανάγνωσης, εγγραφής και εκτέλεσης σε καθένα από τα τρία bits για κάθε χρήστη, ομάδα και άλλα.

Τα χαρακτηριστικά DAC περιλαμβάνουν:

- ❏ Ο χρήστης μπορεί να μεταβιβάσει την ιδιοκτησία αντικειμένου σε άλλον χρήστη.
- ❏ Ο χρήστης μπορεί να καθορίσει τον τύπο πρόσβασης άλλων χρηστών.
- ❏ Μετά από αρκετές προσπάθειες, οι αποτυχίες εξουσιοδότησης περιορίζουν την πρόσβαση των χρηστών.
- ❏ Οι μη εξουσιοδοτημένοι χρήστες δεν έχουν τα χαρακτηριστικά των αντικειμένων, όπως μέγεθος αρχείου, όνομα αρχείου και διαδρομή καταλόγου.
- ❏ Η πρόσβαση στο αντικείμενο καθορίζεται κατά την εξουσιοδότηση της λίστας ελέγχου πρόσβασης (ACL) και βασίζεται στην αναγνώριση χρήστη ή / και στην ιδιότητα μέλους ομάδας.

Το DAC είναι εύκολο να εφαρμοστεί και διαισθητικό, αλλά έχει ορισμένα μειονεκτήματα, όπως:

- ❏ Έμφυτες ευπάθειες (Δούρειος ίππος)
- ❏ Συντήρηση ACL
- ❏ Συντήρηση εκχώρησης και ανάκλησης δικαιωμάτων
- ❏ Περιορισμένη αρνητική ισχύς εξουσιοδότησης[36]

Mandatory access control (MAC)

Ο υποχρεωτικός έλεγχος πρόσβασης (MAC) είναι μια στρατηγική ασφαλείας που περιορίζει την ικανότητα των μεμονωμένων κατόχων πόρων να παραχωρούν ή να αρνούνται την πρόσβαση σε αντικείμενα πόρων σε ένα σύστημα αρχείων. Τα κριτήρια MAC καθορίζονται από τον διαχειριστή του συστήματος, που επιβάλλονται αυστηρά από το λειτουργικό σύστημα (OS) ή τον πυρήνα ασφαλείας και δεν μπορούν να τροποποιηθούν από τους τελικούς χρήστες.

Συχνά χρησιμοποιούμενος σε κυβερνητικές και στρατιωτικές εγκαταστάσεις, ο υποχρεωτικός έλεγχος πρόσβασης λειτουργεί εκχωρώντας μια ετικέτα ταξινόμησης σε κάθε αντικείμενο συστήματος αρχείων. Οι ταξινομήσεις περιλαμβάνουν εμπιστευτικό, μυστικό και απόρρητο. Σε κάθε χρήστη και συσκευή στο σύστημα εκχωρείται παρόμοιο επίπεδο ταξινόμησης και εκκαθάρισης. Όταν ένα άτομο

[36] "What is Discretionary Access Control (DAC)? - Techopedia." https://www.techopedia.com/definition/229/discretionary-access-control-dac. Πρόσβαση στις 6 Οκτ. 2020.

ή συσκευή προσπαθεί να αποκτήσει πρόσβαση σε έναν συγκεκριμένο πόρο, το λειτουργικό σύστημα ή ο πυρήνας ασφαλείας θα ελέγξει τα διαπιστευτήρια της οντότητας για να προσδιορίσει εάν θα παραχωρηθεί πρόσβαση. Ενώ είναι η πιο ασφαλής ρύθμιση ελέγχου πρόσβασης που διατίθεται, το MAC απαιτεί προσεκτικό σχεδιασμό και συνεχή παρακολούθηση για την ενημέρωση όλων των ταξινομήσεων των αντικειμένων πόρων και των χρηστών.[37]

Role-based access control (RBAC)

Ο έλεγχος πρόσβασης βάση ρόλου (RBAC) είναι μια μέθοδος περιορισμού της πρόσβασης στο δίκτυο βάση των ρόλων των μεμονωμένων χρηστών σε μια επιχείρηση. Το RBAC επιτρέπει στους υπαλλήλους να έχουν δικαιώματα πρόσβασης μόνο στις πληροφορίες που χρειάζονται για να κάνουν τη δουλειά τους και τους εμποδίζει να έχουν πρόσβαση σε πληροφορίες που δεν σχετίζονται με αυτούς. Ο ρόλος ενός υπαλλήλου σε έναν οργανισμό καθορίζει τις άδειες που χορηγεί το άτομο και διασφαλίζει ότι οι υπάλληλοι χαμηλότερου επιπέδου δεν μπορούν να έχουν πρόσβαση σε ευαίσθητες πληροφορίες ή να εκτελούν εργασίες υψηλού επιπέδου.

Στο μοντέλο δεδομένων ελέγχου πρόσβασης βάση ρόλων, οι ρόλοι βασίζονται σε διάφορους παράγοντες, συμπεριλαμβανομένης της εξουσιοδότησης, της ευθύνης και της ικανότητας εργασίας. Ως εκ τούτου, οι εταιρείες μπορούν να ορίσουν εάν ένας χρήστης είναι τελικός χρήστης, διαχειριστής ή εξειδικευμένος χρήστης. Επιπλέον, η πρόσβαση σε υπολογιστικούς πόρους μπορεί να περιορίζεται σε συγκεκριμένες εργασίες, όπως η δυνατότητα προβολής, δημιουργίας ή τροποποίησης αρχείων. Ο περιορισμός της πρόσβασης στο δίκτυο είναι σημαντικός για οργανισμούς που έχουν πολλούς εργαζόμενους, απασχολούν εργολάβους ή επιτρέπουν την πρόσβαση σε τρίτους, όπως πελάτες και προμηθευτές, καθιστώντας δύσκολη την αποτελεσματική παρακολούθηση της πρόσβασης στο δίκτυο. Οι εταιρείες που εξαρτώνται από το RBAC είναι σε θέση να ασφαλίσουν τα ευαίσθητα δεδομένα και τις κρίσιμες εφαρμογές τους.[38]

Servlet Container

Είναι ένα στοιχείο του web server που διευκολύνει την φόρτωση, εκτέλεση και αρχικοποίηση των servlets. Παραδείγματα τέτοιων servlet containers είναι το Tomcat, το Glassfish κλπ.

Ένας web server χρησιμοποιεί το Hyper Text Transfer Protocol (HTTP) για την επικοινωνία με τελικούς χρήστες. Ένας server βασισμένος σε java επικοινωνεί μέσω μηνυμάτων HTTP, με χρήση δύο βασικών κλάσεων, της java.net.Socket και της java.net.ServerSocket. Το Servlet container διαχειρίζεται μεγάλο αριθμό από servlets, δηλαδή μπορεί να διαχειριστεί πολλά requests.

Τα Servlets μπορούν να παραμετροποιηθούν ώστε να επιτρέπουν την πρόσβαση σε εξουσιοδοτημένους χρήστες, ορίζοντας ρόλους και περιορισμούς.

Οι ρόλοι ασφαλείας ορίζουν την λειτουργία μιας εφαρμογής, η οποία αποτελείται από χρήστες και γκρουπ. Η σχέση μεταξύ χρηστών και groups βασίζεται στην υλοποίηση που χρησιμοποιείται. Οι ρόλοι μπορούν να οριστούν στο Java EE deployment descriptor αρχείο σαν web.xml και η αντίστοιχη χαρτογράφηση των ρόλων στο web server deployment descriptor αρχείο σαν serv-web.xml.

Όσον αφορά τον ορισμό των περιορισμών, οι άδειες πρόσβασης μπορούν να οριστούν κάνοντας χρήση του auth-constraint στοιχείου στο web.xml αρχείο. Όταν ο χρήστης αυθεντικοποιηθεί, το auth-constraint που έχει οριστεί στο deployment descriptor ελέγχει αν ο χρήστης ανήκει σε κάποιον από τους ρόλους.

Παρακάτω παρατίθεται παράδειγμα ορισμού ρόλων.

```
<web-app>
       <security-role-mapping>
              <role-name>clerk</role-name>
              <principal-name>john</principal-name>
              <principal-name>mwebester</principal-name>
              <group-name>HR</group-name>
       </security-role-mapping>
```

[37] "What is mandatory access control (MAC)? - SearchSecurity." https://searchsecurity.techtarget.com/definition/mandatory-access-control-MAC. Πρόσβαση στις 6 Οκτ. 2020.

[38] "What is role-based access control (RBAC)? - Definition from ...." https://searchsecurity.techtarget.com/definition/role-based-access-control-RBAC. Πρόσβαση στις 6 Οκτ. 2020.

```
        <security-role-mapping>
                <role-name>manager</role-name>
                <principal-name>dsmith</principal-name>
        </security-role-mapping>
</web-app>
```

3.2.3 Pluggable Authentication Module (PAM)

Ο κύριος στόχος του PAM είναι να επιτρέψει στους προγραμματιστές να χρησιμοποιήσουν μία ποιοτική διεπαφή αυθεντικοποίησης, αφήνοντας ταυτοχρονα την επιλογή της τεχνολογίας που θα χρησιμοποιηθεί στον system administrator.

Οι τεχνολογίες αυθεντικοποίησης εκτελούνται στα login modules, οι οποίες μπορούν να εγκατασταθούν μετά την υλοποίηση της εφαρμογής, μέσω του αρχείου login.config. Το login.config μπορεί όχι μόνο να αναγνωρίσει ποια modules να καλέσει, αλλά και όλες τις συνθήκες που χρειάζονται για μία ολοκληρωμένη και επιτυχής αυθεντικοποίηση.

Το PAM επιτρέπει στις νέες μεθόδους ελέγχου ταυτότητας ή τις τεχνολογίες να προστίθενται εύκολα στις υπάρχουσες εφαρμογές. Επίσης, μπορεί να αλλάξει μια πολιτική ελέγχου ταυτότητας απλά με ενημέρωση του αρχείου login.config, αντί να γραφτεί από την αρχή όλη η εφαρμογή.

Μέσω του Java Authentication & Authorization Service που θα περιγραφεί παρακάτω μπορεί να ενσωματωθεί το Pluggable Authentication Modules (PAM). Ακολουθούν μερικά PAM modules:

- ❏ com.sun.security.auth.module.NTLoginModule
- ❏ com.sun.security.auth.module.NTSystem
- ❏ com.sun.security.auth.module.JndiLoginModule
- ❏ com.sun.security.auth.module.KeyStoreLoginModule
- ❏ com.sun.security.auth.module.Krb5LoginModule
- ❏ com.sun.security.auth.module.SolarisSystem
- ❏ com.sun.security.auth.module.UnixLoginModule
- ❏ com.sun.security.auth.module.UnixSystem

3.2.4 Java Authentication & Authorization Service (JAAS)

Το Java Authentication and Authorization Service (JAAS) αναπτύχθηκε ως μη υποχρεωτικό πακέτο (επέκταση) στην τυπική έκδοση J2SDK. Υπάρχουν δύο σκοποί που μπορεί να χρησιμοποιηθεί το JAAS ως εξής:

- ❖ Για έλεγχο ταυτότητας χρηστών, να προσδιοριστεί με αξιοπιστία και ασφάλεια ποιος εκτελεί τον κώδικα Java αυτήν τη στιγμή, ανεξάρτητα από το εάν ο κώδικας εκτελείται ως εφαρμογή, applet, bean ή servlet.
- ❖ Για την εξουσιοδότηση των χρηστών, ώστε να διασφαλιστεί ότι έχουν τα δικαιώματα ελέγχου πρόσβασης που απαιτούνται για την εκτέλεση των εργασιών τους.

Ο έλεγχος ταυτότητας JAAS πραγματοποιείται με τη μορφή plugin. Αυτό επιτρέπει στις εφαρμογές Java να παραμένουν ανεξάρτητες από τις υποκείμενες τεχνολογίες ελέγχου ταυτότητας. Νέες ή ενημερωμένες τεχνολογίες μπορούν να συνδεθούν χωρίς να απαιτούνται τροποποιήσεις στην ίδια την εφαρμογή. Μια εφαρμογή για μια συγκεκριμένη τεχνολογία ελέγχου ταυτότητας που θα χρησιμοποιηθεί καθορίζεται κατά το χρόνο εκτέλεσης. Η υλοποίηση καθορίζεται σε ένα αρχείο διαμόρφωσης σύνδεσης.

3.2.4.1 Βασικές κλάσεις και Διεπαφές

Οι βασικές κλάσεις και διεπαφές που σχετίζονται με το JAAS μπορούν να χωριστούν σε τρεις κατηγορίες: Κοινές, Αυθεντικοποίησης και Εξουσιοδότησης.

3.2.4.1.1 Κοινές Κλάσεις

Οι κοινές κλάσεις είναι αυτές που μοιράζονται τόσο τα στοιχεία αυθεντικοποίησης όσο και τα στοιχεία εξουσιοδότησης JAAS. Η βασική κλάση JAAS είναι η javax.security.auth.Subject, η οποία αντιπροσωπεύει μια ομαδοποίηση σχετικών πληροφοριών για μια μεμονωμένη οντότητα, όπως ένα άτομο. Περιλαμβάνει τα Principals της οντότητας, τα δημόσια διαπιστευτήρια και τα ιδιωτικά διαπιστευτήρια. Επίσης ένα διαπιστευτήριο, όπως ορίζεται από το JAAS, μπορεί να είναι οποιοδήποτε object.

Subject
Για να εξουσιοδοτηθεί η πρόσβαση σε πόρους, οι εφαρμογές πρέπει πρώτα να πιστοποιήσουν την πηγή του αιτήματος. Το JAAS framework ορίζει τον όρο subject που αντιπροσωπεύει την πηγή ενός αιτήματος. Ένα subject μπορεί να είναι οποιαδήποτε οντότητα, όπως ένα άτομο ή μια υπηρεσία. Μόλις επικυρωθεί το subject, ένα javax.security.auth.Subject συμπληρώνεται με σχετικές ταυτότητες ή Principals. Ένα subject μπορεί να έχει πολλούς Principals. Για παράδειγμα, ένα άτομο μπορεί να έχει ένα όνομα Principal ("John Doe") και ένα SSN Principal ("123-45-6789"), το οποίο το διακρίνει από τα άλλα subjects.

Ένα subject μπορεί επίσης να διαθέτει χαρακτηριστικά που σχετίζονται με την ασφάλεια, τα οποία αναφέρονται ως διαπιστευτήρια. Ευαίσθητα διαπιστευτήρια που απαιτούν ειδική προστασία, όπως ιδιωτικά κρυπτογραφικά κλειδιά, αποθηκεύονται σε ένα ιδιωτικό set διαπιστευτηρίων. Τα διαπιστευτήρια που προορίζονται για κοινή χρήση, όπως πιστοποιητικά δημοσίου κλειδιού, αποθηκεύονται σε ένα δημόσιο set διαπιστευτηρίων. Απαιτούνται διαφορετικά δικαιώματα για πρόσβαση και τροποποίηση των διαφορετικών set διαπιστευτηρίων.

Τα subjects δημιουργούνται χρησιμοποιώντας αυτούς τους κατασκευαστές:

```
public Subject();
public Subject(boolean readOnly, Set principals,
                   Set pubCredentials, Set privCredentials)
```

Ο πρώτος κατασκευαστής δημιουργεί ένα subject με κενά (μη μηδενικά) sets Principals και διαπιστευτηρίων. Ο δεύτερος κατασκευαστής δημιουργεί ένα subject με τα καθορισμένα sets Principals και διαπιστευτηρίων. Έχει επίσης ένα δυαδικό όρισμα που μπορεί να χρησιμοποιηθεί για να κάνει το subject read-only. Σε ένα read-only subject, τα sets των Principals και διαπιστευτηρίων είναι αμετάβλητα.

Παρακάτω φαίνονται κάποιες από τις μεθόδους των subjects:

- Ένα σετ από Principal objects επιστρέφονται από την subject.getPrincipals(). Για την διαχείριση του set χρησιμοποιούνται οι μέθοδοι add(), remove() και contains().
- Ένα set από public credentials επιστρέφονται από την κλήση της subject.getPublicCredentials().
- Το ίδιο και για το set των private credentials που επιστρέφονται μέσω της subject.getPrivateCredentials().

Principals
Όπως αναφέρθηκε προηγουμένως, όταν ένα subject επικυρωθεί, συμπληρώνεται με συσχετισμένες ταυτότητες ή Principals. Ένα subject μπορεί να έχει πολλούς Principals. Για παράδειγμα, ένα άτομο μπορεί να έχει ένα όνομα Principal ("John Doe") και ένα SSN Principal ("123-45-6789"), το οποίο το διακρίνει από άλλα subjects. Ένας Principal πρέπει να ενσωματώσει τις διεπαφές java.security.Principal και java.io.Serializable.

Credentials
Εκτός από τους συνδεδεμένους Principals, ένα subject μπορεί να διαθέτει χαρακτηριστικά που σχετίζονται με την ασφάλεια, τα οποία αναφέρονται ως διαπιστευτήρια. Ένα διαπιστευτήριο μπορεί να περιέχει πληροφορίες που χρησιμοποιούνται για τον έλεγχο ταυτότητας

του subject σε νέες υπηρεσίες. Τέτοια διαπιστευτήρια περιλαμβάνουν κωδικούς πρόσβασης, Kerberos tickets και πιστοποιητικά δημοσίου κλειδιού. Τα διαπιστευτήρια ενδέχεται επίσης να περιέχουν δεδομένα που απλώς επιτρέπουν στο άτομο να εκτελεί συγκεκριμένες δραστηριότητες. Τα κρυπτογραφικά κλειδιά, για παράδειγμα, αντιπροσωπεύουν διαπιστευτήρια που επιτρέπουν στο υποκείμενο να υπογράφει ή να κρυπτογραφεί δεδομένα. Οι δημόσιες και ιδιωτικές κλάσεις διαπιστευτηρίων δεν αποτελούν μέρος της βασικής βιβλιοθήκης κλάσεων του JAAS. Ωστόσο, οι προγραμματιστές μπορούν να επιλέξουν να εφαρμόσουν τις κλάσεις διαπιστευτηρίων τους σε δύο διεπαφές που σχετίζονται με διαπιστευτήρια: Refreshable και Destroyable.

Refreshable

Η διεπαφή javax.security.auth.Refreshable παρέχει τη δυνατότητα ανανέωσης του διαπιστευτηρίου. Για παράδειγμα, ένα διαπιστευτήριο με μια συγκεκριμένη χρονικά περιορισμένη διάρκεια ζωής μπορεί να εφαρμόσει αυτήν τη διεπαφή για να επιτρέψει στους καλούντες να ανανεώσουν τη χρονική περίοδο για την οποία είναι έγκυρο το διαπιστευτήριο.

Destroyable

Η διεπαφή javax.security.auth.Destroyable παρέχει τη δυνατότητα καταστροφής των περιεχομένων των διαπιστευτηρίων.

3.2.4.1.2 Κλάσεις και Διεπαφές Αυθεντικοποίησης

Η αυθεντικοποίηση αντιπροσωπεύει τη διαδικασία με την οποία επαληθεύεται η ταυτότητα ενός subject και πρέπει να εκτελείται με ασφαλή τρόπο. Διαφορετικά, ένας επιτιθέμενος μπορεί να υποδυθεί άλλους για να αποκτήσει πρόσβαση σε ένα σύστημα. Ο έλεγχος ταυτότητας συνήθως περιλαμβάνει το άτομο που δείχνει κάποια μορφή αποδεικτικών στοιχείων για να αποδείξει την ταυτότητά του. Τέτοια αποδεικτικά στοιχεία μπορεί να είναι πληροφορίες που μόνο το άτομο θα μπορούσε να γνωρίζει ή να έχει (όπως κωδικός πρόσβασης ή δακτυλικό αποτύπωμα), ή μπορεί να είναι πληροφορίες που μόνο θα μπορούσε να παράγει το υποκείμενο (όπως υπογεγραμμένα δεδομένα χρησιμοποιώντας ένα ιδιωτικό κλειδί).

Για τον έλεγχο ταυτότητας ενός subject (χρήστης ή υπηρεσία), εκτελούνται τα ακόλουθα βήματα:

1. Μια εφαρμογή δημιουργεί ένα LoginContext.

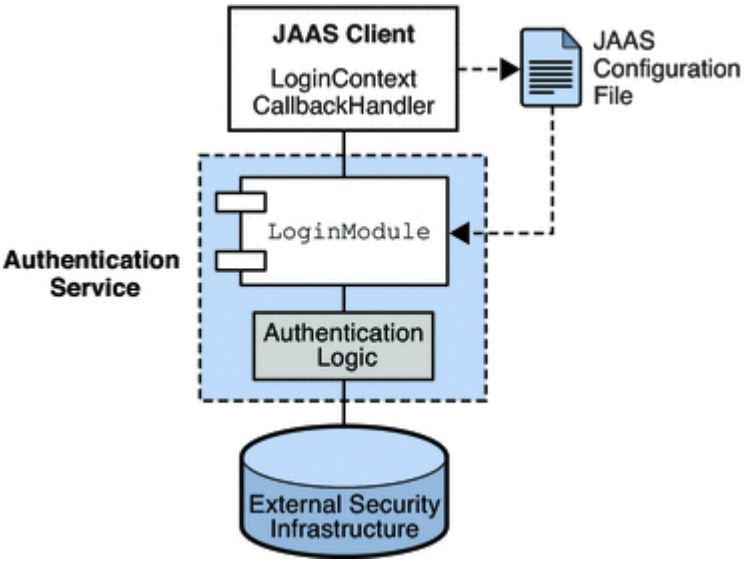
2. Το LoginContext συμβουλεύεται ένα Configuration για τη φόρτωση όλων των LoginModules που έχουν ρυθμιστεί για αυτήν την εφαρμογή.

3. Η εφαρμογή επικαλείται τη μέθοδο login της LoginContext.

4. Η μέθοδος login καλεί όλα τα φορτωμένα LoginModules. Κάθε LoginModule προσπαθεί να πιστοποιήσει το subject. Μετά την επιτυχία της διαδικασίας, το LoginModules συσχετίζει τους σχετικούς Principals και τα διαπιστευτήρια με ένα subject object που αντιπροσωπεύει το subject που επικυρώνεται.

5. Το LoginContext επιστρέφει την κατάσταση ελέγχου ταυτότητας στην εφαρμογή.

6. Εάν ο έλεγχος ταυτότητας είναι επιτυχής, η εφαρμογή ανακτά το subject από το LoginContext.



LoginContext

Η κλάση javax.security.auth.login.LoginContext παρέχει τις βασικές μεθόδους που χρησιμοποιούνται για τον έλεγχο ταυτότητας subjects και παρέχει έναν τρόπο ανάπτυξης μιας εφαρμογής ανεξάρτητης από την υποκείμενη τεχνολογία ελέγχου ταυτότητας. Το LoginContext συμβουλεύεται ένα Configuration για να προσδιορίσει τις υπηρεσίες ελέγχου ταυτότητας ή LoginModule(s), που έχουν διαμορφωθεί για μια συγκεκριμένη εφαρμογή. Επομένως, διαφορετικά LoginModules μπορούν να συνδεθούν σε μια εφαρμογή χωρίς να απαιτούνται τροποποιήσεις στην ίδια την εφαρμογή.

Το LoginContext προσφέρει τέσσερις κατασκευαστές:

```
public LoginContext(String name) throws LoginException;

public LoginContext(String name, Subject subject) throws LoginException;

public LoginContext(String name, CallbackHandler callbackHandler)
          throws LoginException

public LoginContext(String name, Subject subject,
          CallbackHandler callbackHandler) throws LoginException
```

Όλοι οι κατασκευαστές μοιράζονται μια κοινή παράμετρο: name. Αυτό το όρισμα χρησιμοποιείται από το LoginContext ως ευρετήριο στο login configuration για να προσδιορίσει ποια στοιχεία σύνδεσης έχουν ρυθμιστεί για την εφαρμογή που δημιουργεί το LoginContext. Οι κατασκευαστές που δεν λαμβάνουν ένα subject ως παράμετρο εισαγωγής δημιουργούν ένα νέο subject. Οι null είσοδοι δεν επιτρέπονται για όλους τους κατασκευαστές.
Ο πραγματικός έλεγχος ταυτότητας πραγματοποιείται με μια κλήση στην ακόλουθη μέθοδο:

```
   public void login() throws LoginException;
```

Όταν γίνεται login, όλα τα διαμορφωμένα LoginModules καλούνται να πραγματοποιήσουν τον έλεγχο ταυτότητας. Εάν ο έλεγχος ταυτότητας πετύχει, το subject μπορεί να ανακτηθεί χρησιμοποιώντας την ακόλουθη μέθοδο:

```
    public Subject getSubject();
```

Για αποσύνδεση ενός subject και κατάργηση των επικυρωμένων Principals και διαπιστευτηρίων του, παρέχεται η ακόλουθη μέθοδος:

```
   public void logout() throws LoginException;
```

Το παρακάτω δείγμα κώδικα δείχνει τις κλήσεις που είναι απαραίτητες για τον έλεγχο ταυτότητας και αποσύνδεση ενός subject:

```
// let the LoginContext instantiate a new Subject
    LoginContext lc = new LoginContext("entryFoo");
    try {
        // authenticate the Subject
        lc.login();
        System.out.println("authentication successful");

        // get the authenticated Subject
        Subject subject = lc.getSubject();

        ...

        // all finished -- logout
        lc.logout();
    } catch (LoginException le) {
        System.err.println("authentication unsuccessful: " +
            le.getMessage());
    }
```

LoginModule

Η διεπαφή LoginModule δίνει στους προγραμματιστές τη δυνατότητα να εφαρμόσουν διαφορετικά είδη τεχνολογιών ελέγχου ταυτότητας που μπορούν να συνδεθούν σε μια εφαρμογή. Για παράδειγμα, ένας τύπος LoginModule ενδέχεται να εκτελέσει μια μορφή ελέγχου ταυτότητας βάσει ονόματος χρήστη / κωδικού πρόσβασης. Άλλα στοιχεία σύνδεσης μπορούν να συνδεθούν σε συσκευές υλικού όπως έξυπνες κάρτες ή βιομετρικές συσκευές.

CallbackHandler

Σε ορισμένες περιπτώσεις, ένα LoginModule πρέπει να επικοινωνήσει με τον χρήστη για να λάβει πληροφορίες ελέγχου ταυτότητας. Τα LoginModules χρησιμοποιούν ένα javax.security.auth.callback.CallbackHandler για το σκοπό αυτό. Οι εφαρμογές εφαρμόζουν τη διεπαφή CallbackHandler και να μεταβιβάζουν στο LoginContext, το οποίο προωθεί απευθείας στα υποκείμενα LoginModules. Το LoginModule χρησιμοποιεί το CallbackHandler για να συλλέξει πληροφορίες από χρήστες (όπως έναν κωδικό πρόσβασης ή έναν αριθμό pin έξυπνης κάρτας) ή για να παρέχει πληροφορίες στους χρήστες (όπως πληροφορίες κατάστασης). Επιτρέποντας στην εφαρμογή να καθορίσει το CallbackHandler, τα υποκείμενα LoginModules μπορούν να παραμείνουν ανεξάρτητα από τους διαφορετικούς τρόπους αλληλεπίδρασης των εφαρμογών με τους χρήστες. Για παράδειγμα, η εφαρμογή ενός CallbackHandler για μια εφαρμογή GUI ενδέχεται να εμφανίσει ένα παράθυρο για να ζητήσει είσοδο από έναν χρήστη. Από την άλλη πλευρά, η εφαρμογή ενός CallbackHandler για ένα εργαλείο χωρίς GUI μπορεί απλώς να ζητήσει από το χρήστη να βάλει input απευθείας από τη γραμμή εντολών.

Ο CallbackHandler είναι μια διεπαφή με μία μέθοδο:

```
void handle(Callback[] callbacks)
        throws java.io.IOException, UnsupportedCallbackException;
```

Το LoginModule μεταβιβάζει στη μέθοδο χειρισμού CallbackHandler μια σειρά από Callbacks, για παράδειγμα ένα NameCallback για το όνομα χρήστη και ένα PasswordCallback για τον κωδικό πρόσβασης και το CallbackHandler εκτελεί την ζητούμενη αλληλεπίδραση του χρήστη και ορίζει τις κατάλληλες τιμές στα Callbacks. Για παράδειγμα, για να επεξεργαστεί ένα NameCallback, το CallbackHandler μπορεί να ζητήσει ένα όνομα, να ανακτήσει την τιμή από τον χρήστη και να καλέσει τη μέθοδο setName του NameCallback για να αποθηκεύσει το όνομα.

Callback
Το πακέτο javax.security.auth.callback περιέχει τη διασύνδεση Callback καθώς και πολλές εφαρμογές. Τα LoginModules μπορούν να μεταφέρουν μια σειρά από Callbacks απευθείας στη μέθοδο χειρισμού ενός CallbackHandler.

3.2.4.1.3 Κλάσεις Εξουσιοδότησης

Για να γίνει η εξουσιοδότηση JAAS, παρέχοντας δικαιώματα ελέγχου πρόσβασης που βασίζονται όχι μόνο σε ποιον κώδικα εκτελείται, αλλά και σε ποιον τον εκτελεί, απαιτούνται τα εξής:

- Ο χρήστης πρέπει να πιστοποιηθεί, όπως περιγράφεται στην ενότητα LoginContext.
- Το subject που είναι το αποτέλεσμα ελέγχου ταυτότητας πρέπει να συσχετιστεί με ένα πλαίσιο ελέγχου πρόσβασης, όπως περιγράφεται στην ενότητα subject.
- Οι Principal καταχωρήσεις πρέπει να διαμορφωθούν στην πολιτική ασφαλείας, όπως περιγράφεται παρακάτω.

Η Policy κλάση και οι κλάσεις για εξουσιοδότηση AuthPermission και PrivateCredentialPermission περιγράφονται παρακάτω.

Policy
Η κλάση java.security.Policy είναι μια αφηρημένη κλάση που αντιπροσωπεύει την πολιτική ελέγχου πρόσβασης σε όλο το σύστημα. Το Policy API υποστηρίζει policy-based queries.

Ως προεπιλογή, το JDK παρέχει μια file-based ενσωμάτωση υποκατηγορίας , η οποία αναβαθμίστηκε για να υποστηρίζει Principal-based καταχωρήσεις σε policy αρχεία.

AuthPermission
Η κλάση javax.security.auth.AuthPermission ενσωματώνει τα βασικά δικαιώματα που απαιτούνται για το JAAS. Ένα AuthPermission περιέχει ένα όνομα (αναφέρεται επίσης ως "target name") αλλά δεν περιέχει λίστα ενεργειών, είτε υπάρχει η ονομαστική άδεια είτε όχι.

Εκτός από τις κληρονομικές μεθόδους (από την κλάση java.security.Permission), ένα AuthPermission έχει δύο δημόσιους κατασκευαστές:

```
public AuthPermission(String name);
public AuthPermission(String name, String actions);
```

Ο πρώτος κατασκευαστής δημιουργεί ένα νέο AuthPermission με το καθορισμένο όνομα. Ο δεύτερος κατασκευαστής δημιουργεί επίσης ένα νέο αντικείμενο AuthPermission με το καθορισμένο όνομα, αλλά έχει ένα πρόσθετο όρισμα ενεργειών που προς το παρόν δεν χρησιμοποιείται και πρέπει να είναι null. Αυτός ο κατασκευαστής υπάρχει αποκλειστικά για το Policy Object, ώστε να δημιουργεί νέα αντικείμενα δικαιωμάτων. Για τις υπόλοιπες περιπτώσεις, ο πρώτος κατασκευαστής είναι κατάλληλος.

Προς το παρόν, το αντικείμενο AuthPermission χρησιμοποιείται για την προστασία της πρόσβασης στα Policy, στα Subjects, στα LoginContext και Configuration objects.

PrivateCredentialPermission
Η κλάση javax.security.auth.PrivateCredentialPermission προστατεύει την πρόσβαση σε ιδιωτικά διαπιστευτήρια ενός subject και παρέχει έναν δημόσιο κατασκευαστή:

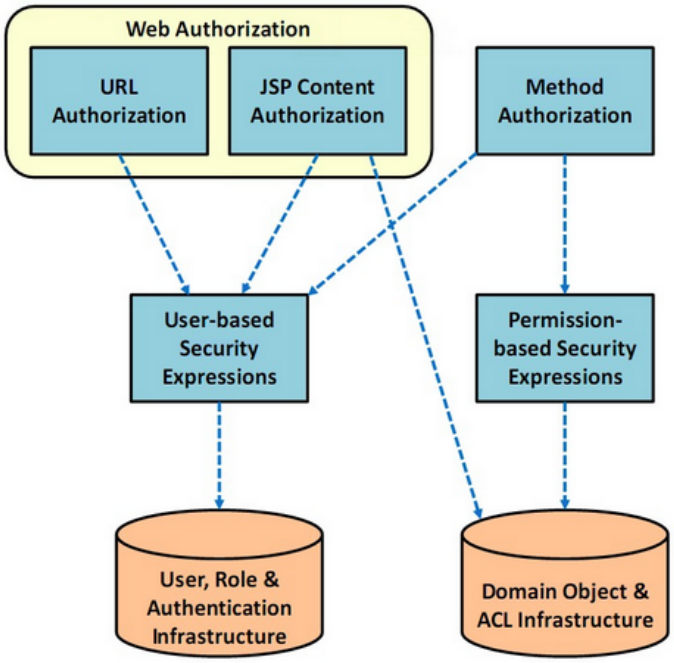
```
    public PrivateCredentialPermission(String name, String actions);
```

[39]

[39] "Java Authentication and Authorization Service (JAAS ...." https://docs.oracle.com/javase/10/security/java-authentication-and-authorization-service-jaas-reference-guide.htm. Πρόσβαση στις 7 Οκτ. 2020.

3.2.5 Spring Security Framework

Το Spring Security είναι ένα ισχυρό και εξαιρετικά προσαρμόσιμο framework ελέγχου ταυτότητας και πρόσβασης. Είναι επίσης ένα framework που επικεντρώνεται στην παροχή ταυτότητας και εξουσιοδότησης σε εφαρμογές Java. Όπως όλα τα projects της Spring, η πραγματική δύναμη της Spring Security βρίσκεται στο πόσο εύκολα μπορεί να επεκταθεί για να ικανοποιήσει τις προσαρμοσμένες απαιτήσεις.[40] Χρησιμοποιεί το javax.servlet.Filter για την εφαρμογή ελέγχου ταυτότητας και εξουσιοδότησης. Επίσης το Servlet API και το Spring Web MVC μπορούν να εισάγουν και να χρησιμοποιήσουν το Spring Security Framework.



Παρακάτω παρατίθενται τα βασικότερα χαρακτηριστικά του Spring Security Framework:

- ❏ Ενσωμάτωση αυθεντικοποίησης και εξουσιοδότηση χρήστη.
- ❏ Προσφέρει login και logout χαρακτηριστικά.
- ❏ Role-Based Έλεγχος Εξουσιοδότησης
- ❏ Προσφέρει σύνδεσμο για αυθεντικοποίηση και εξουσιοδότηση database-based.
- ❏ Υποστηρίζει κρυπτογραφημένους κωδικούς.
- ❏ Υποστηρίζει form-based αυθεντικοποίηση
- ❏ Προσφέρει page-based αυθεντικοποίηση και εξουσιοδότηση χρήστη.

Ακόμα το Spring Security Framework περιέχει πολλά sub-modules, τα οποία φαίνονται παρακάτω.

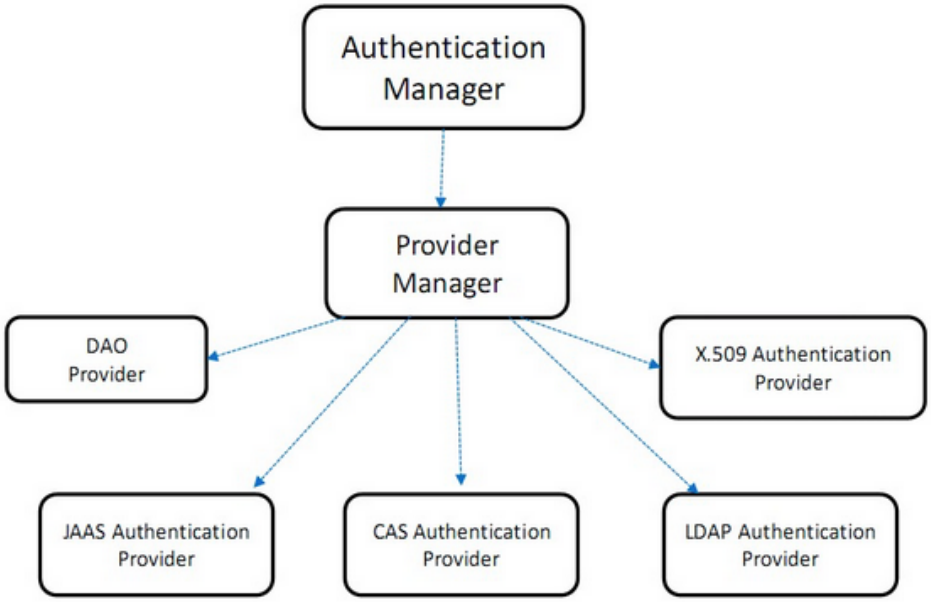
Module	Περιγραφή
Core (spring-security-core.jar)	Περιλαμβάνει authentication και access-control APIs.
Web (spring-security-web.jar)	Περιλαμβάνει Servlet filters και web-based authentication APIs. Απαιτείται για εφαρμογές web με web authentication και URL authorization.
Remoting (spring-security-remoting.jar)	Περιλαμβάνει το Spring Remoting API, που είναι απαραίτητο για

[40] "Spring Security." https://spring.io/projects/spring-security. Πρόσβαση στις 8 Οκτ. 2020.

	απομακρυσμένες client εφαρμογές.
Config (spring-security-config.jar)	Περιλαμβάνει API για να ενεργοποιήσει την χρήση των Spring Security XML namespaces σαν XML configuration
LDAP (spring-security-ldap.jar)	Περιλαμβάνει API για την υποστήριξη LDAP.
ACL (spring-security-acl.jar)	Περιλαμβάνει API για την ενσωμάτωση domain object security με χρήση access control lists (ACLs)
CAS (spring-security-cas.jar)	Περιλαμβάνει API για την ενσωμάτωση Spring Security web authentication με έναν CAS single-on server.

3.2.5.1 Spring Authentication

Το Spring Security παρέχει ολοκληρωμένη υποστήριξη για έλεγχο ταυτότητας. Ο έλεγχος ταυτότητας είναι πώς επαληθεύεται η ταυτότητα του χρήστη που προσπαθεί να αποκτήσει πρόσβαση σε έναν συγκεκριμένο πόρο. Ένας συνηθισμένος τρόπος ελέγχου ταυτότητας χρηστών είναι η απαίτηση του χρήστη να εισάγει ένα όνομα χρήστη και έναν κωδικό πρόσβασης. Μόλις πραγματοποιηθεί έλεγχος ταυτότητας, πραγματοποιείται και η εξουσιοδότηση ανάλογα με την ταυτότητα του χρήστη.
Η διαδικασία αυθεντικοποίησης φαίνεται στο παρακάτω διάγραμμα.



Το Spring Security υποστηρίζει ένα πλήθος από pluggable Authentication Providers, όπως για παράδειγμα το LDAP, τα X.509(πιστοποιητικά), Database(JDBC), JAAS και OAuth. Ο **AuthenticationManager**, που είναι μια διεπαφή που επεξεργάζεται όλα τα authentication requests, είναι ο πυρήνας επεξεργασίας.[41] Στη συνέχεια ο **ProviderManager** που είναι μία ενσωμάτωση authentication manager που έχει την ευθύνη για την αυθεντικοποίηση ενός ή περισσότερων authentication providers. Επίσης ο **AuthenticationProvider** ενσωματώνει το **UserDetailService** που είναι υπεύθυνο να πάρει τις πληροφορίες του χρήστη. Οι πληροφορίες που λαμβάνονται αντιστοιχίζονται με το δοσμένο username/ password κατα το login. Οι πληροφορίες αποθηκεύονται στο ApplicationContext ή στην βάση δεδομένων ή σε ένα Custom user Table. Παρόλο που οι πληροφορίες του χρήστη αποθηκεύονται στο Application Context, αποθηκεύονται ως plain-text. Έτσι είναι επιτακτική η ανάγκη για κρυπτογράφηση των passwords. Μπορεί να γίνει κρυπτογράφηση του κωδικού με το SHA1 <password-encoder> στοιχείο.

[41] "Spring Authentication - Cheng." 31 Ιουλ. 2019, https://chengyu.home.blog/2019/07/31/spring-authentication/. Πρόσβαση στις 9 Οκτ. 2020.

Ευάλωτο ApplicationContext.xml

ApplicationContext.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <sec:authentication-manager id="authManager">
4 <sec:authentication-provider>
5 <sec:user-service>
6 <sec:user name="admin" password="password" authorities="ROLE_ADMIN,ROLE_USER"/>
7 <sec:user name="user" password="password" authorities="ROLE_USER"/>
8 </sec:user-service></sec:authentication-provider>
9 </sec:authentication-manager>
```

Ασφαλές ApplicationContext.xml

ApplicationContext.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <sec:authentication-manager id="authManager">
4 <sec:authentication-provider>
5 <password-encoder hash="sha" />
6 <sec:user-service>
7 <sec:user name="admin" password="password" authorities="ROLE_ADMIN,ROLE_USER"/>
8 <sec:user name="user" password="password" authorities="ROLE_USER"/>
9 </sec:user-service></sec:authentication-provider>
10 </sec:authentication-manager>
```

[42]

3.2.5.2 Security Expressions

Το Spring Security περιλαμβάνει εκφράσεις ασφαλείας Spring Expression Language (SpEL) για εξουσιοδότηση. Οι εκφράσεις αξιολογούνται με βάση ένα context-dependent “root object” που ονομάζεται **SecurityExpressionRoot**. Στο web context, το root object ονομάζεται **WebSecurityExpressionRoot**, που προέρχεται από το SecurityExpressionRoot.

Όροι έκφρασης ασφάλειας χρήστη και προδιαγραφές

Όροι:

- Authentication: το αντικείμενο αυθεντικοποίησης του τρέχοντος χρήστη, το οποίο έχει ληφθεί από το SecurityContext.
- Principal: το principal αντικείμενο του τρέχοντος χρήστη, που προέρχεται από το αντικείμενο Authentication.

Προδιαγραφές:

- HasRole ([role]): Ο χρήστης έχει τον καθορισμένο ρόλο και, στη συνέχεια, επιστρέφει το "true"
- permitAll: Πάντα true
- denyAll: Πάντα false
- isAnonymous (): Επιστρέφει true αν ο χρήστης είναι ανώνυμος
- isRememberMe (): Επιστρέφει ο true αν ο χρήστης έχει αυθεντικοποιηθεί χρησιμοποιώντας την επιλογή Remember-Me
- isAuthenticated (): επιστροφή true εάν ο χρήστης δεν είναι ανώνυμος χρήστης
- isFullyAuthenticated (): Επιστρέφει true αν ο χρήστης δεν είναι ανώνυμος χρήστης και δεν επικυρώθηκε χρησιμοποιώντας την επιλογή Remember-Me

[42] "下一步Storing Username and Password - Cheng." 31 Ιουλ. 2019, https://chengyu.home.blog/2019/07/31/storing-username-and-password/. Πρόσβαση στις 9 Οκτ. 2020.

Όροι WebSecurityExpressionRoot

- ❏ Request: δηλώνει το HttpServletRequest
- ❏ hasIpAddress (ipAddr): επιστρέφει true αν η διεύθυνση IP του client είναι ίδια με την καθορισμένη διεύθυνση IP. Το IpAddr μπορεί να είναι μία μεμονωμένη διεύθυνση IP ή αλλιώς μια σειρά διευθύνσεων IP χρησιμοποιώντας IP / netmask notation.

3.2.5.3 Εξουσιοδότηση Web

Η εξουσιοδότηση ασχολείται με τον έλεγχο της πρόσβασης σε ασφαλείς πόρους. Η εξουσιοδότηση με εκφράσεις ιστού επιτρέπουν τη δημιουργία κανόνων πρόσβασης από την άποψη των χαρακτηριστικών του χρήστη, όπως η κατάσταση ελέγχου ταυτότητας, οι ρόλοι των χρηστών και η διεύθυνση IP του χρήστη.
Το Spring security υποστηρίζει δύο επιλογές εξουσιοδότησης ιστού. Η πρώτη χρησιμοποιεί τις εκφράσεις ασφαλείας ιστού που περιγράφηκαν για τον έλεγχο της πρόσβασης σε διευθύνσεις URL, προαιρετικά ζεύγη μεθόδων URL / HTTP και η δεύτερη περιλαμβάνει την εμφάνιση ή απόκρυψη περιεχομένου JSP χρησιμοποιώντας το Spring Security

URL-Based Authorization

Για την ενσωμάτωση κανόνων πρόσβασης URL-Based, πρέπει να γίνει χρήση των tags <intercept-url> κάτω από τα tags <http>. Το tag <intercept-url> περιλαμβάνει attributes που φαίνονται παρακάτω:

- ❏ Pattern: προσδιορίζει το url pattern. Χρησιμοποιεί την σύνταξη Ant από προεπιλογή (π.χ * και ** wildcards), αλλά υποστηρίζονται και κανονικές εκφράσεις επίσης.
- ❏ Access: περιλαμβάνει τη λίστα των ρόλων των χρηστών που μπορούν να έχουν πρόσβαση στο url.
- ❏ Method: Προαιρετική Παράμετρος που ορίζει HTTP μέθοδο για εξουσιοδότηση.
- ❏ Filters: πιθανή τιμή της παραμέτρου είναι το “none”, που υποδηλώνει ότι το request θα κάνει bypass το Spring Security φίλτρο. Το request δεν θα έχει SecurityContext. Αυτό είναι κυρίως για στατικούς πόρους όπως Javascript, εικονες, CSS κλπ.
- ❏ Requires-channel: μπορεί να είναι είτε “http” ειτε “https”.

Τα προσδιορισμένα URLs που θα γίνουν intercept στέλνονται σαν metadata στο FilterSecurityInterceptor. Πρέπει να είναι βέβαιο ότι το ορισμένο url-pattern στο <intercept-url> τελειώνει με “*”, διαφορετικά ένας επιτιθέμενος θα μπορεί να περάσει παραμέτρους στο URL για να κάνει bypass τον κανόνα εξουσιοδότησης.

Παραδείγματα:

Για την απόκλιση εικόνων από το interception:

```
<intercept-url pattern="/images/**" filters="none" />
```

Πρόσβαση σε όλους τους χρήστες στην αρχική σελίδα:

```
<intercept-url pattern="/home" method=”GET” access=”permitAll” />
```

Μη αυθεντικοποιημένοι χρήστες μπορούν να δημιουργήσουν νέο λογαριασμό [RESTful URI/method]:

```
<intercept-url pattern="/users" method=”POST” access=”isAnonymous()” />
```

JSP Page Content Authorization

Η 2η μέθοδος είναι με την εξουσιοδότηση JSP Page Content, στην οποία το περιεχόμενο ενός προστατευμένου πόρου εμφανίζεται στον χρήστη ανάλογα με το user status του, τον ρόλο του κλπ. Για να εισαχθεί αυτή η μέθοδος εξουσιοδότησης πρέπει να εισαχθεί το παρακάτω:

Η βιβλιοθήκη των tags έχει τρία tags:

- ❏ <security: authentication>: Έκθεση του τρέχοντος Authentication αντικειμένου στο JSP
- ❏ <security: authorize>: εμφάνιση και απόκρυψη του περιεχομένου της ετικέτας με βάση τον τρέχοντα χρήστη και εαν ταιριάζει με την καθορισμένη συνθήκη
- ❏ <security: accesscontrollist>: εμφανίζει ή αποκρύπτει τις ετικέτες το περιεχόμενο με βάση την τρέχουσα άδεια χρήστη ταιριάζει με το καθορισμένο domain object.

Το κάθε tag έχει το δικό του Attribute:

- ❖ access: Δείχνει το περιεχόμενο του tag όταν η καθορισμένη έκφραση πρόσβασης Web Security είναι true.
- ❖ url: Δείχνει το περιεχόμενο του tag όταν ο χρήστης έχει άδεια πρόσβασης στο καθορισμένο url.
- ❖ Method: Περιορίζει την πρόσβαση σε συγκεκριμένες HTTP Μεθόδους (GET, POST, PUT, DELETE etc)

Παραδείγματα:

Αν ο χρήστης δεν είναι αυθεντικοποιημένος εμφανίζει ένα login link:

```
<security:authorize access="isAnonymous()">
	<a href="${loginUrl}">Log in</a>
</security:authorize>
```

Αν ο χρήστης έχει τον ρόλο του “instructor” θα εμφανίζεται ένα link “newcourse”:

```
<security:authorize access="hasRole('instructor')">
	<a href="${CreateUser}">New Course</a>
</security:authorize>
```

Αν ο χρήστης είναι αυθεντικοποιημένος εμφανίζει ένα logout link:

```
<security:authorize access="isAnonymous()">
	<a href="${logoutUrl}">Log out</a>
</security:authorize>
```

3.2.5.4 Method Εξουσιοδότηση

Εκτός από την εξουσιοδότηση web URL requests και JSP Content, το Spring Security χρησιμοποιεί και security expression annotations για να προστατέψει μεθόδους ή κλάσεις. Εισάγει expressions-based @Pre/@Post, τα οποία υποστηρίζουν και permission-based security expressions, και @secured annotations.

- ❏ @PreAuthorize: διαβεβαιώνει ότι η μέθοδος καλείται μόνο όταν η καθορισμένη έκφραση είναι true.
- ❏ @PostAuthorize: διαβεβαιώνει ότι η μέθοδος επιστρέφει τιμή μόνο όταν η καθορισμένη έκφραση είναι true.
- ❏ @Secured: Ορίζει μία λίστα στοιχείων ασφάλειας στο method level. Οι προυποθεσεις ασφάλειας, όπως Ρόλοι/ Άδειες μπορουν να καθοριστούν με την μέθοδο @security. Μόνο ο χρήστης με αυτούς τους ρόλους και άδειες μπορούν να καλέσουν αυτή τη μέθοδο.

Παραδείγματα:

Ο χρήστης με write ή admin άδεια μπορεί να επεξεργαστεί ένα μήνυμα

```
@PreAuthorize("hasPermission(#message, write) or hasPermission(#message, admin)")
public void editMessage(Message message) {...}
```

Μονο χρηστες με read permission μπορουν να δουν το forum

```
@PostAuthorize("hasPermission(new myapp.model.Forum(#id), read)")
public Forum getForum(long id) {...}
```

Χρήστες με admin ρόλο ή άδεια μπορούν να διαβάσουν blocked μηνύματα

```
@PostAuthorize("hasRole('admin') or hasPermission(filterObject, admin)")
public Message getMessage(long id) {...}
```

[43]

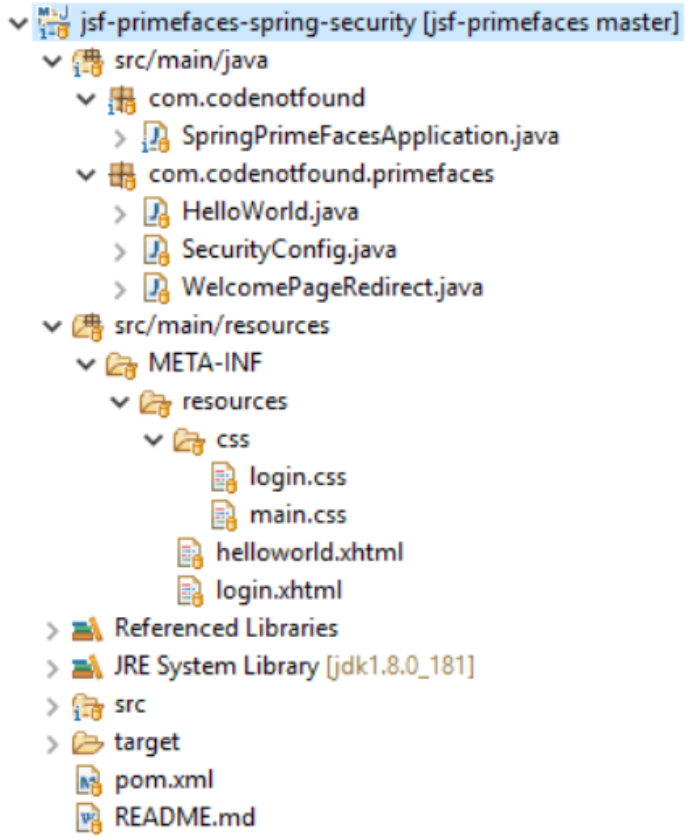
3.2.6 Spring Security + JSF

Σε αυτό το κεφάλαιο θα περιγραφεί ένας οδηγός εγκατάστασης login με χρήση του Spring Security Framework και του JSF PrimeFaces. Για την επίτευξη της εγκατάστασης θα χρησιμοποιηθούν τα παρακάτω εργαλεία/ frameworks:

- ❏ PrimeFaces 6.2
- ❏ JoinFaces 3.3
- ❏ Spring Boot 2.1
- ❏ Spring SEcurity 5.1
- ❏ Maven 3.5

[43] "Configuring Anonymous Login - Cheng." 31 Ιουλ. 2019, https://chengyu.home.blog/2019/07/31/configuring-anonymous-login/. Πρόσβαση στις 9 Οκτ. 2020.

Η δομή καταλόγου του project είναι η εξής:



3.2.6.1 Maven Setup

Το παράδειγμα βασίζεται στο Hello World Primefaces Tutorial στο οποίο εμφανίζεται ένα χαιρετηστήριο μήνυμα με βάση το όνομα που βάζει ο χρήστης στη φόρμα. Επίσης χρησιμοποιείται το welcome page από το PrimeFaces redirect example.

Για να χρησιμοποιηθεί το Spring Security πρέπει να προστεθεί το `spring-boot-starter-security` στο Maven POM αρχείο Αυτό θα συμπεριλάβει τις κύριες security dependencies που χρειάζονται για την ασφάλεια της JSF εφαρμογής.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<groupId>com.codenotfound</groupId>
	<artifactId>jsf-primefaces-spring-security</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<packaging>jar</packaging>
	<name>jsf-primefaces-spring-security</name>
	<description>JSF PrimeFaces Spring Security Example</description>
	<url>https://codenotfound.com/jsf-primefaces-spring-security-example.html</url>
	<parent>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-parent</artifactId>
	<version>2.1.0.RELEASE</version>
	<relativePath />
	<!-- lookup parent from repository -->
	</parent>
```

```
        <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <java.version>1.8</java.version>
        <joinfaces.version>3.3.0-rc2</joinfaces.version>
        </properties>
        <dependencyManagement>
        <dependencies>
                <dependency>
                <groupId>org.joinfaces</groupId>
                <artifactId>joinfaces-dependencies</artifactId>
                <version>${joinfaces.version}</version>
                <type>pom</type>
                <scope>import</scope>
                </dependency>
        </dependencies>
        </dependencyManagement>
        <dependencies>
        <dependency>
                <groupId>org.joinfaces</groupId>
                <artifactId>primefaces-spring-boot-starter</artifactId>
        </dependency>
        <dependency>
                <groupId>javax.enterprise</groupId>
                <artifactId>cdi-api</artifactId>
        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
        </dependencies>
        <build>
        <plugins>
                <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                </plugin>
        </plugins>
        </build>
</project>
```

Το Spring Security έχει default login generator, αλλά σε αυτό το παράδειγμα θα γίνει custom login page με χρήση των PrimeFaces στοιχείων. Η login.xhtml σελίδα βρίσκεται στο `src/main/resources/META-INF/resources` και αποτελείται από ένα `<p: inputText>` για το username και ένα `<p: password>` για τον κωδικό. Επίσης υπάρχει το `<p:commandButton>`για την καταχώρηση της φόρμας. Κάτι που πρέπει να προσέξει ο προγραμματιστής είναι, να βεβαιωθεί ότι τα IDs των input πεδίων πρέπει να είναι ‘username’ και ‘password’ αντίστοιχα, διότι το Spring Security θα ψάξει για παραμέτρους με τα συγκεκριμένα IDs.

3.2.6.2 JSF Security Αυθεντικοποίηση και Εξουσιοδότηση

Επίσης προσδιορίζεται το `prependId="false"` στις φόρμες, διότι διαφορετικά του JSF Framework θα βάλει prefix στις παραμέτρους με το ID της φόρμας. Μία άλλη επιλογή θα ήταν να γίνει Override των default field names με τη χρήση της `usernameParameter()` και της `passwordParameter()` μεθόδων στο formLogin() που θα χρησιμοποιηθεί παρακάτω:

```
<?xml version="1.0" encoding="UTF-8"?>
<html            xmlns="http://www.w3.org/1999/xhtml"           xmlns:h="http://java.sun.com/jsf/html"
xmlns:f="http://xmlns.jcp.org/jsf/core"                             xmlns:p="http://primefaces.org/ui"
xmlns:pe="http://primefaces.org/ui/extensions">
        <h:head>
        <title>Login</title>
        <h:outputStylesheet name="/css/login.css" />
        </h:head>
        <h:body>
        <h:form prependId="false">
                <p:panelGrid columns="1" styleClass="ui-fluid center ui-noborder">
                <h2>Please login</h2>
                <p:outputLabel value="Login failed!" styleClass="red" rendered="${!empty param['error']}"
/>
                <p:inputText id="username" placeholder="User name" />
                <p:password id="password" placeholder="Password" />
                <p:commandButton value="Login" ajax="false" />
                </p:panelGrid>
        </h:form>
        </h:body>
</html>
```

Αν το SpringSecurity είναι στο classpath του SpringBoot αυτόματα θα διαμορφωθεί ένα πλήθος από βασικά χαρακτηριστικά ασφάλειας για μία web εφαρμογή.

Για να γίνει customize της ασφάλειας της web εφαρμογής, πρέπει να δημιουργηθεί μια SecurityConfig κλάση που επεκτείνει το WebSecurityConfigurerAdapter, το οποίο είναι μία βασική κλάση που προσφέρει μία default security configuration. Η κλάση γίνεται annotate με το @EnableSecurity για την ενεργοποίηση της Spring Security's web security υποστήριξη.

Επίσης πρέπει να γίνει override της μεθόδου `configure(HttpSecurity http)` για να οριστεί πότε και πώς οι χρήστες πρέπει να αυθεντικοποιούνται. Προσδιορίζοντας το `authorizeRequests()anyRequest().authenticated()` διαβεβαιώνεται ότι κάθε request στην εφαρμογή απαιτεί ο χρήστης να είναι αυθεντικοποιημένος.

Οι στατικοί πόροι (CSS, JavaScript, ...) χρειάζεται να είναι προσβάσιμοι από τον καθένα, διαφορετικά η εμφάνιση και η αίσθηση του login page δεν θα είναι ίδια με την υπόλοιπη εφαρμογή. Προσθέτοντας το `antMatchers("/javax.faces.resource/**").permitAll()` επιτρέπει στον καθένα να έχει πρόσβαση στο URL που ξεκινάει με `/javax.faces.resource/` (όπου βρίσκονται οι στατικοί πόροι σε μία JSF εφαρμογή).

Επιπλέον αφού ορίστηκε ένα custom PrimeFaces login page, πρέπει να προσδιοριστεί η τοποθεσία του με χρήση του `formLogin().loginPage("/login.xhtml")`, έτσι όταν χρειάζεται αυθεντικοποίηση, ο browser να ανακατευθύνει τον χρήστη στο `/login.xhtml`. Επίσης χρειάζεται το permitAll() στο login page, ώστε ο καθένας να έχει πρόσβαση στη σελίδα, αλλιώς θα εγκλωβιστεί ο χρήστης σε μια λούπα ανακατευθύνσεων.

Αν το login αποτύχει, γίνεται ανακατεύθυνση στην ίδια σελίδα login με μια HTTP παράμετρο `error=true` με τη χρήση του `failureUrl("/login.xhtml?error=true")`. Αυτό επιτρέπει την εμφάνιση μηνύματος σφάλματος στον χρήστη.

Με τη χρήση του `WebSecurityConfigureAdapter`, η λειτουργίες του logout εφαρμόζονται αυτόματα. Η προεπιλογή είναι ότι όταν κατευθυνθεί στο /logout θα αποσυνδέσει το χρήστη. Προσδιορίζοντας το `logout().logoutSuccessUrl("/login.xhtml")` ο χρήστης ανακατευθύνεται στο login page, αν έχει κάνει επιτυχώς αποσύνδεση.

Το Spring Security εφαρμόζει μέτρα για την πρόληψη CSRF επιθέσεων απαιτώντας ένα τυχαία παραγόμενο token σαν HTTP παράμετρο. Παρόλα αυτά επειδή το JSF 2.2 περιέχει ήδη μία ξεχωριστή προστασία έναντι σε CSRF επιθέσεις, πρέπει να γίνει απενεργοποίηση της προστασίας από το Spring Security με το `http.csrf().disable()`.

Εκτός από αυτά, στο παράδειγμα πρέπει να γίνει override του default χρήστη `AuthenticationManager` που το Spring Boot θέτει με το να κάνει auto-wire έναν `AuthenticationManagerBuilder` στο `configureGlobal()` της `SecurityConfig` `@Configuration` κλάσης.

Επίσης για αυτό το παράδειγμα θα χρησιμοποιηθεί in-memory αυθεντικοποίηση στην οποία ορίζονται δύο χρήστες (‘john.doe’ και ‘jane.doe’) με διαφορετικούς ρόλους (‘USER’ και ‘ADMIN’).

Επιπλέον η αποθήκευση κωδικού στο Spring Security υπέστει μία μεγάλη διόρθωση από την version 5. Λόγω αυτών των αλλαγών πρέπει να υπάρχει το prefix {noop} στους κωδικούς, ώστε ο DelegatingPasswordEncoder να χρησιμοποιήσει τον NoOpPasswordEncoder για να τους επαληθεύσει.

```
package com.codenotfound.primefaces;

import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

  @Override
  protected void configure(HttpSecurity http) throws Exception {
    // require all requests to be authenticated except for the resources
    http.authorizeRequests().antMatchers("/javax.faces.resource/**")
        .permitAll().anyRequest().authenticated();
    // login
    http.formLogin().loginPage("/login.xhtml").permitAll()
        .failureUrl("/login.xhtml?error=true");
    // logout
    http.logout().logoutSuccessUrl("/login.xhtml");
    // not needed as JSF 2.2 is implicitly protected against CSRF
    http.csrf().disable();
  }

  @Autowired
  public void configureGlobal(AuthenticationManagerBuilder auth)
      throws Exception {
    auth.inMemoryAuthentication().withUser("john.doe")
        .password("{noop}1234").roles("USER").and()
        .withUser("jane.doe").password("{noop}5678").roles("ADMIN");
  }
}
```

3.2.6.3 Role Based Access Control (RBAC)

Το Spring Security έχει το δικό του taglib που προσφέρει βασική υποστήριξη για την πρόσβαση σε πληροφορίες ασφάλειας και εφαρμογή περιορισμών ασφαλείας σε JSPs.

Στη σελίδα helloworld.xhtml προσθέτουμε ένα στοιχείο στο οποίο χρησιμοποιούμε το authorize tag με σκοπό την εμφάνιση ενός μηνύματος, σε περιπτωση που ο χρήστης έχει τον ρόλο 'USER' ή 'ADMIN'. Επίσης προστίθεται ένα logout κουμπί <p: commandButton> στο κάτω μέρος της σελίδας. Να σημειωθεί ότι δεν χρησιμοποιείται το <h: form>, αφού το JSF θέτει την action της φόρμας αυτόματα στην τρέχουσα σελίδα και αυτό που είναι θεμιτό αν γίνει είναι η πλοήγηση στο URL του default logout που προσφέρεται από το Spring Security.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:p="http://primefaces.org/ui"
  xmlns:sec="http://www.springframework.org/security/tags">

<h:head>
  <title>PrimeFaces Hello World Example</title>
  <h:outputStylesheet name="/css/main.css" />
</h:head>

<h:body>

  <div class="authorization-div">
    <sec:authorize access="hasRole('ROLE_USER')">
      <p:outputLabel value="You have the USER role" />
    </sec:authorize>
    <sec:authorize access="hasRole('ROLE_ADMIN')">
      <p:outputLabel value="You have the ADMIN role" />
    </sec:authorize>
  </div>

  <h:form>
    <p:panel header="PrimeFaces Hello World Example">
      <h:panelGrid columns="2" cellpadding="4">

        <h:outputText value="First Name: " />
        <p:inputText value="#{helloWorld.firstName}" />

        <h:outputText value="Last Name: " />
        <p:inputText value="#{helloWorld.lastName}" />

        <p:commandButton value="Submit" update="greeting"
          oncomplete="PF('greetingDialog').show()" />
      </h:panelGrid>
    </p:panel>
```

```
    <p:dialog header="Greeting" widgetVar="greetingDialog"
      modal="true" resizable="false">
      <h:panelGrid id="greeting" columns="1" cellpadding="4">
        <h:outputText value="#{helloWorld.showGreeting()}" />
      </h:panelGrid>
    </p:dialog>
  </h:form>

  <h:form onsubmit="this.action='#{request.contextPath}/logout';"
    class="logout-form">
    <p:commandButton value="Logout" ajax="false" />
  </h:form>

</h:body>
</html>
```

Εν κατακλείδι, παρατίθεται ένα παράδειγμα στο οποίο φαίνεται πώς μπορεί να γίνει προσβάσιμο το τρέχων αντικείμενο Authentication που είναι αποθηκευμένο στο security context. Επίσης για την εμφάνιση του ονόματος του αυθεντικοποιημένου χρήστη χρησιμοποιείται η μέθοδος `SecurityContextHolder.getContext().getAuthentication()`.

```
package com.codenotfound.primefaces;

import javax.inject.Named;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;

@Named
public class HelloWorld {

  private String firstName = "";
  private String lastName = "";

  public String getFirstName() {
    return firstName;
  }

  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }

  public String getLastName() {
    return lastName;
  }

  public void setLastName(String lastName) {
    this.lastName = lastName;
  }

  public String showGreeting() {
    Authentication authentication =
        SecurityContextHolder.getContext().getAuthentication();

    return "Hello " + authentication.getName() + "!";
```

```
    }
}
```

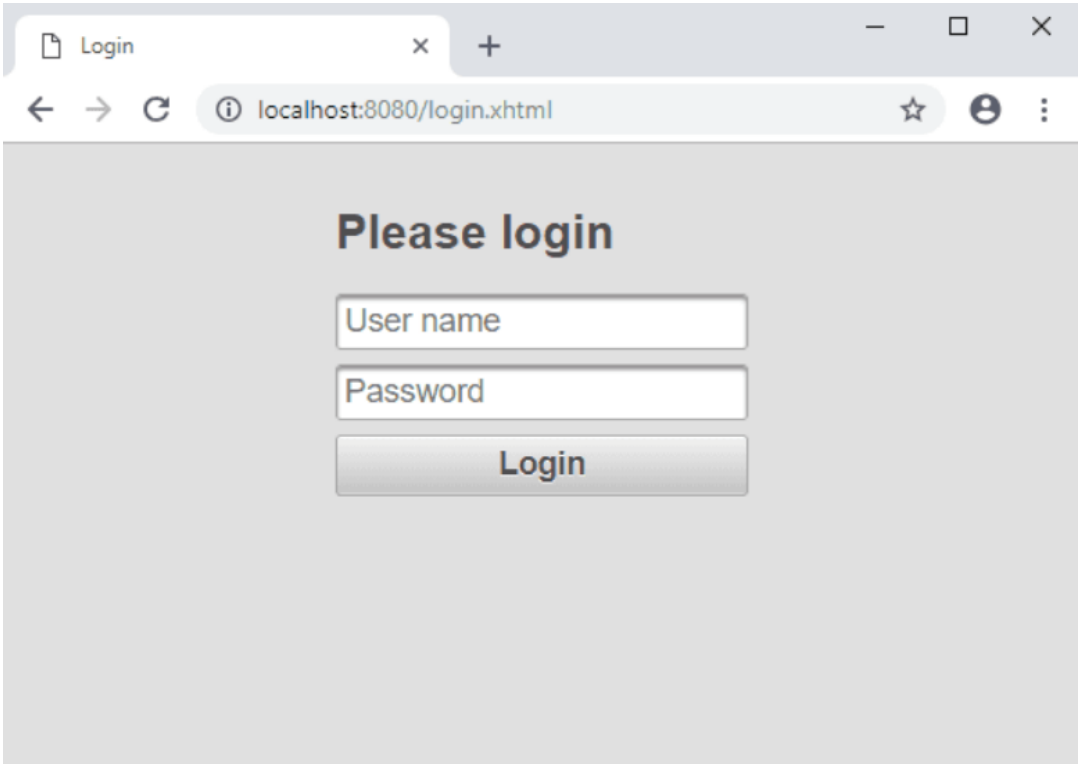
3.2.6.4 Δοκιμή των JSF ρόλων ασφάλειας

Πρώτα πρέπει να εκκινηθεί ο Maven με την εντολή:

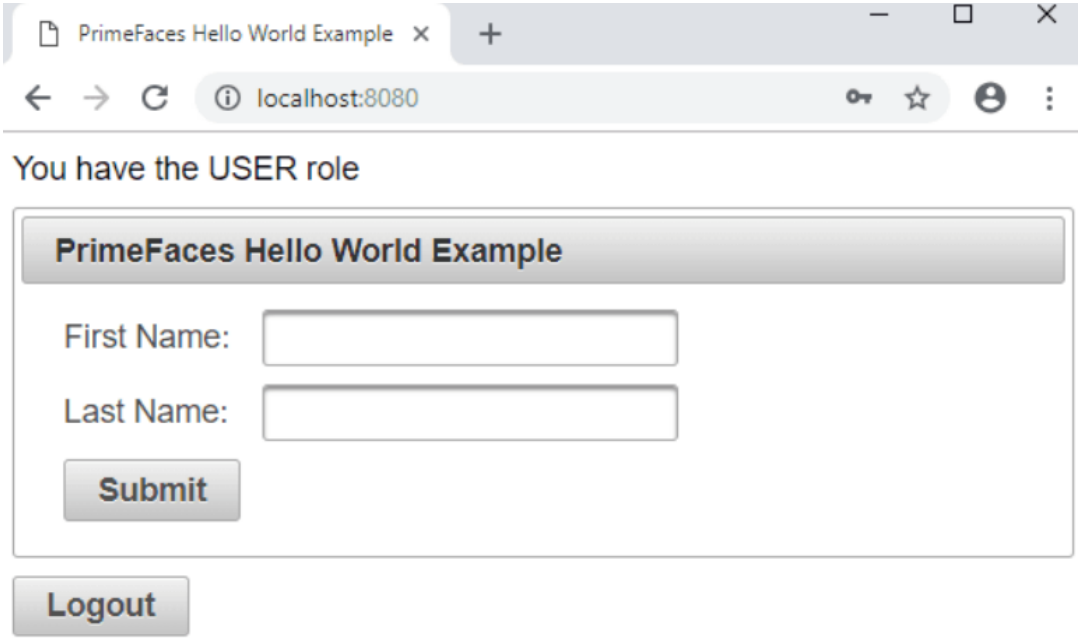
```
mvn spring-boot:run
```

Όταν το Spring Boot ξεκινήσει, πρέπει να ανοίξει ένας web browser και να χρησιμοποιηθεί το ακόλουθο URL: http://localhost:8080/helloworld.xhtml.

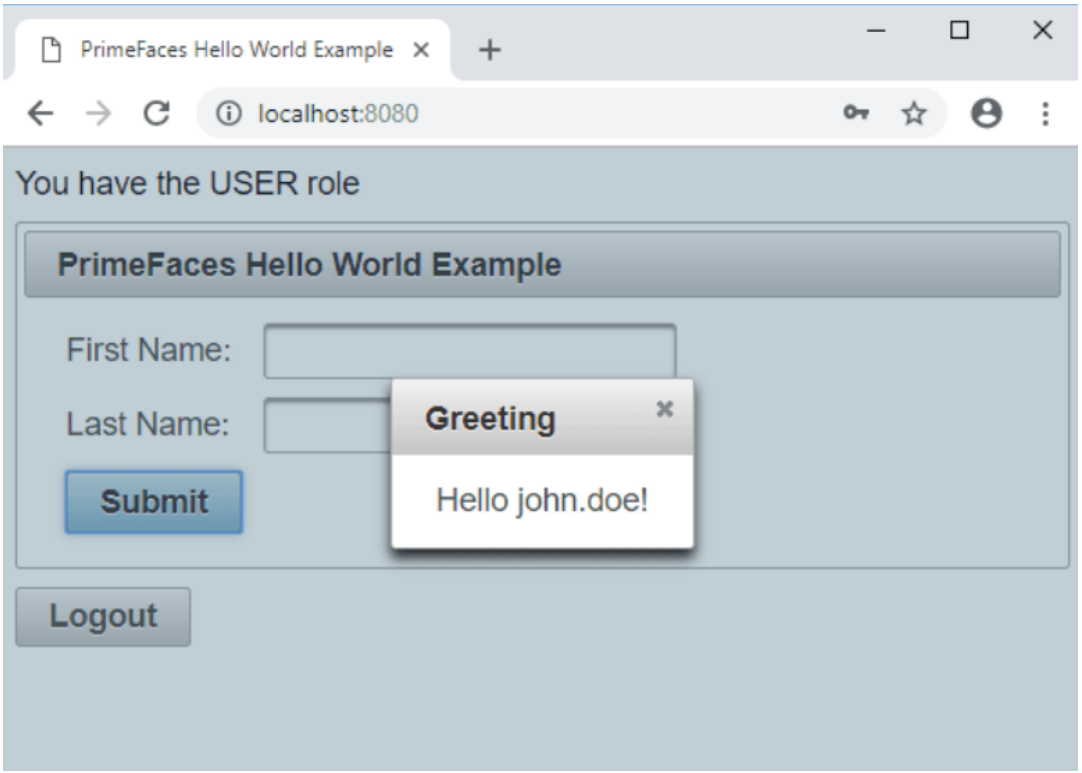
Εφόσον στην αρχή ο χρήστης δεν είναι αυθεντικοποιημένος, το Spring Security θα κάνει ανακατεύθυνση του χρήστη στο login page.



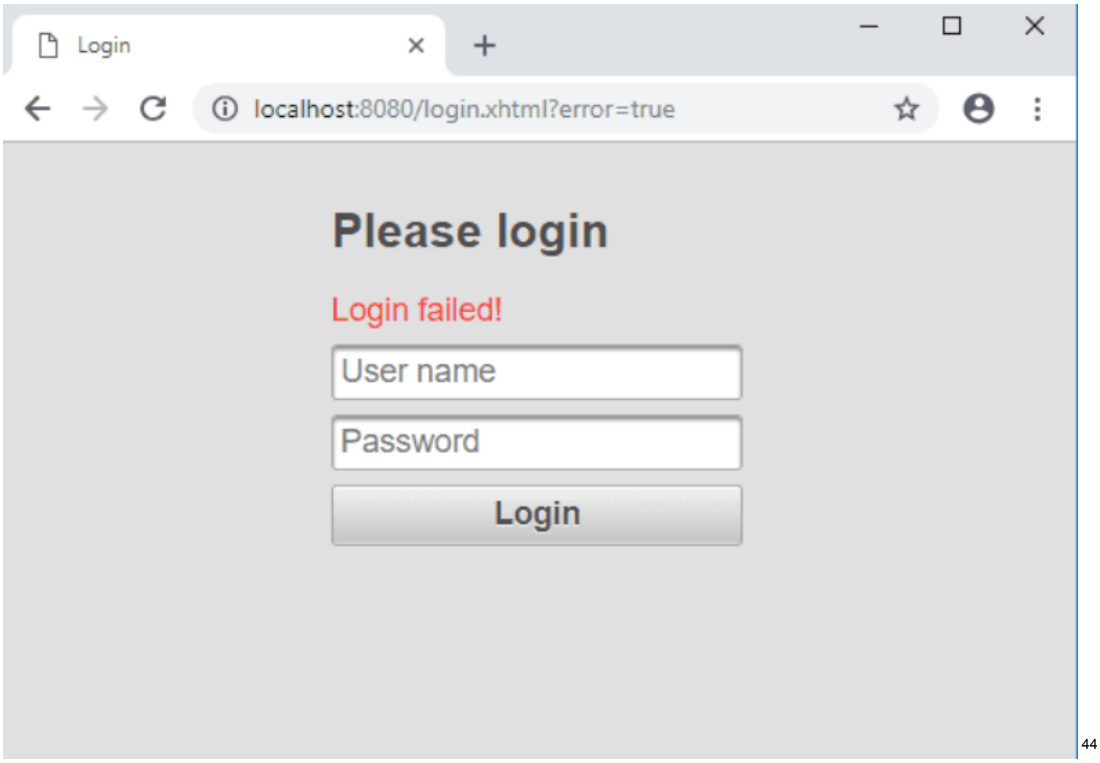
Στη συνέχεια αν χρησιμοποιηθεί το username=john.doe και password=1234 και πατηθεί το Login θα εμφανιστεί η Hello World σελίδα και ο ρόλος του χρήστη εμφανίζεται στο πάνω μέρος της σελίδας.



Αν πατηθεί το Submit κουμπί, τότε θα εμφανιστεί το username που χρησιμοποιήθηκε στο login.



Επίσης αν το Logout κουμπί πατηθεί θα γίνει ανακατεύθυνση στο login page. Τέλος αν τοποθετηθεί το ίδιο username αλλά με λάθος password θα εμφανιστεί ένα μήνυμα σφάλματος.



[44]

3.3 Κρυπτογραφία

Η κρυπτογραφία χρησιμοποιείται για την κρυπτογράφηση ή την αποκρυπτογράφηση δεδομένων. Χρησιμεύει σε ψηφιακές υπογραφές και σε έλεγχο μηνυμάτων, στη διαχείριση κλειδιών και στον υπολογισμό τα κρυπτογραφικών κατακερματισμένων. Η κρυπτογραφία είναι μια τεχνική κρυπτογράφησης των δεδομένων καθαρού κειμένου σε έναν scrambled κώδικα. Αυτά τα κρυπτογραφημένα δεδομένα αποστέλλονται μέσω δημόσιου ή ιδιωτικού δικτύου προς τον προορισμό για να διασφαλιστεί η εμπιστευτικότητα. Αυτά τα κρυπτογραφημένα δεδομένα είναι γνωστά και ως "Ciphertext". Ισχυρά κλειδιά κρυπτογράφησης χρησιμοποιούνται για να αποφευχθεί το σπάσιμο των κλειδιών. Ο στόχος της κρυπτογραφίας δεν αφορά αποκλειστικά την εμπιστευτικότητα, αφορά επίσης την ακεραιότητα, τον έλεγχο ταυτότητας και τη μη αποποίηση.
Η κρυπτογραφία παίζει σημαντικό ρόλο στην προστασία εφαρμογών από επιθέσεις και αποκάλυψη ευαίσθητων διαπιστευτηρίων και δεδομένων. Η κρυπτογράφηση δεδομένων, διαπιστευτηρίων και scripts δημιουργεί ένα αμυντικό επίπεδο ενάντια σε διάφορες απειλές χρησιμοποιώντας κρυπτογράφηση και ψηφιακές υπογραφές.

3.3.1 Ορισμοί Κρυπτογραφίας

Cipher
Οποιαδήποτε μέθοδος μετατροπής ενός μηνύματος για να αποκρύψει το νόημά του. Ο όρος χρησιμοποιείται επίσης συνώνυμα με τον όρο ciphertext ή cryptogram και αναφέρεται στην κρυπτογραφημένη μορφή του μηνύματος. [45]

Digital Signature
Η ψηφιακή υπογραφή είναι ένα μαθηματικό σχήμα για την επαλήθευση της αυθεντικότητας των ψηφιακών μηνυμάτων ή εγγράφων. Μια έγκυρη ψηφιακή υπογραφή, δίνει στον παραλήπτη έναν πολύ ισχυρό λόγο να πιστεύει ότι το μήνυμα δημιουργήθηκε από έναν γνωστό αποστολέα (έλεγχο ταυτότητας) και ότι το μήνυμα δεν άλλαξε κατά τη μεταφορά (ακεραιότητα).[46]

---

[44] "JSF Primefaces Spring Security Example - CodeNotFound.com." 7 Δεκ. 2018, https://codenotfound.com/jsf-primefaces-spring-security-example.html. Πρόσβαση στις 11 Οκτ. 2020.
[45] "Cipher | cryptology | Britannica." https://www.britannica.com/topic/cipher. Πρόσβαση στις 11 Οκτ. 2020.
[46] "Digital signature - Wikipedia." https://en.wikipedia.org/wiki/Digital_signature. Πρόσβαση στις 11 Οκτ. 2020.

Απορρητο/Confidentiality
Το απόρρητο είναι περίπου ισοδύναμο με το privacy. Τα μέτρα που λαμβάνονται για τη διασφάλιση του confidentiality έχουν σχεδιαστεί για να αποτρέπουν την έκθεση ευαίσθητων πληροφοριών σε λάθος άτομα, διασφαλίζοντας παράλληλα ότι τα εξουσιοδοτημένα άτομα έχουν πρόσβαση σε αυτές. Είναι συνηθισμένο να κατηγοριοποιούνται τα δεδομένα σύμφωνα με το ποσό και τον τύπο της ζημιάς που θα μπορούσε να γίνει σε περίπτωση που πέσουν σε λάθος χέρια. Στη συνέχεια μπορούν να εφαρμοστούν περισσότερα ή λιγότερο αυστηρά μέτρα σύμφωνα με αυτές τις κατηγορίες.

Ακεραιότητα/Integrity
Η ακεραιότητα περιλαμβάνει τη διατήρηση της συνέπειας, της ακρίβειας και της αξιοπιστίας των δεδομένων καθ 'όλη τη διάρκεια του κύκλου ζωής του. Τα δεδομένα δεν πρέπει να αλλάζουν κατά τη μεταφορά και πρέπει να ληφθούν μέτρα για να διασφαλιστεί ότι τα δεδομένα δεν μπορούν να τροποποιηθούν από μη εξουσιοδοτημένα άτομα (για παράδειγμα, κατά παράβαση του απορρήτου). Αυτά τα μέτρα περιλαμβάνουν δικαιώματα αρχείων και στοιχεία ελέγχου πρόσβασης χρήστη. [47]

Διαθεσιμότητα/Availability
Για να είναι χρήσιμο ένα σύστημα πληροφοριών, πρέπει να είναι διαθέσιμο σε εξουσιοδοτημένους χρήστες. Τα μέτρα διαθεσιμότητας προστατεύουν την έγκαιρη και αδιάκοπη πρόσβαση στο σύστημα. Μερικές από τις πιο θεμελιώδεις απειλές για τη διαθεσιμότητα είναι μη κακόβουλης φύσης και περιλαμβάνουν αστοχίες υλικού, μη προγραμματισμένη διακοπή λειτουργίας λογισμικού και ζητήματα εύρους ζώνης δικτύου. Οι κακόβουλες επιθέσεις περιλαμβάνουν διάφορες μορφές σαμποτάζ που προορίζονται να προκαλέσουν βλάβη σε έναν οργανισμό, απαγορεύοντας την πρόσβαση των χρηστών στο σύστημα πληροφοριών.[48]

3.3.2 Java Security με Κρυπτογραφία

Η Java προσφέρει κρυπτογραφική λειτουργικότητα με χρήση δύο APIs:
1. JCA (Java Cryptography Architecture)
2. JCE (Java Cryptography Extensions)

3.3.2.1 Java Cryptography Architecture (JCA)

Η Java πλατφόρμα δίνει ιδιαίτερη έμφαση στην ασφάλεια, συμπεριλαμβανομένης της ασφάλειας γλώσσας, της κρυπτογράφησης, της υποδομής δημοσίου κλειδιού, του ελέγχου ταυτότητας, του ασφαλή ελέγχου και πρόσβασης επικοινωνίας.

Το JCA είναι ένα σημαντικό κομμάτι της πλατφόρμας και περιέχει μια αρχιτεκτονική "provider" και ένα σύνολο API για ψηφιακές υπογραφές, σύνοψη μηνυμάτων (κατακερματισμούς), πιστοποιητικά και επικύρωση πιστοποιητικών, κρυπτογράφηση (συμμετρική / ασύμμετρη κρυπτογράφηση ροής/block), δημιουργία κλειδιών και διαχείριση, και ασφαλή δημιουργία τυχαίων αριθμών. Αυτά τα APIs επιτρέπουν στους προγραμματιστές να ενσωματώνουν εύκολα την ασφάλεια στον κώδικα της εφαρμογής τους. Οι ακόλουθες engine κλάσεις είναι διαθέσιμες:

- SecureRandom: χρησιμοποιείται για τη δημιουργία τυχαίων ή ψευδο-τυχαίων αριθμών.
- MessageDigest: χρησιμοποιείται για τον υπολογισμό της σύνοψης μηνυμάτων (κατακερματισμός) συγκεκριμένων δεδομένων.
- Signature:αρχικοποιείται με κλειδιά, χρησιμοποιείται για την υπογραφή δεδομένων και την επαλήθευση ψηφιακών υπογραφών.
- Cipher: αρχικοποιείται με κλειδιά, χρησιμοποιείται για κρυπτογράφηση / αποκρυπτογράφηση δεδομένων. Υπάρχουν διάφοροι τύποι αλγορίθμων: συμμετρική μαζική κρυπτογράφηση (π.χ. AES), ασύμμετρη κρυπτογράφηση (π.χ. RSA) και κρυπτογράφηση βάση κωδικού πρόσβασης (π.χ. PBE).
- Message Authentication Codes (MAC): δημιουργούν επίσης τιμές κατακερματισμού, αλλά αρχικοποιούνται με κλειδιά για την προστασία της ακεραιότητας των μηνυμάτων.

[47] "confidentiality, integrity, and availability (CIA triad) - WhatIs.com." https://whatis.techtarget.com/definition/Confidentiality-integrity-and-availability-CIA. Πρόσβαση στις 11 Οκτ. 2020.
[48] "Confidentiality, Integrity and Availability - The CIA Triad ...." 4 Αυγ. 2018, https://www.certmike.com/confidentiality-integrity-and-availability-the-cia-triad/. Πρόσβαση στις 11 Οκτ. 2020.

- KeyFactory: χρησιμοποιείται για τη μετατροπή υπαρχόντων αδιαφανών κρυπτογραφικών κλειδιών τύπου Key σε βασικές προδιαγραφές (διαφανείς αναπαραστάσεις του υποκείμενου βασικού υλικού) και το αντίστροφο.
- SecretKeyFactory: χρησιμοποιείται για τη μετατροπή υπαρχόντων αδιαφανών κρυπτογραφικών κλειδιών τύπου SecretKey σε βασικές προδιαγραφές (διαφανείς αναπαραστάσεις του υποκείμενου βασικού υλικού) και αντίστροφα. Το SecretKeyFactorys είναι εξειδικευμένα KeyFactorys που δημιουργούν μόνο μυστικά (συμμετρικά) κλειδιά.
- KeyPairGenerator: χρησιμοποιείται για τη δημιουργία ενός νέου ζεύγους δημόσιων και ιδιωτικών κλειδιών κατάλληλων για χρήση με έναν καθορισμένο αλγόριθμο.
- KeyGenerator: χρησιμοποιείται για τη δημιουργία νέων μυστικών κλειδιών για χρήση με έναν καθορισμένο αλγόριθμο.
- KeyAgreement: χρησιμοποιείται από δύο ή περισσότερα μέρη για να συμφωνήσουν και να καθορίσουν ένα συγκεκριμένο κλειδί για χρήση για μια συγκεκριμένη κρυπτογραφική λειτουργία.
- AlgorithmParameters: χρησιμοποιείται για την αποθήκευση των παραμέτρων για έναν συγκεκριμένο αλγόριθμο, συμπεριλαμβανομένης της κωδικοποίησης παραμέτρων και της αποκωδικοποίησης.
- AlgorithmParameterGenerator : χρησιμοποιείται για τη δημιουργία ενός συνόλου AlgorithmParameters κατάλληλο για έναν καθορισμένο αλγόριθμο.
- KeyStore: χρησιμοποιείται για τη δημιουργία και τη διαχείριση ενός keystore. Ένα keystore είναι μια βάση δεδομένων κλειδιών. Τα ιδιωτικά κλειδιά σε ένα keystore συνδέονται με μια αλυσίδα πιστοποιητικών, η οποία επικυρώνει το αντίστοιχο δημόσιο κλειδί. Το keystore περιέχει επίσης πιστοποιητικά από αξιόπιστες οντότητες.
- CertificateFactory: χρησιμοποιείται για τη δημιουργία πιστοποιητικών δημόσιου κλειδιού και λίστες ανάκλησης πιστοποιητικών (CRLs)
- CertPathBuilder: χρησιμοποιείται για την κατασκευή αλυσίδων πιστοποιητικών (επίσης γνωστή ως διαδρομές πιστοποίησης).
- CertPathValidator:χρησιμοποιείται για την επικύρωση αλυσίδων πιστοποιητικών.
- CertStore: χρησιμοποιείται για την ανάκτηση πιστοποιητικών και CRL από ένα αποθετήριο.

[49]

3.3.2.2 Java Cryptography Extension (JCE)

Το Java Cryptography Extension JCE είναι μία επίσημη τυπική επέκταση στην Java πλατφόρμα και επίσης είναι μέρος της Java Cryptography Architecture (JCA). Προσφέρει μία υλοποίηση αλγορίθμων για παραγωγή κλειδιών, κρυπτογράφηση και Message Authentication Codes (MACs). Το JCE περιλαμβάνει διεπαφές και υλοποιήσεις για Message digests, digital signatures κλπ. Τα πακέτα των βασικών κλάσεων και οι διεπαφές είναι:

- ❏ Cipher
- ❏ KeyGenerator
- ❏ SecretKeyFactory
- ❏ Key Arrangement
- ❏ MAC

Το Java cryptography API είναι χωρισμένο στα παρακάτω πακέτα:

- ❏ java.security
- ❏ java.security.cert
- ❏ java.security.spec
- ❏ java.security.interfaces
- ❏ javax.crypto
- ❏ javax.crypto.spec
- ❏ javax.crypto.interfaces[50]

[49] "Java Cryptography Architecture (JCA) Reference Guide." https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html. Πρόσβαση στις 11 Οκτ. 2020.

[50] "Java Cryptography Extension (JCE) - IBM Knowledge Center." https://www.ibm.com/support/knowledgecenter/SSYKE2_7.1.0/com.ibm.java.security.component.71.doc/security-component/JceDocs/jce.html. Πρόσβαση στις 11 Οκτ. 2020.

3.3.2.3 Ciphers Class

Η Cipher (javax.crypto.Cipher) κλάση αναπαριστά το κρυπτογραφικό αλγόριθμο, που μπορεί να χρησιμοποιηθεί και για κρυπτογράφηση αλλά και για αποκρυπτογράφηση δεδομένων. Η getInstance() μέθοδος της cipher κλάσης παίρνει ως όρισμα μία συμβολοσειρά που αναπαριστά την απαιτούμενη μετατροπή και επιστρέφει ένα αντικείμενο Cipher που υλοποιεί την δοσμένη μετατροπή. Παρακάτω φαίνεται πώς δημιουργείται το αντικείμενο Cipher κάνοντας χρήση της μεθόδου getInstance():

```
//Creating a Cipher object
Cipher = Cipher.getInstance("AES/CBC/PKCS5-Padding");
```

Όπου

- Η συγκεκριμένη μέθοδος δημιουργεί ένα Cipher instance που χρησιμοποιείται εσωτερικά στον αλγόριθμο κρυπτογράφησης AES.
- Το CBC είναι ένα mode του AES αλγόριθμου.
- Το PKCS5Padding μέρος δηλώνει πως ο αλγόριθμος AES πρέπει να χειριστεί το τελευταία bytes των δεδομένων για να κρυπτογραφήσει, αν τα δεδομένα δεν είναι ευθυγραμμισμένα με 64-bit ή 128 bit block size.

Αρχικοποίηση του Cipher

Η init() μέθοδος της cipher κλάσης παίρνει δύο παραμέτρους, έναν ακέραιο που αναπαριστά τον τρόπο λειτουργίας (κρυπτογράφηση/ αποκρυπτογράφηση) και ένα key object που χρησιμοποιείται για να αναπαραστήσει το δημόσιο κλειδί. Η αρχικοποίηση του αντικειμένου Cipher γίνεται με χρήση της μεθόδου init() όπως φαίνεται παρακάτω:

```
// initializing a Cipher object
 byte[] keyBytes = new byte[]{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ,11, 12, 13, 14, 15};
String algorithm ="RawBytes";
SecretKeySpec key = new SecretKeySpec(keyBytes, algorithm);
cipher.init(Cipher.ENCRYPT_MODE, publicKey);
```

Για την αποκρυπτογράφηση δεδομένων:

```
cipher.init(Cipher.DECRYPT_MODE, key);
```

Κρυπτογράφηση ή αποκρυπτογράφηση κειμένου

Όταν ο cipher είναι σωστά αρχικοποιημένος, τότε μπορεί να ξεκινήσει η κρυπτογράφηση και η αποκρυπτογράφηση δεδομένων. Αυτό μπορεί να γίνει με τις μεθόδους update() ή doFinal(). Η update() μέθοδος της κλάσης cipher χρησιμοποιείται όταν έχουμε να κρυπτογραφήσουμε πολλαπλά blocks of data.

```
byte[] data1 = "abcdefghijklmnopqrstuvwxyz".getBytes("UTF-8");
byte[] data2 = "zyxwvutsrqponmlkjihgfedcba".getBytes("UTF-8");
byte[] data3 = "01234567890123456789012345".getBytes("UTF-8");

byte[] cipherText1 = cipher.update(data1);
byte[] cipherText2 = cipher.update(data2);
byte[] cipherText3 = cipher.doFinal(data3);
```

Διαφορετικά αν πρέπει να κρυπτογραφεί μόνο ένα μπλοκ, τότε χρησιμοποιείται η doFinal() με τα δεδομένα που πρέπει να κρυπτογραφηθούν ή να αποκρυπτογραφηθούν.

```
byte[] plainText  = "abcdefghijklmnopqrstuvwxyz".getBytes("UTF-8");
byte[] cipherText = cipher.doFinal(plainText);
```

Παράδειγμα
Κρυπτογράφηση

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.Signature;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
public class CipherExample1{
	public static void main(String args[]) throws Exception{
		//Creating a Signature object
		Signature sign = Signature.getInstance("SHA256withRSA");
		//Creating KeyPair generator object
		KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("RSA");
//Initializing the key pair generator
		keyPariGen.initialize(2048);
		//Generating the pair of keys
		KeyPair pair = keyPairGen.generateKeyPair();
		//Creating a Cipher object
		Cipher Cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
		//Initializing a Cipher Object
		cipher.init(Cipher.ENCRYPT_MODE, pair.getPublic());
		//Adding data to the cipher
		byte[] input = "Hello Everyone".getBytes();
		cipher.update(input);
		//encrypting the data
		byte[] cipherText = cipher.doFinal();
		System.out.println(new String(cipherText, "UTF8"));}
}
```

Αποτέλεσμα

```
4??*{??▯?l0▯M?▯?+D??G?<???&▯?▯?@?`k?Q▯
?15?Ss??P???ja?▯?▯▯?G?????D??H.??C?h?Q?.?^?????
4?A??nU????G?}????o[5?▯J▯??È?;?r2q??▯c?▯o▯?*jKC??I▯?ZE:▯???%?#??
???t4
?▯@Y~?ak?B?#▯??y???W??~@eklW?¨d▯.j?Q?i{=?▯A*?A??s?▯"V????7?7?7]?? ?>????7??k|    W
```

Αποκρυπτογράφηση

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
public class CipherDecryptExample{
	public static void main(String args[]) throws Exception{
		//Creating KeyPair generator object
		KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("RSA");
		//Initializing the key pair generator
		keyPariGen.initialize(2048);
```

```
            //Generating the pair of keys
            KeyPair pair = keyPairGen.generateKeyPair();
            //Creating a Cipher object
            Cipher Cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
            //Initializing a Cipher Object
            cipher.init(Cipher.ENCRYPT_MODE, pair.getPublic());
            //Adding data to the cipher
            byte[] input = "Hello Everyone".getBytes();
            cipher.update(input);
            //encrypting the data
            byte[] cipherText = cipher.doFinal();
            System.out.println(new String(cipherText, "UTF8"));
            //Initializing the same cipher for decryption
            cipher.init(cipher.DECRYPT_MODE, pair.getPrivate());
            //Decrypting the text
            byte[] decipheredText = cipher.doFinal(cipherText);
            System.out.println(new String(decipheredText));
    }
}
```

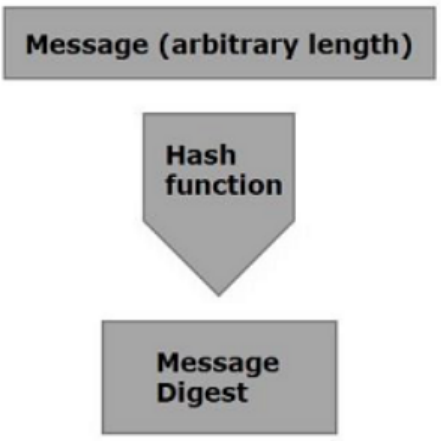
Αποτέλεσμα

```
??l?+?B?FCE?y???Rv?????R-??J?▯???▯?
▯?e???▯Q?]I?g?K???q▯Z▯?? ?*▯?(<?P?r?
???▯O5{??
??uX▯`?<?=??:?q?▯????q7??Jp?@?n-
Hello Everyone
```

[51]

3.3.2.4 Message Digest

Οι hash functions μετατρέπουν μία αριθμητική τιμή σε μία άλλη ακολουθία συγκεκριμένου μήκους. Η τιμή που επιστρέφεται από μία hash function είναι γνωστή και ως message digest. Το message digest είναι μία μέθοδος για την επαλήθευση της ακεραιότητας ενός μηνύματος. Πρώτα τα message digest παίρνουν ως είσοδο ένα μήνυμα και δημιουργούν ένα block από bits, που αναπαριστά το αποτύπωμά του μηνύματος. Μία μικρή αλλαγή στο μήνυμα παράγει μεγάλη αλλαγή στο αποτύπωμα.



Υπολογισμός ενός Message Digest

[51] "Java Cipher - Jenkov Tutorials." 15 Νοε. 2019, http://tutorials.jenkov.com/java-cryptography/cipher.html. Πρόσβαση στις 12 Οκτ. 2020.

Το message digest υπολογίζεται πριν κρυπτογραφηθούν τα δεδομένα και μετά στέλνονται τα κρυπτογραφημένα δεδομένα μαζί με το message digest. Όταν λαμβάνεται το μήνυμα γίνεται η αποκρυπτογράφηση και έπειτα υπολογίζεται το message digest από το αποκρυπτογραφημένο κείμενο, κι αν ταιριάζει σημαίνει ότι το μήνυμα δεν έχει τροποποιηθεί κατά τη μεταφορά του. Διαφορετικά αποδεικνύεται ότι το μήνυμα τροποποιήθηκε. Για να υπολογιστεί το message digest χρησιμοποιείται το java.security.MessageDigest.

Δημιουργία ενός Message Digest

Η κλάση Message Digest χρησιμοποιεί τη μέθοδο getInstance(), η οποία δέχεται μία συμβολοσειρά ώστε να αναγνωρίσει το όνομα του αλγόριθμου που θα χρησιμοποιηθεί ή επιστρέφει ένα MessageDigest που υλοποιεί τον αναγνωρισμένο αλγόριθμο.

```
MessageDigest md = MessageDigest.getInstance("SHA-256");
```

Αυτή μέθοδος παράγει ένα instance του MessageDigest χρησιμοποιώντας τον αλγόριθμο κρυπτογραφικού κατακερματισμού SHA-256 για τον υπολογισμό του message digest.

Μετά τη δημιουργία του message digest object() Πρέπει να δοθεί με τη μέθοδο update() το κείμενο που θα κρυπτογραφεί ώστε να παραχθεί από αυτό το message digest.

```
md.update(msg.getBytes());
```

Έπειτα πρέπει να χρησιμοποιήσουμε τη μέθοδο digest(). Η κλάση MessageDigest υπολογίζει την συνάρτηση κατακερματισμού του τρέχοντος object και επιστρέφει το message digest σε μορφή byte array.

```
byte[] digest = md.digest();
```

Παράδειγμα

```
import java.security.MessageDigest;
import java.util.Scanner;
public class MessageDigestExample {
	public static void main(String args[]) throws Exception{
	//Reading data from user
	Scanner sc = new Scanner(System.in);
	System.out.println("hello everyone");
	String message = sc.nextLine();
	//Creating the MessageDigest object
	MessageDigest md = MessageDigest.getInstance("SHA-256");
	//Passing data to the created MessageDigest Object
	md.update(message.getBytes());
	//Compute the message digest
	byte[] digest = md.digest();
	System.out.println(digest);
	//Converting the byte array into HexString format
	StringBuffer hexString = new StringBuffer();
	for (int i=0; i<digest.length; i++){
		hexString.append(Integer.toHexString(0xFF & digest[i]));
	}
	System.out.println("Hex format: " + hexString.toString());
}
}
```

Αποτέλεσμα

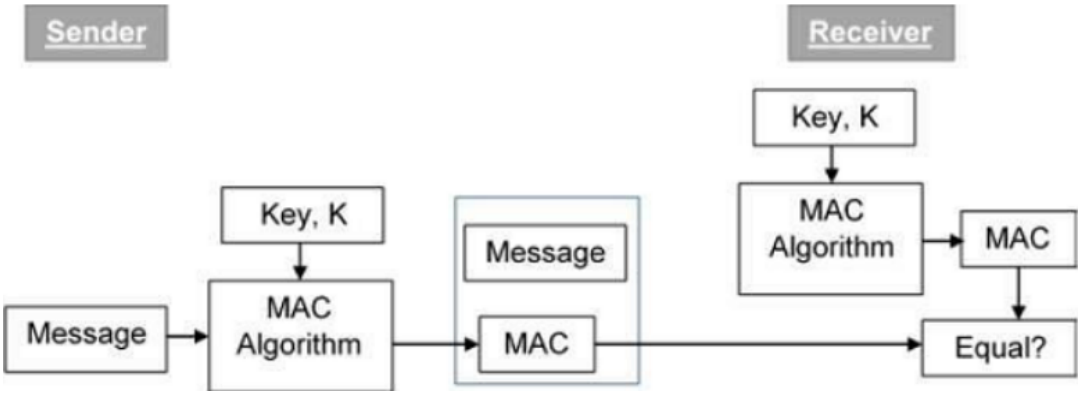
```
hello everyone

[B@55f96302
Hex format : e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

52

3.3.2.5 Message Authentication Code (MAC)

Ένας Message Authentication Code (MAC) είναι ένα κρυπτογραφικό άθροισμα δεδομένων που χρησιμοποιεί ένα session κλειδί λειτουργίας για τον εντοπισμό τόσο τυχαίων όσο και εκούσιων τροποποιήσεων των δεδομένων.

Ένα MAC απαιτεί δύο εισόδους: ένα μήνυμα και ένα μυστικό κλειδί που είναι γνωστά μόνο στον εντολέα του μηνύματος και στους παραλήπτες που προορίζονται. Αυτό επιτρέπει στον παραλήπτη του μηνύματος να επαληθεύσει την ακεραιότητα του μηνύματος και να επιβεβαιώσει ότι ο αποστολέας του μηνύματος έχει το κοινό μυστικό κλειδί. Εάν ένας αποστολέας δεν γνωρίζει το μυστικό κλειδί, τότε η τιμή κατακερματισμού θα ήταν διαφορετική, κάτι που δείχνει στον παραλήπτη ότι το μήνυμα δεν προέρχεται από τον αρχικό αποστολέα.



Στην Java η κλάση MAC βρίσκεται στο πακέτο javax.crypto. Για την δημιουργία ενός message of authentication code ακολουθούνται τα επόμενα βήματα:

Βήμα 1: Δημιουργία keyGenerate αντικειμένου:
Στην κλασική KeyGenerator υπάρχει η μέθοδος getInstance(), η οποία παίρνει ως όρισμα μία συμβολοσειρά που αναπαριστά τον αλγόριθμο που θα χρησιμοποιηθεί για την παραγωγή κλειδιών και επιστρέφει ένα αντικείμενο KeyGenerator που παράγει τα μυστικά κλειδιά.

```
//Creating a KeyGenerator object
KeyGenerator keyGen = KeyGenerator.getInstance("DES");
```

Βήμα 2: Δημιουργία SecureRandom αντικειμένου:
Η SecureRandom κλάση του πακέτου java.Security προσδιορίζει έναν δυνατό random number generator για την παραγωγή τυχαίων αριθμών στην Java. Η χρήση αυτής της κλάσης γίνεται με τον ακόλουθο τρόπο:

```
//Creating a SecureRandom object
SecureRandom secRandom = new SecureRandom();
```

[52] "Java Cryptography - Message Digest - Tutorialspoint." https://www.tutorialspoint.com/java_cryptography/java_cryptography_message_digest.htm. Πρόσβαση στις 12 Οκτ. 2020.

Βήμα 3: Αρχικοποίηση του KeyGenerator:
Η κλάση KeyGenerator διαθέτει μία μέθοδο init() η οποία παίρνει ως όρισμα ένα αντικείμενο SecureRandom και αρχικοποιεί τον τρέχοντα KeyGenerator.

```
//Initializing the KeyGenerator
keyGen.init(secRandom);
```

Βήμα 4: GenerateKey:
Η μέθοδος generateKey() χρησιμοποιείται για την παραγωγή κλειδιού όπως φαίνεται παρακάτω:

```
//Creating/Generating Key
Key key = keyGen.generateKey();
```

Βήμα 5: Αρχικοποίηση του MAC Object:
Η μέθοδος init() της κλάσης Mac παίρνει ως όρισμα ένα Key Object και αρχικοποιεί το τρέχον MAC Object χρησιμοποιώντας το δοσμένο κλειδί.

```
//Initializing the Mac object
mac.init(key);
```

Βήμα 6: Ολοκλήρωση της διαδικασίας Mac:
Η μέθοδος doFinal() της κλάσης Mac χρησιμοποιείται για την ολοκλήρωση της διαδικασίας το περασμα των απαιτούμενων δεδομένων σε αυτήν τη μέθοδο γίνεται ως εξής:

```
//Computing the Mac
String msg = new  String("Hello everyone");
byte[] bytes = msg.getBytes();
byte[] macResult = mac.doFinal(bytes);
```

Παράδειγμα

```
import java. security.Key;
import java. security.SecureRandom;
import javax.crypto.KeyGenerator;
import javax.crypto.Mac;
public class MessageAuthenticationCode {
   public static void main(String args[]) throws Exception{
        //Creating a KeyGenerator object
        KeyGenerator keyGen = KeyGenerator.getInstance("DES");
        //Creating a SecureRandom object
        SecureRandom secRandom = new SecureRandom();
        //Initializing the KeyGenerator
        keyGen.init(secRandom);
        //Creating Generating a key
        Key key = keyGen.generateKey();
        //Creating a Mac object
        Mac mac = Mac.getInstance("HmacSHA256");
        //Initializing the Mac object
        mac.init(key);
        //Computing the Mac
```

```
        String msg = new String("valid");
        byte[] bytes = msg.getBytes();
        byte[] macResult = mac.doFinal(bytes);
        System.out.println("Mac result:");
        System.out.println(new String(macResult));
        }
    }
```

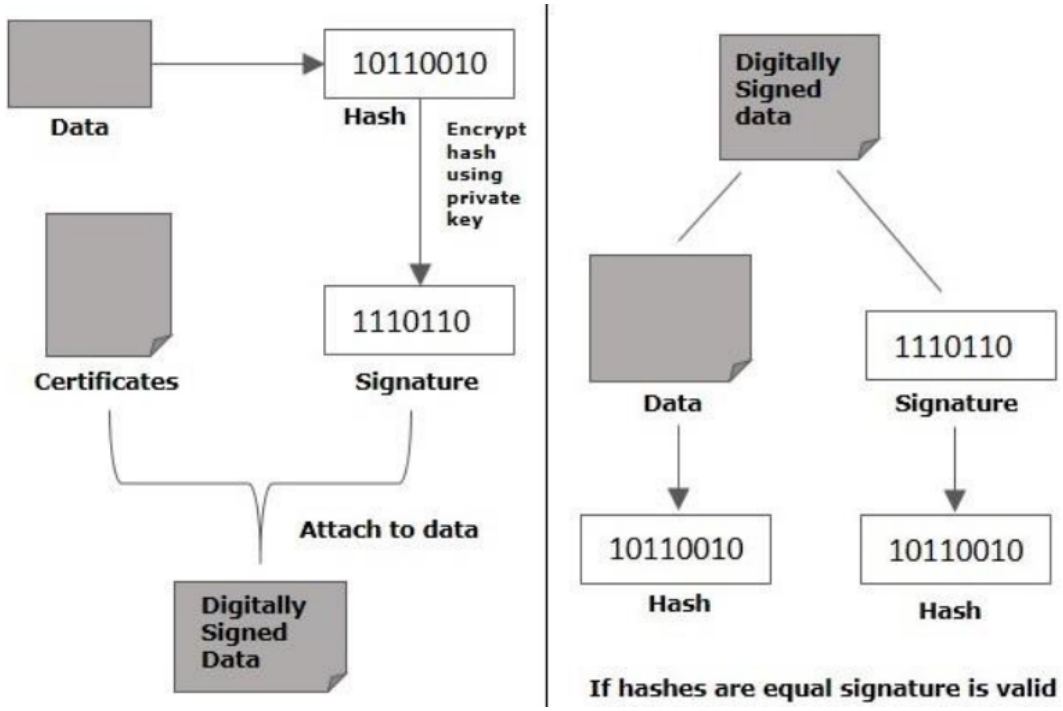
Αποτέλεσμα

```
Mac result:
LØýøÝ□c□á¼ÅŽÜËê\æe□û+˜fC?(±`□äQ
```

[53]

3.3.2.6 Ψηφιακές Υπογραφές

Οι ψηφιακές υπογραφές επιτρέπουν την επαλήθευση του συντάκτη, της ημερομηνίας και της ώρας των υπογραφών, καθώς και την επικύρωση του περιεχομένου του μηνύματος. Περιλαμβάνει επίσης λειτουργία ελέγχου ταυτότητας για επιπλέον δυνατότητες.



Για την ψηφιακή υπογραφή δεδομένα η Java χρησιμοποιεί την κλάση java.security.Signature. Η υπογραφή διαχωρίζεται από τα δεδομένα. Παράγεται δημιουργώντας ένα message digest(hash) από τα δεδομένα και κρυπτογραφώντας το message digest με το ιδιωτικό κλειδί του αποστολέα. Το κρυπτογραφημένο message digest είναι γνωστό ως ψηφιακή υπογραφή. Οι ψηφιακές υπογραφές βοηθούν στην αυθεντικοποίηση των πηγών των μηνυμάτων.

Δημιουργία, υπολογισμός και επαλήθευση υπογραφής
Για τη δημιουργία μιας ψηφιακής υπογραφής ακολουθείται η παρακάτω διαδικασία:

[53] "Java Cryptography - Creating a MAC - Tutorialspoint." https://www.tutorialspoint.com/java_cryptography/java_cryptography_creating_mac.htm. Πρόσβαση στις 13 Οκτ. 2020.

Βήμα 1: Δημιουργία αντικειμένου KeyPairGenerator
Η KeyPairGenerator κλάση διαθέτει τη μέθοδο getInstance() η οποία παίρνει ως όρισμα μία συμβολοσειρά η οποία χρησιμοποιείται για τον ορισμό του αλγορίθμου παραγωγής του κλειδιού και επιστρέφει ένα αντικείμενο KeyPairGenerator που παράγει τα κλειδιά.

```
//Creating KeyPair generator object
KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("DSA");
```

Βήμα 2: Αρχικοποίηση του αντικειμένου KeyPairGenerator
Η μέθοδος initialize() χρησιμοποιείται για την αρχικοποίηση Key Pair Generator. ο τρόπος χρήσης της φαίνεται παρακάτω:

```
//Initializing the KeyPairGenerator
keyPairGen.initialize(2048);
```

Βήμα 3: Παραγωγή του KeyPairGenerator
Η δημιουργία του KeyPair γίνεται χρησιμοποιώντας τη μέθοδο createKeyPair().

```
//Generate the pair of keys
KeyPair pair = keyPairGen.generateKeyPair();
```

Βήμα 4: Λήψη ιδιωτικού κλειδιού από τα κλειδιά
Για να πάρει ο προγραμματιστής το ιδιωτικό κλειδί από το αντικείμενο KeyPair που παράχθηκε, χρησιμοποιείται η μέθοδος getPrivate().

```
//Getting the private key from the key pair
PrivateKey privKey = pair.getPrivate();
```

Βήμα 5: Δημιουργία αντικειμένου υπογραφής
Η μέθοδος getInstance() της κλάσης Signature δέχεται μία παράμετρο συμβολοσειράς ορίζει τον απαιτούμενο αλγόριθμο υπογραφής και επιστρέφει το σχετικό αντικείμενο Signature.Η δημιουργία ενός αντικειμένου της κλάσης Signature με χρήση της μεθόδου getInstance() φαίνεται παρακάτω:

```
//Creating a Signature object
Signature sign = Signature.getInstance("SHA256withDSA");
```

Βήμα 6: Αρχικοποίηση του αντικειμένου Signature
Η μέθοδος initSign() της κλάσης Signature δέχεται ένα αντικείμενο PrivateKey και αρχικοποιεί το τρέχον αντικείμενο Signature.

```
//Initialize the signature
sign.initSign(privKey);
```

Βήμα 7: Προσθήκη δεδομένων στο αντικείμενο Signature
Η μέθοδος update() της κλάσης Signature δέχεται έναν πίνακα από bytes που αναπαριστά τα δεδομένα που θα υπογραφούν ή επαληθευτούν και ενημερώνει το τρέχον αντικείμενο με τα δοσμένα δεδομένα.

```
byte[] bytes = "Hello how are you".getBytes();

//Adding data to the signature
sign.update(bytes);
```

Βήμα 8: Υπολογισμός της υπογραφής
Η μέθοδος sign() επιστρέφει τα bytes της υπογραφής από τα ενημερωμένα δεδομένα.

```
//Calculating the signature
byte[] signature = sign.sign();
```

Βήμα 9: Επαλήθευση μιας υπογραφής
Για να επαναληφθεί μία υπογραφή πρέπει να αρχικοποιηθεί το Signature instance σε verification mode, το οποίο γίνεται με την μέθοδο initVerify() και με παράμετρο το δημόσιο κλειδί.

```
Signature signature = Signature.getInstance("SHA256WithDSA");
signature.initVerify(keyPair.getPublic())
```

Έπειτα καλείται η μέθοδος update() με τα δεδομένα που υπογράφει και η διαδικασία τελειώνει καλώντας την verify() που επιστρέφει true ή false ανάλογα αν η υπογραφή μπόρεσε να επαληθευτεί ή όχι.

```
byte[] data2 = "hello everyone".getBytes("UTF-8");
signature2.update(data2);
boolean verified = signature2.verify(digitalSignature);
```

Παράδειγμα - Δημιουργία υπογραφής
Το παρακάτω Java πρόγραμμα δέχεται ένα μήνυμα από τον χρήστη και παράγει μία ψηφιακή υπογραφή για το δοσμένο μήνυμα.

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.Signature;
import java.util.Scanner;

public class CreatingDigitalSignature {
   public static void main(String args[]) throws Exception {
      //Accepting text from user
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter some text");
      String msg = sc.nextLine();

      //Creating KeyPair generator object
      KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("DSA");

      //Initializing the key pair generator
      keyPairGen.initialize(2048);

      //Generate the pair of keys
      KeyPair pair = keyPairGen.generateKeyPair();

      //Getting the private key from the key pair
```

```
      PrivateKey privKey = pair.getPrivate();

      //Creating a Signature object
      Signature sign = Signature.getInstance("SHA256withDSA");

      //Initialize the signature
      sign.initSign(privKey);
      byte[] bytes = "msg".getBytes();

      //Adding data to the signature
      sign.update(bytes);

      //Calculating the signature
      byte[] signature = sign.sign();

      //Printing the signature
      System.out.println("Digital signature for given text: "+new String(signature, "UTF8"));
   }
}
```

Έξοδος

Το παραπάνω πρόγραμμα παράγει την ακόλουθη έξω:

```
Enter the message
Hello everyone
Digital signature for given text: 0=□=?u?□????._??i? -d□le???□?□ ?n?□?9??f?j??`?□&q???Sm}?
/
```

[54]

Παράδειγμα - Επαλήθευση υπογραφής

Το παρακάτω πρόγραμμα επαληθεύει μία υπογραφή

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.Signature;
import java.util.Scanner;
 public class SignatureVerificationExample {
  public static void main(String args[]) throws Exception{
   //Creating KeyPair generator object
   KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("DSA");
   //Initializing the key pair generator
   keyPairGen.initialize(2048);
   //Generate the pair of keys
   KeyPair pair = keyPairGen.generateKeyPair();
   //Getting the privatekey from the key pair
   PrivateKey privKey = pair.getPrivate();
   //Creating a Signature object
   Signature sign = Signature.getInstance("SHA256withDSA");
   //initializing the signature
   sign.initSign(privKey);
```

[54] "Java Cryptography - Creating Signature - Tutorialspoint." https://www.tutorialspoint.com/java_cryptography/java_cryptography_creating_signature.htm. Πρόσβαση στις 13 Οκτ. 2020.

```
   byte[] bytes = "Hello everyone".getBytes();
   //Adding data to the signature
   sign.update (bytes) ;
   // Calculating the signature
   byte[] signature = sign.sign();
   // Initializing the signature
   sign.initVerify(pair.getPublic());
   sign.update (bytes) ;
   // Verifying the signature
   boolean bool = sign.verify(signature);
   if(bool) {
     System.out.println("Signature verified");
   } else {
     System.out.println("Signature failed");
   }
}
```

Αποτέλεσμα

```
Signature verified
```

[55]

3.3.2.7 Διαχείριση κλειδιών

Κλειδί και πιστοποιητικό:

Η κλάση της Java για τη διαχείριση μιας ποικιλίας πιστοποιητικών ταυτότητας είναι η java.security.cert.certificate. Ένα πιστοποιητικό ταυτότητας είναι η δέσμευση ενός εντολέα σε ένα δημόσιο κλειδί το οποίο επιβεβαιώνεται από άλλο principal (π.χ. CA).Η κλάση αυτή έχει μία υποκλάση, την X509certificate κλάση. Έτσι αυτές οι δύο κλάσεις χρησιμοποιούνται σαν ένα πιστοποιητικό αναγνώρισης στο HTTPS και TLS.

Τα κλειδιά και τα πιστοποιητικά συχνά συνδέονται με ένα άτομο η μία οργάνωση. Επιβάλλεται να υπάρχει ένας σαφής ορισμός πώς αυτά θα μεταδίδονται θα μοιράζονται και θα αποθηκεύονται. Τα κλειδιά είναι ένα σημαντικό μέρος του κρυπτογραφικού αλγορίθμου της Java επειδή επιτρέπουν την επαλήθευση και παραγωγή ψηφιακών υπογράφων ή την διεξαγωγή κρυπτογράφησης.Οι ψηφιακές υπογραφές δημιουργούνται με ένα ιδιωτικό κλειδί και μετά μεταφέρονται ηλεκτρονικά μαζί με τα δεδομένα που υπογράφηκαν. Όταν η ψηφιακή υπογραφή ληφθεί πρέπει να επαληθευθεί, το οποίο προϋποθέτει ότι το δημόσιο κλειδί αντιστοιχεί στο ιδιωτικό κλειδί με το οποίο έγινε παραγωγή της υπογραφής. Το βασικό αντικείμενο του συστήματος διαχείρισης κλειδιού είναι όταν χρειάζεται κάποιος να υπογράψει ψηφιακά η διαχείριση κλειδιού πρέπει να δημιουργεί ή να δίνει το ιδιωτικό κλειδί του χρήστη ώστε να του δίνει τη δυνατότητα να υπογράψει. Επίσης όταν χρειάζεται να επαληθεύσει μία ψηφιακή υπογραφή το σύστημα διαχείρισης κλειδιού θα πρέπει να δίνει το κατάλληλο δημόσιο κλειδί για την επαλήθευση.

Υπάρχουν τρία στοιχεία ενός συστήματος διαχείρισης κλειδιού

- ❏ Κλειδιά: χρησιμοποιούνται για την υπογραφή δεδομένων. Όπως για παράδειγμα αρχεία JAR, συνήθως δίνονται από μία οντότητα (ένα άτομο ή έναν οργανισμό) και περιλαμβάνουν το ιδιωτικό κλειδί, το δημόσιο κλειδί ή και τα δύο κλειδιά.
- ❏ Πιστοποιητικά: χρησιμοποιούνται για την επαλήθευση της ψηφιακής υπογραφής.
- ❏ Ταυτότητες: χρησιμοποιούνται για την διαχείριση ταυτοτήτων με τα κλειδιά τους που αποθηκεύονται στη βάση δεδομένων των κλειδιών (keystore).

KeyStore:

[55] "Java Cryptography - Verifying Signature - Tutorialspoint." https://www.tutorialspoint.com/java_cryptography/java_cryptography_verifying_signature.htm. Πρόσβαση στις 13 Οκτ. 2020.

Το KeyStore είναι μία σημαντική κλαση στο JCA που διαχειρίζεται το σύστημα διαχείρισης κλειδιών της Java (java.security.KeyStore) και διαθέτει τυποποιημένους μηχανισμούς για τη διαχείριση και αποθήκευση κρυπτογραφικών κλειδιών και πιστοποιητικών (password-protected database).

Οι καταχωρήσεις στο KeyStore γίνονται με χρήση της διεπαφής keyStore.Entry, για παράδειγμα keyStore.PrivateKeyEntry, keyStore.SecretKeyEntry και keystore.TrustedCertificateEntry.

Βήμα 1: Δημιουργία KeyStore

Για την υλοποίηση του KeyStore χρησιμοποιείται η μέθοδος getInstance().

```
KeyStore keyStore = KeyStore.getInstance(KeyStore.getDefaultType());
```

Ή διαφορετικά

```
KeyStore keyStore = KeyStore.getInstance(“PKCS12”);
```

Βήμα 2: Φόρτωση KeyStore

Πριν χρησιμοποιηθεί ένα keystore πρέπει να φορτωθεί. Τα KeyStores είναι συνήθως γραμμένα στο δίσκο ή σε κάποιο άλλο είδος αποθήκης για μετέπειτα χρήση. Η κλάση KeyStore υποθέτει ότι πρέπει να διαβαστούν τα δεδομένα της προτού χρησιμοποιηθεί. Όμως είναι πιθανό να γίνει αρχικοποίηση ενός κενού KeyStore δηλαδή χωρίς καθόλου δεδομένα.

Η μέθοδος KeyStore load() χρησιμοποιείται για την φόρτωση των δεδομένων του KeyStore από ένα αρχείο ή κάποιο άλλο χώρο αποθήκευσης. Η μέθοδος load() παίρνει δύο παραμέτρους:
1. Ένα InputStream από το οποίο φορτώνονται τα δεδομένα.
2. Ένας πίνακας χαρακτήρων που περιέχουν τον κωδικό του KeyStore.

```
char[] keyStorePassword = "6789xyz".toCharArray();
try(InputStream keyStoreData = new FileInputStream("keystore.ks")){
   keyStore.load(keyStoreData, keyStorePassword);
}
```

Το παραπάνω παράδειγμα φορτώνει το αρχείο keystore.ks. Εάν δεν επιθυμεί ο προγραμματιστής να φορτωθούν δεδομένα στο KeyStore, απλά πρέπει να περαστεί η τιμή null στην παράμετρο InputStream.

```
keyStore3.load(null, keyStorePassword);
```

Βήμα 3: Λήψη Κλειδιών
Για την λήψη κλειδιών από το KeyStore χρησιμοποιείται η μέθοδος getEntry(). Μία keystore καταχώρηση αντιστοιχείται σε ένα ψευδώνυμο που αναγνωρίζει το κλειδί και προστατεύεται με έναν κωδικό. Συνεπώς για να υπάρχει πρόσβαση σε ένα κλειδί πρέπει να περαστεί το ψευδώνυμο του κλειδιού και ο κωδικός στη μέθοδο getEntry().

Για παράδειγμα:

```
char[] keyPassword = "789xyz".toCharArray()
KeyStore.ProtectionParameter entryPassword = new    KeyStore.PasswordProtection(keyPassword);
KeyStore.Entry keyEntry = keyStore3.getEntry("keyAlias", entryPassword);
```

Βήμα 4: Ρύθμιση Κλειδιών

Ο χρήστης μπορεί επίσης να θέσει τα κλειδιά σε ένα KeyStore instance.

Για παράδειγμα:

```
SecretKey secretKey = getSecretKey();
KeyStore.SecretKeyEntry secretKeyEntry = new KeyStore.SecretKeyEntry(secretKey);
keyStore3.setEntry("keyAlias2", secretKeyEntry, entryPassword);
```

Βήμα 5: Αποθήκευση του KeyStore

Για την αποθήκευση ενός keyStore χρησιμοποιείται η μέθοδος store().

```
char[] keyStorePassword = "678xyz".toCharArray();
try(FileOutputStream keyStoreOutputStream = new FileOutputStream("data/keystore.ks")){
       keyStore3.store(keyStoreOutputSteram, keyStorePassword);
}
```

Keytool:

Τέλος το keytool εργαλείο χρησιμοποιείται μέσω της γραμμής εντολών, με το οποίο μπορεί να δημιουργήσει ο χρήστης ζεύγη δημόσιου / ιδιωτικού κλειδιού και να τα αποθηκεύσει. Διανείμετε με το Java SDK ή το JRE και για να εκτελεστεί πρέπει να ανοίξει γραμμή εντολών και να γίνει αλλαγή directory στο bin directory της εγκαταστάσεις του Java SDK.

3.3.2.8 Java Secure Socket Extension (JSSE)

Το JSSE εφαρμόζει την ασφαλή επικοινωνία στο ίντερνετ. Είναι ένα framework σχεδιασμένο για ασφαλή επικοινωνία σε μη αξιόπιστα δίκτυα. Το JSSE API υποστηρίζει τα παρακάτω πρωτόκολλα ασφαλείας:

SSL	v3.0
TLS	v1.0, v1.1 και v1.2
DTLS	v1.0 και v1.2

Η λειτουργία του JSSE περιλαμβάνει κρυπτογράφηση δεδομένων, αυθεντικοποίηση server, αυθεντικοποίηση client και ακεραιότητα μηνύματος. Χρησιμοποιώντας το JSSE οι προγραμματιστές μπορούν να προσφέρουν ένα ασφαλές μονοπάτι για τα δεδομένα μεταξύ ενός client και ενός server που τρέχει σε οποιοδήποτε πρωτόκολλο (όπως το HTTP, Telnet ή το FTP) πάνω από TCP/IP.

Παράδειγμα μη ασφαλούς κώδικα

```
import java.io.*;
import java.net.*;
...

int port = availablePortNumber;
ServerSocket s;
try {
s = new ServerSocket(port);
Socket c = s.accept();
OutputStream out = c.getOutputStream();
InputStream in = c.getInputStream();
// Send messages to the client through the OutputStream
```

```
// Receive messages from the client through the InputStream
} catch (IOException e) {
}
```

Παράδειγμα ασφαλούς κώδικα

```
import java.io.*;
import javax.net.ssl.*;
...
int port = availablePortNumber;
String host = "hostname";
try {
    SSLSocketFactory sslFact=(SSLSocketFactory)SSLSocketFactory.getDefault();
    SSLSocket s = (SSLSocket)sslFact.createSocket(host, port);
    OutputStream out = s.getOutputStream();
    InputStream in = s.getInputStream();
     // Send messages to the server through the OutputStream
     // Receive messages from the server through the InputStream
}
catch (IOException e) {
}
```

3.4 Διαχείριση Συνόδου

3.4.1 Εισαγωγή

Διαχείριση συνόδου είναι το σύνολο των κανόνων που διέπουν την αλληλεπίδρασης μεταξύ ενός χρήστη και μιας εφαρμογής web. Οι ιστοσελίδες και οι browsers χρησιμοποιούν το HTTP για να επικοινωνήσουν. Μία σύνοδος web είναι μία σειρά από HTTP requests και responses που δημιουργούνται από τον χρήστη. Επειδή το πρωτόκολλο HTTP είναι stateless τρέχει την κάθε εντολή ξεχωριστά χωρίς να γνωρίζει τις προηγούμενες εντολές.

Για την υλοποίηση διαχείρισης συνόδου πρέπει να υπάρχουν στην εφαρμογή συστήματα αυθεντικοποίησης και εξουσιοδότησης.
Υπάρχουν δύο τύποι διαχείρισης συνόδου:

- ❏ Cookie-based
- ❏ URL rewriting

Αυτοί οι τύποι διαχείριση συνόδου χρησιμοποιούνται είτε μαζί είτε ξεχωριστά. Η διαχείριση συνόδου χρησιμοποιείται από τον διαχειριστή για την παρακολούθηση της συχνότητας των επισκεπτών σε μία ιστοσελίδα και την κίνηση μέσα στα sites.

3.4.1.1 Παρακολούθηση Συνόδου

Η παρακολούθηση Συνόδου είναι ένας μηχανισμός που χρησιμοποιούν τα Java servlets για την διατήρηση της κατάστασης για μία σειρά από requests από τον ίδιο χρήστη για κάποιο χρονικό διάστημα.

Για να χρησιμοποιηθεί η παρακολούθηση συνόδου πρέπει:

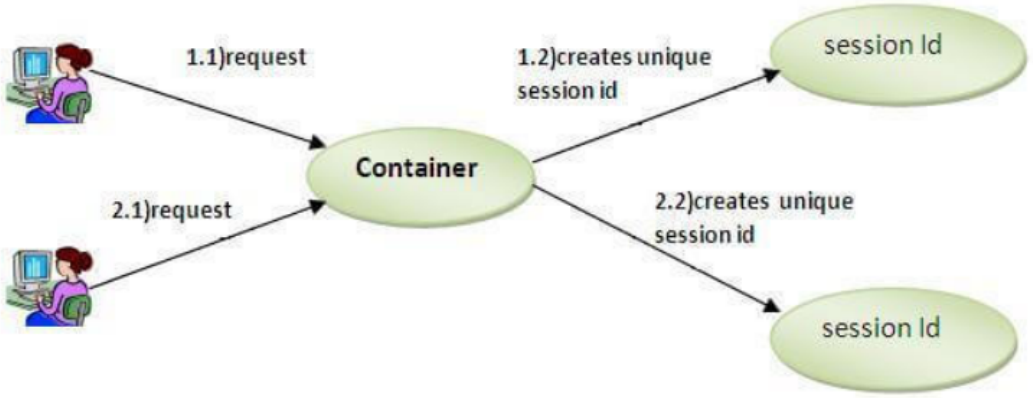
- ❏ Να εξασφαλιστεί μία σύνοδος (ένα HttpSession αντικείμενο) για έναν χρήστη.

- ❏ Η λήψη ή η αποθήκευση δεδομένων από ένα αντικείμενο HttpSession.
- ❏ Η ακύρωση της συνόδου, κάτι το οποίο είναι προαιρετικό.

3.4.1.1.1 Παρακολούθηση Συνόδου με τη μέθοδο HttpSession

Η μέθοδος αυτή είναι ένας τρόπος για να δοθεί ταυτότητα σε έναν χρήστη σε ένα ή περισσότερα requests. Επίσης βοηθάει στη διαχείριση και προβολή πληροφοριών σχετικά με μία σύνοδο όπως για παράδειγμα τον χρόνο δημιουργίας, το αναγνωριστικό συνόδου και την τελευταία φόρα πρόσβασης. Τέλος βοηθάει στην σύνδεση του αντικειμένου με τις συνόδους επιτρέποντας τις πληροφορίες του χρήστη να διατηρούνται σε πολλές συνδέσεις του χρήστη.

Όταν ένας χρήστης εισέρχεται σε έναν ιστότοπο (ή μια διαδικτυακή εφαρμογή) για πρώτη φορά το HttpSession λαμβάνεται μέσω του request.getSession() και ο χρήστης παίρνει ένα μοναδικό αναγνωριστικό για να προσδιορίσει το session του.



Το HttpSession παραμένει ενεργό έως ότου δεν έχει χρησιμοποιηθεί για περισσότερο από την τιμή χρονικού ορίου που καθορίζεται στο tag στο αρχείο περιγραφής ανάπτυξης (web.xml). Η προεπιλεγμένη τιμή χρονικού ορίου είναι 30 λεπτά και χρησιμοποιείται αν δεν καθοριστεί η τιμή στο tag. Αυτό σημαίνει ότι όταν ο χρήστης δεν επισκέπτεται την εφαρμογή στον καθορισμένο χρόνο εφαρμογής ιστού, το session του καταστρέφεται από servlet container.

Ο παρακάτω κώδικας δείχνει πώς δημιουργείται ένα αντικείμενο HttpSession.

```
protected void doPost(HttpServletRequest req,
    HttpServletResponse res)
    throws ServletException, IOException {
        HttpSession session = req.getSession();
}
```

Μπορεί να αποθηκευτεί η πληροφορία του χρήστη μέσα σε ένα αντικείμενο συνόδου χρησιμοποιώντας τη μέθοδο setAttribute() και αργότερα όταν χρειαστεί αυτές οι πληροφορίες μπορούν να ληφθούν από την σύνοδο. Με αυτόν τον τρόπο μπορούν να αποθηκευτούν πληροφορίες σε μία σύνοδο. Παρακάτω αποθηκεύεται ένα username, ένα emailid και το userage στη σύνοδο με τα attributes: name uName, uemailid και uAge αντίστοιχα.

```
session.setAttribute("uName", "ChaitanyaSingh");
session.setAttribute("uemailId", "myemailid@gmail.com");
session.setAttribute("uAge", "30");
```

Η πρώτη παράμετρος είναι το name του attribute και η δεύτερη η τιμή του attribute.
Για να ληφθεί η τιμή από την σύνοδο χρησιμοποιείται η μέθοδος getAttribute() της διεπαφής HttpSession. Παρακάτω γίνεται λήψη των τιμών του attribute με χρήση των attribute names.

```
String userName = (String) session.getAttribute("uName");
String userEmailId = (String) session.getAttribute("uemailId");
String userAge = (String) session.getAttribute("uAge");
```

Μέθοδοι του HttpSession

public void setAttribute(String name, Object value): Συνδέει το αντικείμενο με ένα όνομα και αποθηκεύει το ζεύγος ονόματος / τιμής ως χαρακτηριστικό του αντικειμένου HttpSession. Εάν υπάρχει ήδη ένα χαρακτηριστικό, τότε αυτή η μέθοδος αντικαθιστά τα υπάρχοντα χαρακτηριστικά.

public Object getAttribute(String name): Επιστρέφει το αντικείμενο συμβολοσειράς που καθορίζεται στην παράμετρο, από το session αντικείμενο. Εάν δεν βρεθεί κανένα αντικείμενο για το καθορισμένο χαρακτηριστικό, τότε η μέθοδος getAttribute() επιστρέφει null.

public Enumeration getAttributeNames(): Επιστρέφει μια απαρίθμηση που περιέχει το όνομα όλων των αντικειμένων που είναι δεσμευμένα ως χαρακτηριστικά στο αντικείμενο συνόδου.

public void removeAttribute(String name): Καταργεί το δοσμένο χαρακτηριστικό από το session.

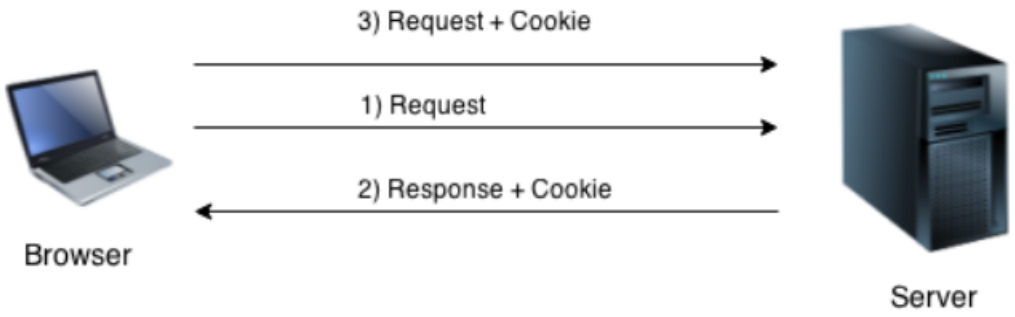
setMaxInactiveInterval(int interval): Ορίζει το χρόνο αδράνειας της περιόδου λειτουργίας σε δευτερόλεπτα. Αυτή είναι η ώρα σε δευτερόλεπτα που καθορίζει το χρονικό διάστημα που οι συνεδρίες παραμένουν ενεργές από το τελευταίο αίτημα που ελήφθη από τον client.

Για την πλήρη λίστα των μεθόδων, ο προγραμματιστής μπορεί να επισκεφθεί το official documentation.
[56]

3.4.1.1.2 Παρακολούθηση Συνόδου με τη μέθοδο των Cookies

Ένα cookie είναι ένα μικρό ποσό πληροφορίας πού αποθηκεύεται στο client-side και οι servers το χρησιμοποιούν όταν επικοινωνούν με τους clients.

Από προεπιλογή, κάθε αίτημα θεωρείται νέο αίτημα. Στην τεχνική των cookies, προσθέτουμε cookie με response από το servlet. Έτσι, το cookie αποθηκεύεται στην προσωρινή μνήμη του προγράμματος περιήγησης. Μετά από αυτό, εάν ένα αίτημα αποσταλεί από τον χρήστη, το cookie προστίθεται στο request από προεπιλογή. Έτσι, αναγνωρίζεται ο παλιός χρήστης.



Υπάρχουν 2 τύποι cookie σε servlets.

1. **Μη μόνιμο cookie**: Είναι έγκυρο για μία σύνοδο μόνο. Διαγράφεται κάθε φορά που ο χρήστης κλείνει τον browser.
2. **Μόνιμο cookie**: Είναι έγκυρο για πολλές συνοδούς. Δεν διαγράφεται κάθε φορά που ο χρήστης κλείνει τον browser. Διαγράφεται μόνο αν ο χρήστης κάνει logout.

[56] "HttpSession with example in Servlet - BeginnersBook.com." https://beginnersbook.com/2013/05/http-session/. Πρόσβαση στις 15 Οκτ. 2020.

Παράδειγμα χρήσης των cookies φαίνεται στο παρακάτω κώδικα όπου γίνεται η αποστολή δεδομένων στον client με δημιουργία και αποστολή ενός cookie.

```
Cookie ColorCookie = new Cookie("color","white");
response.addCookie(ColorCookie);
```

Αυτός ο κώδικας δημιούργησε ένα cookie με το όνομα “color” και την τιμή “white”.

Παρακάτω φαίνεται άλλο ένα παράδειγμα πού δείχνει τον τρόπο λήψης της τιμής για ένα συγκεκριμένο cookie.

```
Cookie[] cookies = request.getCookies();
String userId = null;
for(Cookie cookie : cookies){
	if("uid".equals(cookie.getName())){
			userId = cookie.getValue();
	}
}
```

Μπορεί να γίνει ένα iteration μέσα στον πίνακα των cookies και να βρεθεί το cookie που χρειάζεται. Δεν υπάρχει κάποια μέθοδος για την λήψη ενός cookie με κάποιο συγκεκριμένο όνομα.

Πλεονεκτήματα των Cookies

1. Η πιο απλή μέθοδος για διατήρηση κατάστασης.
2. Τα cookies διατηρούνται στο client-side.

Μειονεκτήματα των Cookies

1. Δεν λειτουργεί αν η λειτουργία των cookies είναι απενεργοποιημένη από τον browser.
2. Μόνο textual πληροφορία μπορεί να οριστεί σε ένα αντικείμενο cookie.[57]

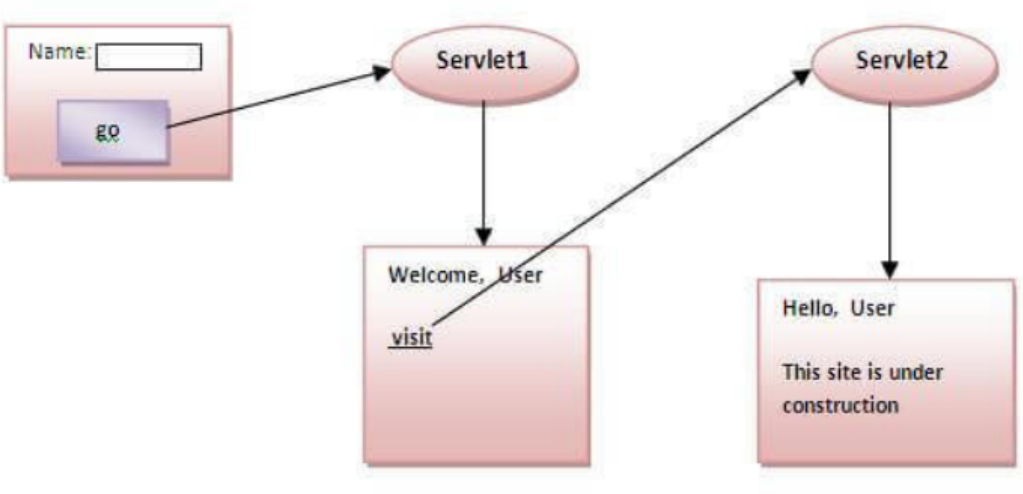
3.4.1.1.3 Παρακολούθηση Συνόδου με τη μέθοδο URL Rewriting

Κατά την επανεγγραφή διευθύνσεων URL, γίνεται προσθήκη ενός token ή ενός αναγνωριστικού στη διεύθυνση URL του επόμενου Servlet ή του επόμενου πόρου. Μπορούν να σταλούν ζεύγη παραμέτρων ονόματος / τιμής χρησιμοποιώντας την ακόλουθη μορφή:

url?name1=value1&name2=value2&??

Ένα όνομα και μια τιμή διαχωρίζονται χρησιμοποιώντας το σύμβολο ίσο =, ένα ζεύγος παραμέτρων ονόματος / τιμής διαχωρίζεται από μια άλλη παράμετρο χρησιμοποιώντας το σύμβολο ampersand (&). Όταν ο χρήστης κάνει κλικ στην υπερσύνδεση, τα ζεύγη ονόματος / τιμής παραμέτρου θα μεταβιβαστούν στον διακομιστή. Από ένα Servlet, μπορούμε να χρησιμοποιήσουμε τη μέθοδο getParameter() για να λάβουμε μια τιμή παραμέτρου.

[57] "Cookies in Servlet - javatpoint." https://www.javatpoint.com/cookies-in-servlet. Πρόσβαση στις 15 Οκτ. 2020.



Πλεονεκτήματα του URL Rewriting

1. Θα λειτουργεί πάντα είτε τη ρύθμιση για τα cookies είναι απενεργοποιημένη είτε όχι.
2. Επιπλέον καταχώρηση φόρμας δεν απαιτείται σε κάθε σελίδα.

Μειονεκτήματα του URL Rewriting

1. Δουλεύει μόνο με links.
2. Μπορεί να στείλει μόνο textual πληροφορίες.

Παράδειγμα

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet{
        public void doGet(HttpServletRequest request, HttpServletResponse response){
        try{
                response.setContentType("text/htm;");
                PrintWriter out = response.getWriter();
                String n = request.getParameter("userName");
                out.print("Welcome"+n);
                //appending the username in the query sting
                out.print("<a href='servlet2?uname="n+n+"'>visit</a>");
                out.close();
        }catch(Exception e{System.out.println(e);}
}
}
```

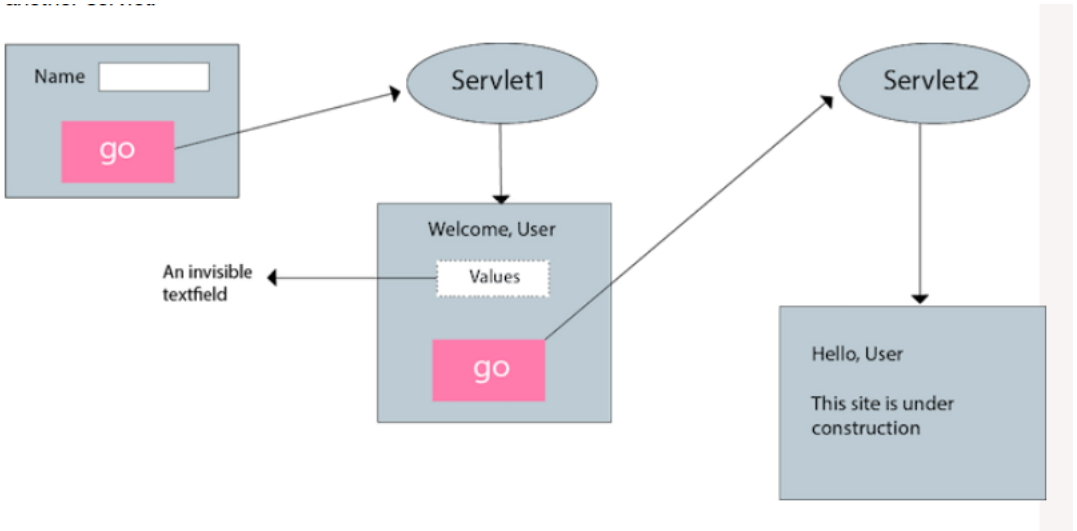
[58]

3.4.1.1.4 Παρακολούθηση Συνόδου με τη μέθοδο Hidden Fields

Στην περίπτωση των Hidden Fields ένα κρυφό (αόρατο) πεδίο κειμένου χρησιμοποιείται για τη διατήρηση της κατάστασης ενός χρήστη.

Σε αυτήν την περίπτωση, αποθηκεύονται οι πληροφορίες στο κρυφό πεδίο και λαμβάνονται από άλλο servlet. Αυτή η προσέγγιση είναι καλύτερη αν πρέπει να υποβληθεί η φόρμα σε όλες τις σελίδες και δεν είναι θεμιτή η εξάρτηση από το πρόγραμμα περιήγησης.

[58] "URL Rewriting in Servlet - javatpoint." https://www.javatpoint.com/url-rewriting-in-session-tracking. Πρόσβαση στις 15 Οκτ. 2020.



Ο κώδικας για την αποθήκευση τιμής σε αόρατο πεδίο είναι:

```
<input type="hidden" name="uname" value="Vimal Jaiswal">
```

Παράδειγμα χρήσης

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http. ";
public class FirstServlet extends HttpServlet{
 public void doGet(HttpServletRequest request, HttpServletResponse response)
try{
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String n=request.getParameter("userName");
    out.print("Welcome "+n);
    //creating form that has invisible textfield
    out.print("<form action="servlet2'>");
    out.print("<input type='hidden' name='uname'    value='"+n+"'>");
    out.print("<input type='submit' value='go'>");
    out.print("</form>");
    out.close();
catch(Exception e){System.out.println(e);}
}
}
```

Πλεονεκτήματα του Hidden Fields

1. Θα λειτουργεί πάντοτε είτε είναι ενεργοποιημένη η δυνατότητα των cookies είτε όχι.

Μειονεκτήματα του Hidden Fields

1. Διατηρείται στο server side.
2. Απαιτείται επιπλέον καταχώρηση φόρμας σε κάθε σελίδα.
3. Μπορούν να χρησιμοποιηθούν μόνο textual πληροφορίες.[59]

[59] "Hidden Form Field in Servlet - javatpoint." https://www.javatpoint.com/hidden-form-field-in-session-tracking. Πρόσβαση στις 15 Οκτ. 2020.

3.4.2 Διαχείριση Συνόδου με το Spring Framework

Υπάρχουν πολλοί τρόποι για να εκμεταλλευτεί κανείς και να χρησιμοποιήσει ένα HTTP Session σε μια εφαρμογή ιστού με βάση το Spring Framework. Παρακάτω παρατίθεται μία σύντομη περιγραφή χρήσης του.

3.4.2.1 Προσεγγίσεις

3.4.2.1.1 Προσέγγιση 1

Απλή ενσωμάτωση του HttpSession όπου απαιτείται:

```
@Service
public class ShoppingCartService {

 @Autowired
 private HttpSession httpSession;

 ...
}
```

Δεδομένου ότι η παραπάνω υπηρεσία είναι απλή, αυτό λειτουργεί καλά. Το Spring εισάγει έξυπνα έναν proxy στην HttpSession και αυτός ο proxy ξέρει πώς να αναθέσει εσωτερικά στη σωστή session το αίτημα.

Το catch, του να χειρίζεται κάποιος το session με αυτόν τον τρόπο όμως είναι ότι το αντικείμενο που ανακτάται και αποθηκεύεται πίσω στη συνεδρία θα πρέπει να το διαχειριστεί ο χρήστης:

```
public void removeFromCart(long productId) {
 ShoppingCart shoppingCart = getShoppingCartInSession();
 shoppingCart.removeItemFromCart(productId);
 updateCartInSession(shoppingCart);
}
```

3.4.2.1.2 Προσέγγιση 2

Αποδοχή της συνόδου σαν μία παράμετρο, κάτι το οποίο λειτουργεί μόνο στο επίπεδο web:

```
@Controller
public class ShoppingCartController {

  @RequestMapping("/addToCart")
  public String addToCart(long productId, HttpSession httpSession) {
    //do something with the httpSession
  }

}
```

3.4.2.1.3 Προσέγγιση 3

Σε αυτή την προσέγγιση δημιουργείται ένα Bean με session scope.

```
@Component
@Scope(proxyMode=ScopedProxyMode.TARGET_CLASS, value="session")
public class ShoppingCart implements Serializable{
...
}
```

Το Spring δημιουργεί έναν proxy για ένα bean session και καθιστά τον proxy διαθέσιμο σε υπηρεσίες που εισάγουν αυτό το bean. Ένα πλεονέκτημα της χρήσης αυτής της προσέγγισης είναι ότι οι αλλαγές κατάστασης σε αυτό το bean αντιμετωπίζονται από το Spring. Θα φροντίσει να ανακτήσει αυτό το bean από το session και να διαδώσει τυχόν αλλαγές στο bean πίσω στη συνεδρία. Περαιτέρω, εάν το bean είχε Spring lifecycle μεθόδους(ας πούμε @PostConstruct ή @PreDestroy annotated μεθόδους), θα καλούνταν κατάλληλα.

3.4.2.1.4 Προσέγγιση 4

Annotating τα στοιχεία μοντέλου του MVC Spring με το @SessionAttribute annotation.

```
@SessionAttributes("shoppingCart")
public class OrderFlowController {
 public String step1(@ModelAttribute("shoppingCart") ShoppingCart shoppingCart) {
 }
 public String step2(@ModelAttribute("shoppingCart") ShoppingCart shoppingCart) {
 }
 public String step3(@ModelAttribute("shoppingCart") ShoppingCart shoppingCart, SessionStatus status) {
  status.setComplete();
 }
}
```

Το annotation είναι ένα συγκεκριμένο μοντέλο, για να διατηρηθεί ένα state κατά τη διάρκεια του HttpSession πριν από την απόδοση της προβολής.[60]

3.4.2.2 Διαχείριση Συνόδου με το Spring Security

Παρακάτω θα παρουσιαστεί πώς το Spring Security επιτρέπει στον χρήστη να ελέγξει τα HTTP Sessions.

Με το Spring Security μπορεί να υπάρχει έλεγχος ως προς το πότε θα δημιουργείται ένα session και πώς το Spring Security θα αλληλεπιδράσει με αυτό:

- ❏ **always** - ένα session θα δημιουργείται συνέχεια, αν δεν υπάρχει ήδη ένα.
- ❏ **ifRequired -** ένα session θα δημιουργείται μόνο όταν απαιτείται (default)
- ❏ **never -** το framework δεν θα δημιουργεί ποτέ ένα session αλλά θα χρησιμοποιεί ένα αν υπάρχει.
- ❏ **stateless** - κανένα session δεν θα δημιουργείται ή θα χρησιμοποιείται από το Session Security

```
<http create-session="ifRequired">...</http>
```

[60] "Using Http Session With Spring Based Web ... - DZone." 29 Απρ. 2014, https://dzone.com/articles/using-http-session-spring. Πρόσβαση στις 17 Οκτ. 2020.

Java Configuration

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.sessionManagement()               .sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED)
}
```

Είναι πολύ σημαντικό να αντιληφθεί ότι αυτό το configuration ελέγχει μόνο ότι κάνει το String Security και όχι ολόκληρη την εφαρμογή. Το Spring Security μπορεί να μην δημιουργήσει το session, αν του δοθεί η οδηγία να μην το κάνει, αλλά μπορεί να το κάνει η εφαρμογή. Από προεπιλογή το Spring Security θα δημιουργήσει ένα session όταν χρειάζεται - με τη ρύθμιση “ifRequired”.

Για μια πιο stateless εφαρμογή, η επιλογή “never” θα διασφαλίσει ότι το Spring Security δεν θα δημιουργήσει κανένα session. Ωστόσο, εάν η εφαρμογή δημιουργήσει ένα session, τότε το Spring Security θα το χρησιμοποιήσει.

Τέλος, η αυστηρότερη επιλογή δημιουργίας session - “stateless” - αποτελεί εγγύηση ότι η εφαρμογή δεν θα δημιουργήσει καθόλου session.

3.4.2.2.1 Έλεγχος Ταυτόχρονης Συνεδρίας

Όταν ένας χρήστης που είναι ήδη αυθεντικοποιημένος προσπαθεί να αυθεντικοποιηθεί ξανά, η εφαρμογή μπορεί να αντιμετωπίσει το γεγονός με διαφορετικούς τρόπους. Μπορεί είτε να ακυρώσει το ενεργό session του χρήστη και να αυθεντικοποιήσει τον χρήστη ξανά με ένα νέο session, ή να επιτρέψει στα sessions να υπάρχουν ταυτόχρονα.

Το πρώτο βήμα για την ενεργοποίηση της υποστήριξης ταυτόχρονου ελέγχου session είναι η προσθήκη ενός listener στο web.xml.

```
<listener>
   <listener-class>
     org.springframework.security.web.session.HttpSessionEventPublisher
   </listener-class>
</listener>
```

Ή να οριστεί σαν Bean:

```
@Bean
public HttpSessionEventPublisher httpSessionEventPublisher() {
           return new HttpSessionEventPublisher();
}
```

Αυτό το βήμα είναι απαραίτητο για να είναι βέβαιο ότι το Spring Security session registry ειδοποιείται όταν το session καταστρέφεται.

Για να ενεργοποιηθεί το σενάριο που επιτρέπει πολλαπλά ταυτόχρονα sessions για τον ίδιο χρήστη πρέπει να χρησιμοποιηθεί το <session-management> στοιχείο στο XML Configuration:

```
<http ...>
           <session-management>
               <concurrency-control max-sessions="2" />
           </session-management>
</http>
```

Ή μέσω Java Configuration:

```
@Override
protected void configure(HttpSecurity http) throws Exception {
           http.sessionManagement().maximumSessions(2)
}
```

3.4.2.2.2 Διαχείριση Χρονικού Ορίου Session

Μετά την λήξη χρονικού ορίου ενός session, αν ο χρήστης στείλει ένα request με ληγμένο session id, πρέπει να ανακατευθύνεται στο κατάλληλο URL.

```
<session-management>
           <concurrency-control expired-url="/sessionExpired.html" ... />
</session-management>
```

Παρομοίως αν ο χρήστης στείλει ένα request με ένα session id που δεν έχει λήξει, αλλά είναι μη έγκυρο, πρέπει επίσης να γίνεται ανακατεύθυνση στο κατάλληλο URL:

```
<session-management invalid-session-url="/invalidSession.html">
           ...
</session-management>
```

Το αντίστοιχο Java configuration:

```
http.sessionManagement()
         .expiredUrl("/sessionExpired.html")
         .invalidSessionUrl("/invalidSession.html");
```

Χρήση του Spring Boot
Με τη χρήση του Spring Boot μπορεί εύκολα κάποιος να ρυθμίσει την τιμή του χρόνου λήξης ενός session με χρήση των ιδιοτήτων:

```
server.servlet.session.timeout=15m
```

Αν δεν οριστεί μονάδα χρόνου, το Spring θα υποθέσει ότι είναι seconds.
Με λίγα λόγια, με αυτήν τη διαμόρφωση, μετά από 15 λεπτά αδράνειας, το session θα λήξει. Το session μετά από αυτήν την περίοδο θεωρείται άκυρο.

Εάν διαμορφωθεί το project για χρήση με Tomcat, πρέπει να υπάρχει κατά νου ότι υποστηρίζει μόνο την ακρίβεια λεπτού για το χρονικό όριο περιόδου λειτουργίας, με ελάχιστη τιμή το ένα λεπτό. Αυτό σημαίνει ότι αν καθοριστεί για παράδειγμα μια τιμή χρονικού ορίου 170 δευτερολέπτων, θα έχει ως αποτέλεσμα χρονικό όριο 2 λεπτών.

Τέλος, είναι σημαντικό να αναφερθεί ότι παρόλο που το Spring session υποστηρίζει μια παρόμοια property για αυτόν τον σκοπό (spring.session.timeout), εάν δεν έχει καθοριστεί, τότε η αυτόματη διαμόρφωση θα επιστρέψει στην τιμή της ιδιότητας που αναφέρθηκε παραπάνω.

3.4.2.2.3 Αποτροπή Χρήσης URL Παραμέτρων για Session Tracking

Η έκθεση πληροφοριών του session στη διεύθυνση URL αποτελεί αυξανόμενο κίνδυνο ασφάλειας.

Ξεκινώντας από το Spring 3.0, η λογική επανεγγραφής της διεύθυνσης URL που θα προσαρτούσε το jsessionid στη διεύθυνση URL μπορεί τώρα να απενεργοποιηθεί ρυθμίζοντας το disable-url-rewriting = "true" στο namespace <http>.

Εναλλακτικά, ξεκινώντας με το Servlet 3.0, ο μηχανισμός παρακολούθησης συνεδρίας μπορεί επίσης να διαμορφωθεί στο web.xml:

```
<session-config>
            <tracking-mode>COOKIE</tracking-mode>
</session-config>
```

Και προγραμματιστικά:

```
servletContext.setSessionTrackingModes(EnumSet.of(SessionTrackingMode.COOKIE));
```

Αυτός ο κώδικας επιλέγει που θα αποθηκευτεί το JSESSIONID - στο cookie ή στην παράμετρο URL.

3.4.2.2.4 Προστασία από Session Fixation

Το Framework προσφέρει προστασία ενάντια σε τυπικές επιθέσεις Session Fixation ρυθμίζοντας τι συμβαίνει σε ένα υπάρχον session όταν ο χρήστης προσπαθεί να αυθεντικοποιηθεί ξανά:

```
<session-management session-fixation-protection="migrateSession"> ...
```

Η αντίστοιχη ρύθμιση σε Java:

```
http.sessionManagement()
         .sessionFixation().migrateSession()
```

Από προεπιλογή το Spring Security έχει την προστασία ενεργοποιημένη (“migrateSession”). Κατα τη αυθεντικοποίηση νέου Http Session, το παλιό ακυρώνεται και τα στοιχεία του παλιού αντιγράφονται στο νέο.

Αν αυτό δεν είναι το επιθυμητό υπάρχουν δύο ακόμα επιλογές:

- ❏ όταν το “none” είναι ορισμένο, το original session δεν θα ακυρωθεί
- ❏ όταν το “newSession” είναι ορισμένο, ένα καθαρό session θα δημιουργηθεί χωρίς attributes από το παλιό sesion.

3.4.2.2.5 Ασφαλές Session Cookie

Για να ασφαλιστεί το Session Cookie μπορεί να χρησιμοποιηθεί τα httpOnly και Secure flags.

- ❏ **httpOnly**: αν είναι true δεν μπορεί η Javascript να διαβάσει το cookie (αποφυγή υποκλοπής μέσω Cross Site Scripting Attacks (XSS))
- ❏ **Secure**: αν είναι true τότε το cookie θα σταλεί μόνο σε HTTPS συνδέσεις.

Αυτά τα flags ορίζονται στο web.xml αρχείο:

```
<session-config>
           <session-timeout>1</session-timeout>
           <cookie-config>
               <http-only>true</http-only>
               <secure>true</secure>
           </cookie-config>
</session-config>
```

Αυτή η ρύθμιση είναι διαθέσιμη από το Java Servlet 3. Από προεπιλογή το flag httpOnly είναι true το Secure είναι false.
Η αντίστοιχη Java ρύθμιση είναι:

```
public class MainWebAppInitializer implements WebApplicationInitializer {
       @Override
       public void onStartup(ServletContext sc) throws ServletException {
```

```
            // ...
            sc.getSessionCookieConfig().setHttpOnly(true);
            sc.getSessionCookieConfig().setSecure(true);
        }
    }
```

Αν χρησιμοποιείται το Spring Boot, αυτά τα flags μπορούν να οριστούν στο application.properties:

```
server.servlet.session.cookie.http-only=true
server.servlet.session.cookie.secure=true
```

Τέλος το ίδιο μπορεί να επιτευχθεί με τη χρήση ενός Filter:

```
public class SessionFilter implements Filter {
        @Override
        public void doFilter(
          ServletRequest request, ServletResponse response, FilterChain chain)
          throws IOException, ServletException {
            HttpServletRequest req = (HttpServletRequest) request;
            HttpServletResponse res = (HttpServletResponse) response;
            Cookie[] allCookies = req.getCookies();
            if (allCookies != null) {
                Cookie session =
                  Arrays.stream(allCookies).filter(x -> x.getName().equals("JSESSIONID"))
                        .findFirst().orElse(null);

                if (session != null) {
                    session.setHttpOnly(true);
                    session.setSecure(true);
                    res.addCookie(session);
                }
            }
            chain.doFilter(req, res);
        }
    }
```

[61]

3.4.3 Ευπάθειες Συνόδου και Τεχνικές Μετριασμού τους

Οι Session Hijacking επιθέσεις περιλαμβάνουν την εκμετάλλευση του μηχανισμού ελέγχου του web session. Ο επιτιθέμενος παραβιάζει το session token είτε προβλέποντας το, είτε κλέβοντας το από έναν αυθεντικοποιημένο χρήστη, για να έχει πρόσβαση στον WebServer. Το session token μπορεί να παραβιαστεί με πολλούς τρόπους:

- ❏ Client-side επιθέσεις (κακόβουλοι JavaScript Κώδικες, XSS, Trojans, κλπ.)
- ❏ Man-in-the-browser επιθέσεις
- ❏ Man-in-the-middle επιθέσεις
- ❏ Προβλέψιμο session token
- ❏ Session Sniffing

[61] "Control the Session with Spring Security | Baeldung." 15 Αυγ. 2020, https://www.baeldung.com/spring-security-session. Πρόσβαση στις 17 Οκτ. 2020.

[62]

3.4.3.1 Τύποι επιθέσεων Session Hijacking

1) Επίθεση με χρήση client-side scripting:

Οι Client-side ευπάθειες όπως οι XSS επιθέσεις επιτρέπουν σε έναν επιτιθέμενο να κατασκευάσουν ένα κακόβουλο script για να κλέψουν το session ID ενός χρήστη από την εφαρμογή.

```
http://website.kom/<script>document.cookie="sessionid=abcd";</script>
```

2) Επίθεση με χρήση του <META> tag

Οι επιθέσεις code injection χρησιμοποιούνται από τους επιτιθέμενους για να κάνουν inject κακόβουλο κώδικα στο URL και να το στείλουν στον browser του θύματος.

```
http://website.kon/<meta http-equiv=Set-Cookie content="sessionid=abcd">
```

[63]

3) Επίθεση μέσω HTTP Header Response

Οι HTTP Header Injection ευπάθειες παρουσιάζονται όταν η είσοδος του χρήστη περιλαμβάνεται χωρίς έλεγχο στις Response Header του server. Συγκεκριμένα βασίζονται στην ιδέα ότι ένας επιτιθέμενος μπορεί να προκαλέσει τον server να δημιουργήσει ένα Response που περιλαμβάνει χαρακτήρες carriage-return και line-feed (ή% 0D και% 0A αντίστοιχα στις URI κωδικοποιημένες forms τους) δίνοντας τη δυνατότητα στον επιτιθέμενο να κάνει τον Server να στείλει κατασκευασμένα από τον επιτιθέμενο Headers.

```
http://inj.example.org/redirect.asp?origin=foo%0d%0aSet-Cookie:%20ASPSESSIONIDACCBBTCD=SessionFixed%0d%0a
```

```
HTTP/1.1 302 Object moved
Connection: close
Location: account.asp?origin=foo
Set-Cookie: ASPSESSIONID=SessionFixed
Content-Length: 121
```

[64]

4) Session Fixation

Μια τυπική επίθεση Session Fixation εκτελείται ως εξής:

Ο επιτιθέμενος αποκτά πρόσβαση στο login page της εφαρμογής και λαμβάνει ένα session identifier που δημιουργείται από την εφαρμογή. Αυτό το βήμα δεν είναι απαραίτητο εάν η εφαρμογή ιστού δέχεται αυθαίρετα session identifiers.

1. Ο επιτιθέμενος χρησιμοποιεί μια πρόσθετη τεχνική, όπως CRLF Injection, man-in-the-middle attack, social engineering κ.λπ. και κάνει το θύμα να χρησιμοποιήσει το session identifier που ξέρει ο επιτιθέμενος. Αυτό εξαρτάται από τον τρόπο με τον οποίο η

---

[62] "Session hijacking attack Software Attack | OWASP Foundation." https://owasp.org/www-community/attacks/Session_hijacking_attack. Πρόσβαση στις 18 Οκτ. 2020.

[63] "Session fixation Software Attack | OWASP Foundation." https://owasp.org/www-community/attacks/Session_fixation. Πρόσβαση στις 18 Οκτ. 2020.

[64] "HTTP Header Injection | GracefulSecurity." 7 Μαρ. 2016, https://gracefulsecurity.com/http-header-injection/. Πρόσβαση στις 18 Οκτ. 2020.

εφαρμογή χειρίζεται τα session identifiers. Μπορεί να είναι τόσο απλό όσο η αποστολή κακόβουλου URL, αλλά μπορεί επίσης να απαιτηθεί από τον επιτιθέμενο να δημιουργήσει έναν ψεύτικο ιστότοπο.

2. Το θύμα αποκτά πρόσβαση στο login page της εφαρμογής και συνδέεται στην εφαρμογή. Μετά την αυθεντικοποίηση, η εφαρμογή αντιμετωπίζει όποιον χρησιμοποιεί αυτό το αναγνωριστικό περιόδου λειτουργίας σαν να ήταν αυτός ο χρήστης.

3. Ο επιτιθέμενος χρησιμοποιεί το session identifier για πρόσβαση στην εφαρμογή, αναλαμβάνει το session του χρήστη και πλαστοπροσωπεί το θύμα. Οι περαιτέρω ενέργειες εξαρτώνται από τη λειτουργικότητα του εισβολέα και της εφαρμογής.

Τα ακριβή στάδια της επίθεσης και η δυσκολία της εξαρτώνται από διάφορους παράγοντες. Για παράδειγμα, πολλά εξαρτώνται από τον τρόπο χειρισμού των session identifiers από την εφαρμογή. Εάν η εφαρμογή αποδέχεται session identifiers από τη διεύθυνση URL (μέσω αίτησης GET), η επίθεση είναι απλή. Εάν η εφαρμογή δέχεται αναγνωριστικά περιόδου σύνδεσης από αιτήματα POST, ο εισβολέας ενδέχεται να χρειαστεί να δημιουργήσει έναν ψεύτικο ιστότοπο ηλεκτρονικού ψαρέματος. Γίνεται πιο δύσκολο (αλλά όχι αδύνατο) εάν τα αναγνωριστικά περιόδου λειτουργίας γίνονται αποδεκτά μόνο από cookie - ο εισβολέας πρέπει στη συνέχεια να χρησιμοποιήσει τεχνικές όπως το Cross-site Scripting (XSS).[65]

3.4.3.2 Αντίμετρα στο Session Hijacking

I. **Χρήση SSL για όλες τις επικοινωνίες:** Το SSL σημαίνει Secure Socket Layer. Είναι ένα πρωτόκολλο που χρησιμοποιείται για την ασφάλεια των επικοινωνιών μέσω του Διαδικτύου. Τα τμήματα των συνδέσεων δικτύου κρυπτογραφούνται στο επίπεδο μεταφοράς από SSL το οποίο χρησιμοποιεί συνδυασμό τεχνικής κρυπτογράφησης δημόσιου κλειδιού και συμμετρικού κλειδιού με 128/256 bit. Η μετάδοση κρίσιμων πληροφοριών, όπως αριθμός πιστωτικής κάρτας, διεύθυνση και άλλα στοιχεία πληρωμής μέσω του Διαδικτύου, αποτελούν απειλή. Το SSL αποτρέπει την κλοπή πληροφοριών και μειώνει την πιθανότητα Sniffing.

II. **Σύνδεση HTTPS και Secure Cookies:** Για να διασφαλιστεί η προστασία των cookies, πρέπει να περάσουν μέσω σύνδεσης HTTPS (Secure Flag activated). Οι τακτικές μεταδόσεις HTTP σε συνδυασμό με την επάρκεια ασφαλείας του SSL είναι γνωστές ως HTTPS. Η επικοινωνία μεταξύ πηγής και προορισμού γίνεται μόνο μέσω HTTP αλλά μέσω ασφαλούς σύνδεσης SSL που κρυπτογραφεί και αποκρυπτογραφεί τα δεδομένα που μεταδίδονται. Ακόμα κι αν ο εισβολέας καταγράψει τα κρυπτογραφημένα δεδομένα που μεταδίδονται μεταξύ δύο μερών, δεν είναι ωφέλιμο καθώς δεν μπορεί να αποκρυπτογραφηθεί.

III. **Λειτουργία αποσύνδεσης:** Είναι ένα από τα πιο αμυντικά βήματα για την αποφυγή παραβίασης συνεδρίας, επειδή επιβάλλει έλεγχο ταυτότητας κατά την έναρξη μιας άλλης περιόδου σύνδεσης. Εάν ο χρήστης δεν αποσυνδεθεί, η περίοδος λειτουργίας δεν τελειώνει. Ο εισβολέας μπορεί να κλέψει session tokens και να πάρει τον έλεγχο του session. Είναι επίσης πιθανό ο εισβολέας να αλλάξει ακόμη και τα διαπιστευτήρια σύνδεσης και ο χρήστης να μην μπορεί να ξανασυνδεθεί.

IV. **Δημιουργία αναγνωριστικών μετά από έγκυρη σύνδεση:** Η δημιουργία αναγνωριστικού περιόδου σύνδεσης πριν από τη σύνδεση το εκθέτει σε επιθέσεις Session Fixation. Ο εισβολέας μπορεί να “φιξάρει” το session ID πριν συνδεθεί ο χρήστης και να παραβιάσει τη συνεδρία. Η δημιουργία ID μετά τη σύνδεση αποτρέπει τον εισβολέα να κάνει την επίθεση. Έτσι, αποτρέπει τον εισβολέα από το session hijacking.

V. **Long Session IDs:** Τα Session keys είναι πολύ σημαντικά στην επικοινωνία. Μπορούν να προσδιοριστούν εύκολα με τη βοήθεια μιας επίθεσης brute force, εάν το μήκος του Session Key είναι μικρό. Ως εκ τούτου, για να αποφευχθεί αυτός ο κίνδυνος, μία συμβολοσειρά ή ένας μακρύς τυχαίος αριθμός πρέπει να χρησιμοποιείται ως Session Key.

VI. **Εκπαίδευση των χρηστών:** Οι χρήστες πρέπει να εκπαιδεύονται ώστε να λαμβάνουν προφυλάξεις, όπως το να δίνουν προσοχή στην διαφορά http vs https, να αποσυνδέονται σωστά, να πληκτρολογούν τον σύνδεσμο αντί να κάνουν κλικ σε αυτόν κλπ, προκειμένου να αποφευχθεί το Session Hijacking. Υπάρχει ανάγκη εκπαίδευσης των τελικών χρηστών σχετικά με το πώς είναι μια ψεύτικη σελίδα, τους ισχυρούς κωδικούς πρόσβασης, τη χρήση του HTTPS, του διακομιστή μεσολάβησης, της επιλογής Remember Password, των Keyloggers κ.λπ. Έτσι, σε περίπτωση που ο εισβολέας καταφέρει να κάνει Session hijack να, μπορούν να ληφθούν τα κατάλληλα μέτρα.

VII. **Αναγκαστική επανάληψη αυθεντικοποίησης:** Αναγκάζει τον χρήστη να ξανασυνδεθεί μετά από μια καθορισμένη χρονική περίοδο. Κατά την επανάληψη του login, το id-session δημιουργείται ξανά.Το προηγούμενο session identifier σύνδεσης ακυρωνεται. Εάν ο χρήστης ξεχάσει να αποσυνδεθεί, ο εισβολέας έχει την ευκαιρία να κάνει capture το Session ID και να κάνει

[65] "What Is Session Fixation | Acunetix." 12 Δεκ. 2019, https://www.acunetix.com/blog/web-security-zone/what-is-session-fixation/. Πρόσβαση στις 18 Οκτ. 2020.

Hijack το Session. Επομένως, οι περίοδοι σύνδεσης πρέπει να λήγουν μετά από ένα καθορισμένο χρονικό διάστημα για να αποφευχθεί η επαναχρησιμοποίηση του καταγεγραμμένου Session ID από τον εισβολέα.

VIII. **Αναγκαστικός έλεγχος ταυτότητας ή επαλήθευση ταυτότητας:** Αναγκάζει τον χρήστη να συνδεθεί ξανά μετά από μια καθορισμένη χρονική περίοδο. Αν ξανά κάνει login, το id-session δημιουργείται ξανά. Το προηγούμενο Session ID σύνδεσης πρέπει να γίνεται μη έγκυρο. Επομένως, εάν ο εισβολέας καταγράψει το session-id, δεν είναι χρήσιμο. Ακόμα κι αν το Session έχει γίνει Hijacked περιορίζει τη ζημιά, καθώς θα απαιτείται εκ νέου έλεγχος ταυτότητας μετά από καθορισμένο χρονικό διάστημα.

IX. **Πρόληψη Captcha**: Το CAPTCHA σημαίνει «Πλήρως αυτοματοποιημένη δημόσια δοκιμή Turing για να διακρίνει τους υπολογιστές από τους ανθρώπους. Πρόκειται για μια οπτική εικόνα που περιέχει συνδυασμό χαρακτήρων που πρέπει να εισαχθεί τη στιγμή της σύνδεσης. Οι άνθρωποι θα μπορούν να διαβάζουν και να εισάγουν χαρακτήρες, αλλά ο υπολογιστής δεν θα μπορούσε να το ερμηνεύσει και ως εκ τούτου, δεν θα μπορεί να συνδεθεί. Το CAPTCHA βοηθά στη δημιουργία μιας περιόδου σύνδεσης για έναν μόνο χρήστη και περιορίζει τις πολλές συνεδρίες. Έτσι, ο εισβολέας δεν θα μπορούσε να συνδεθεί, όταν ο χρήστης έχει ήδη συνδεθεί.[66]

3.4.4 Διαχείριση Συνόδου στο JSF

Η διαχείριση συνόδου είναι πολύ σημαντική για όλες τις εφαρμογές web και χρησιμοποιείται για τους παρακάτω σκοπούς:

- ❏ Περιορισμός πρόσβασης σε σελίδα μετά το πέρας του χρονικού ορίου του συνόδου.
- ❏ Περιορισμός των URL Entries επικυρώνοντας τη σύνοδο.
- ❏ Περιορισμός πρόσβασης μη αυθεντικοποιημένων χρηστών.

Παρακάτω παρατίθεται παράδειγμα ελέγχου συνόδου:

Βήμα 1: Δημιουργία δύο JSF σελίδων
1. Login.xhtml
2. Home.xhtml

Βήμα 2: Δημιουργία ενός managed bean
Βήμα 3: Μετά τη δημιουργία των JSF σελίδων πρέπει να δημιουργηθεί η κλάση FIlter με όνομα SessionTimeoutFilter που επεξεργάζεται το μηχανισμό φιλτραρίσματος.

```
public class SessionTimeoutFilter implements Filter {
  private String timeoutPage = "Login";
  public void init(FilterConfig filterConfig) throws ServletException {
    //We will not process anything in init method so we can omit this part too.
  }
  //Triggers for every faces-servlet request
   public void doFilter(ServletRequest request, ServletResponse response, FilterChain filterChain) throws
IOException,
  ServletException {
    if ((request instanceof HttpServletRequest) && (response instanceof HttpServletResponse)) {
      HttpServletRequest httpServletRequest = (HttpServletRequest) request;
      HttpServletResponse httpServletResponse = (HttpServletResponse) response;
      // is session expire control required for this request?
      if (isSessionControlRequiredForThisResource(httpServletRequest)) {
        // is session invalid?
        if (isSessionInvalid(httpServletRequest)) {
```

[66] "(PDF) Session Hijacking: Threat Analysis and Countermeasures." 28 Μαΐ. 2015, https://www.researchgate.net/publication/277307339_Session_Hijacking_Threat_Analysis_and_Countermeasures. Πρόσβαση στις 18 Οκτ. 2020.

```
          String timeoutUrl = httpServletRequest.getContextPath() + "/" + getTimeoutPage();
          System.out.println("Session is invalid! redirecting to timeoutpage : " + timeoutUrl);
          httpServletResponse.sendRedirect(timeoutUrl);
          return;
        }
      }
      filterChain.doFilter(request, response);
    }
    private boolean isSessionControlRequiredForThisResource(HttpServletRequest httpServletRequest) {
      String requestPath = httpServletRequest.getRequestURI();
      boolean controlRequired = !StringUtils.contains(requestPath, getTimeoutPage());
      return controlRequired;
    }
    //Check whether the session is  valid
    private boolean isSessionInvalid(HttpServletRequest httpServletRequest) {
      boolean sessionInValid = (httpServletRequest.getRequestedSessionId() != null)
      && !httpServletRequest.isRequestedSessionIdValid();
      return sessionInValid;
    }
    public void destroy() {
}
    public String getTimeoutPage() {
      //Return timeout page to which we mentioned above
      return timeoutPage;
    }
    public void setTimeoutPage(String timeoutPage)
    {
      //Set timeout page to which we mentioned ablove
      this.timeoutPage = timeoutPage;
    }
```

Βήμα 4: Δημιουργία MyActionListener

Αυτή η κλάση έχει σκοπό τη δημιουργία και διατήρηση των συνόδων

```
public class MySessionListener implements HttpSessionListener {
  public MySessionListener() {
}
  public void sessionCreated(HttpSessionEvent event) {
    System.out.println("Current Session created : " + event.getSession().getId() + " at " + new Date());
  }
  public void sessionDestroyed(HttpSessionEvent event) {
    // get the destroying session...
    HttpSession session = event.getSession();
    System.out.println("Current Session destroyed :" + session.getId() + " Logging out user...");
    // Only if needed
    try {
      prepareLogoutInfoAndLogoutActiveUser(session);
    }
    catch(Exception e) {
      System.out.println("Error while logging out at session destroyed : " + e.getMessage());
    }
  }
  //Clean your logout operations.
  public void prepareLogoutInfoAndLogoutActiveUser(HttpSession httpSession) {
    // Only if needed
```

```
  }
```

Βήμα 5: Παραμετροποίηση του web.xml
Μετά τη δημιουργία των κλάσεων Filter χρειάζεται να συμπεριληφθούν οι κλάσεις στο Faces Servlet στο web.xml αρχείο.

```
<listener>
    <listener-class>bean.MySessionListener</listener-class>
</listener>
<filter>
    <filter-name>SessionTimeoutFilter</filter-name>
    <filter-class>bean.SessionTimeoutFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>SessionTimeoutFilter</filter-name>
    <servlet-name>Faces Servlet</servlet-name>
</filter-mapping>
```

[67]

3.4.4.1 Καλές Πρακτικές για ασφαλή διαχείρηση Συνόδου

Ακύρωση Συνόδου
Η σύνοδος ενός χρήστη ακυρώνεται χειροκίνητα ή αυτόματα, ανάλογα που τρέχει το servlet. Η ακύρωση μίας συνόδου σημαίνει να διαγραφεί το HttpSession object και οι τιμές του από το σύστημα.

```
HttpServletResponse response = (HttpServletResponse) invocation;
getInvocationContext().get(StrutsStatics.HTTP_RESPONSE);
HttpServletRequest request = (HttpServletResponse) invocation;
getInvocationContext().get(StrutsStatics.HTTP_REQUEST);
request.getSession().invalidate();
response.sendRedirect("http://redirectpage.org");
```

Παραμετροποίηση του Session Cookie
Πρέπει να αλλάζει το όνομα του cookie από JSESSIONID, ώστε να προστατεύεται η εφαρμογή από overwrite τιμές από άλλη εφαρμογή.

```
@Bean
	public CookieSerializer cookieSerializer() {
		DefaultCookieSerializer serializer = new DefaultCookieSerializer();
		serializer.setCookieName("MyInfoCookie");
		serializer.setCookiePath("/")
		serializer.setDomainNamePattern("^.+?\\.(\\w+\\.[a-z]+)$");
		return serializer;
	}
```

Παραγωγή τυχαία bytes για SessionID
Όταν η τυχαιότητα του SessionID είναι χαμηλή, ο επιτιθέμενος έχει πλεονέκτημα στο να μαντέψει ένα sessionID. Οπότε η παραγωγή τυχαίων και δύσκολα προβλέψιμων Session IDs είναι υψίστης σημασίας.

---

[67] "Session Management in JSF - JavaFs - blogger." 3 Σεπ. 2013, http://javaannals.blogspot.com/2013/09/session-management-in-jsf.html. Πρόσβαση στις 19 Οκτ. 2020.

```
public static String generateRandomToken(){
	return generateUniqueToken(18);
}
public static byte[] generateRandombytes(int size){
	final byte[] randomBytes = new byte[size];
	FAST_RANDOM_GENERATOR.netxtBytes(randomBytes);
	return randomBytes;
}
```

3.4.4.2 Προϋποθέσεις για ασφαλή διαπιστευτήρια και αναγνωριστικά συνόδου

➔ Μην αποθηκεύετε συμβολοσειρές σύνδεσης, κωδικούς πρόσβασης ή άλλες ευαίσθητες πληροφορίες σε καθαρό κείμενο ή με μη κρυπτογραφικά ασφαλή τρόπο από την πλευρά του πελάτη.

➔ Για τη μετάδοση διαπιστευτηρίων ελέγχου ταυτότητας να χρησιμοποιείται μόνο η μέθοδος αιτήματος HTTP POST.

➔ Εάν η εφαρμογή μπορεί να χειριστεί ένα credential store, θα πρέπει να είναι βέβαιο ότι έχουν αποθηκευτεί μόνο κρυπτογραφικά ισχυροί και one way salted hashes των κωδικών πρόσβασης.

➔ Εφαρμογή ενός χρονικού ορίου αδράνειας περιόδου λειτουργίας που είναι όσο το δυνατόν λιγότερο, με βάση τις λειτουργικές απαιτήσεις της επιχείρησης και τον κίνδυνο εξισορρόπησης.

➔ Δημιουργία ενός νέου αναγνωριστικού περιόδου σύνδεσης σε κάθε επανάληψη αυθεντικοποίησης.

➔ Δημιουργία νέου αναγνωριστικού περιόδου σύνδεσης και απενεργοποίηση του παλιού.

➔ Δημιουργία νέου αναγνωριστικού περιόδου σύνδεσης, εάν η ασφάλεια σύνδεσης αλλάξει από HTTP σε HTTPS, όπως μπορεί να συμβεί κατά τον έλεγχο ταυτότητας.

➔ Οι ερωτήσεις επαναφοράς κωδικού πρόσβασης πρέπει να υποστηρίζονται όταν υπάρχουν επαρκώς τυχαίες απαντήσεις στις ερωτήσεις επαναφοράς κωδικού πρόσβασης.

➔ Απαγόρευση ταυτόχρονων συνδέσεων εάν χρησιμοποιείται το ίδιο αναγνωριστικό χρήστη.

3.4.4.3 Οδηγίες για Ασφαλή διαχείρηση Συνόδου

❑ Απενεργοποίηση της λειτουργίας Remember me για τα πεδία κωδικών.

❑ Απενεργοποίηση των χαρακτηριστικών αυτόματης συμπλήρωσης σε φόρμες πού είσοδος περιμένει ευαίσθητες πληροφορίες, συμπεριλαμβανομένου της αυθεντικοποίησης.

❑ Επιβολή πολιτικής για το μήκος του κλειδιού. Συνήθως χρησιμοποιούνται 8 χαρακτήρες ως κατώτατο όριο μήκους κλειδιού.

❑ Διαβεβαίωση κλειδώματος λογαριασμού μετά από συγκεκριμένο αριθμό μη έγκυρων προσπαθειών σύνδεσης (για παράδειγμα πάνω από πέντε προσπάθειες).

❑ Αν υποστηρίζονται resets με βάση το email, να γίνεται αποστολή μόνο σε έγκυρα καταγεγραμμένα e-mails, ένας κωδικός ή ένα προσωρινός σύνδεσμος.

❑ Πρόληψη κατά της επαναχρησιμοποίησης κωδικού.

❑ Απαίτηση αυθεντικοποίησης σε όλες τις σελίδες εκτός από αυτές που είναι συγκεκριμένα για όλους.

❏ Σύνδεσμοι και προσωρινοί κωδικοί πρέπει να έχουν μικρή προθεσμία λήξης.

3.5 Διαχείριση Σφαλμάτων

3.5.1 Εισαγωγή στις εξαιρέσεις και στη Διαχείριση Σφαλμάτων στην Java

Μία εξαίρεση είναι ένα πρόβλημα που εμφανίζεται κατά τη διάρκεια της εκτέλεσης ενός προγράμματος. Για αυτό το λόγο αυτές οι εξαιρέσεις πρέπει να μεταχειρίζονται κατάλληλα. Η διαχείριση σφαλμάτων στην Java περιλαμβάνει την εκτέλεση πολλών εκατοντάδων γραμμών του κώδικα. Πρέπει να δοθεί επίσης προσοχή τα try και catch blocks μέσα σε loops, τα οποία πρέπει να μετακινηθούν εκτός του loop.

Οι εξαιρέσεις χωρίζονται σε δύο κατηγορίες:
1. Checked (compilation)
2. Unchecked (runtime)

3.5.1.1 Checked εξαίρεση

Είναι μία εξαίρεση που ο compiler της Java απαιτεί από το χρήστη να διαχειριστεί. Αν κάποιος κώδικας μέσα σε μία μέθοδο δώσει μία checked εξαίρεση τότε πρέπει η μέθοδος είτε να διαχειριστεί την εξαίρεση η να προσδιορίσει την εξαίρεση χρησιμοποιώντας το keyword **throws.**

Για παράδειγμα το επόμενο πρόγραμμα Java ανοίγει ένα αρχείο στην τοποθεσία “C:\test\a.txt” και εκτυπώνει τις πρώτες τρεις γραμμές του. Το πρόγραμμα δεν κάνει compile επειδή η function main() χρησιμοποιεί το FileReader() το οποίο δίνει την εξαίρεση FileNotFoundException. Επίσης χρησιμοποιεί την readLine() και την close() μέθοδο, οι οποίες δίνουν επίσης την checked εξαίρεση IOException.

```
import java.io.*;
class Main {
    public static void main(String[] args) {
        FileReader file = new FileReader("C:\\test\\a.txt");
        BufferedReader fileInput = new BufferedReader(file);

        // Print first 3 lines of file "C:\test\a.txt"
        for (int counter = 0; counter < 3; counter++)
            System.out.println(fileInput.readLine());

        fileInput.close();
    }
}
```

Έξοδος

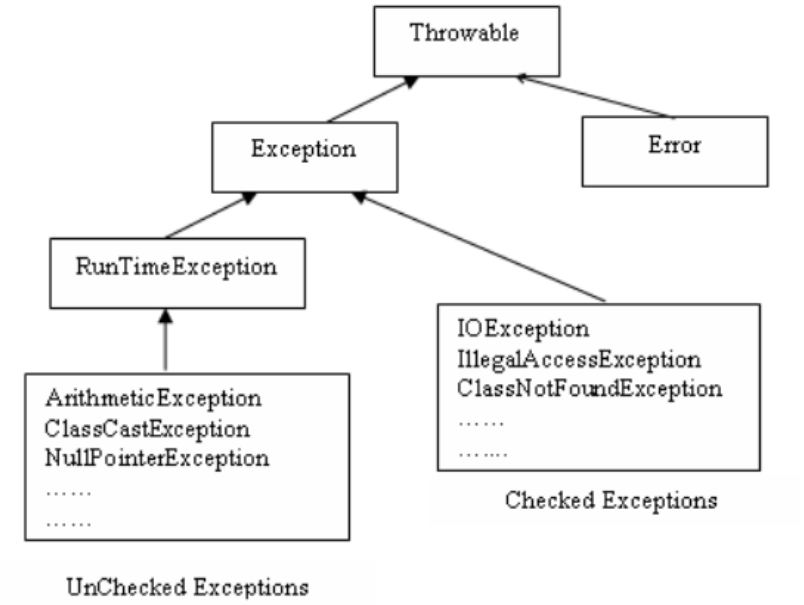
```
Exception in thread "main" java.lang.RuntimeException: Uncompilable source code - unreported exception
java.io.FileNotFoundException; must be caught or declared to be
thrown
    at Main.main(Main.java:5)
```

Για να διορθωθεί το παραπάνω πρόγραμμα, πρέπει είτε να καθοριστεί μια λίστα εξαιρέσεων χρησιμοποιώντας throws, είτε πρέπει να χρησιμοποιηθούν try-catch blocks. Παρακάτω φαίνεται το πρόγραμμα με χρήση των throws. Επειδή το FileNotFoundException είναι υποκλάση του IOException, μπορεί απλά να προσδιοριστεί το IOException στην λίστα throws και να γίνει το παραπάνω πρόγραμμα error-free.

```
import java.io.*;

class Main {
	public static void main(String[] args) throws IOException {
		FileReader file = new FileReader("C:\\test\\a.txt");
		BufferedReader fileInput = new BufferedReader(file);

		// Print first 3 lines of file "C:\test\a.txt"
		for (int counter = 0; counter < 3; counter++)
			System.out.println(fileInput.readLine());

		fileInput.close();
	}
}
```

3.5.1.2 Unchecked εξαίρεση

Αυτές οι εξαιρέσεις δεν ελέγχονται στο Compilation time αλλά κατά τη διάρκεια του runtime. Στην Java οι εξαιρέσεις από τις κλάσεις Error και RuntimeExceptions είναι unchecked εξαιρέσεις, ενώ όλες οι υπόλοιπες κάτω από την κλάση throwable είναι checked όπως φαίνεται παρακάτω:



Για παράδειγμα το παρακάτω Java πρόγραμμα κάνει compile σωστά αλλά δίνει ArithmeticException κατά τη διάρκεια της εκτέλεσης. Ο compiler το αφήνει να κάνει compile επειδή η ArithmeticException είναι μία unchecked εξαίρεση.

```
class Main {
public static void main(String args[]) {
	int x = 0;
```

```
        int y = 10;
        int z = y/x;
}
}
```

Έξοδος

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Main.main(Main.java:5)
Java Result: 1
```

[68]

Συμπεράσματα

Στην ερώτηση αν πρέπει οι εξαιρέσεις να είναι checked ή unchecked, η απάντηση είναι ότι, αν ο client μπορεί να ανακάμψει από μία εξαίρεση, τότε πρέπει η εξαίρεση να είναι checked διαφορετικά, αν ο client δεν μπορεί να κάνει κάτι για ανάκαμψη από την εξαίρεση πρέπει να γίνει unchecked η εξαίρεση.

Επίσης κάτι άλλο πολύ σημαντικό είναι να γίνονται throw οι checked εξαιρέσεις, μόνο όταν ο προγραμματιστής είναι σίγουρος ότι η μέθοδος θα είναι σε θέση να πάρει τα κατάλληλα μέτρα για τη διαχείριση και την ανάκαμψη από την εξαίρεση. Όσο λιγότερες checked εξαιρέσεις έχει ένα πρόγραμμα τόσο πιο απλή βάση με λιγότερα try-catch blocks έχει. Τέλος ακατάλληλα μηνύματα σφάλματος μπορεί να αποκαλύψουν πληροφορίες σχετικά με την εφαρμογή, οι οποίες να βοηθήσουν έναν επιτιθέμενο στην παραβίαση της εφαρμογής.Με ακατάλληλη διαχείριση σφαλμάτων εμφανίζονται στον τελικό χρήστη αναλυτικά εσωτερικά σφάλματα όπως περιεχόμενα βάσεις δεδομένων λεπτομέρειες για εξαιρέσεις και σφάλματα αλλά και stack traces. Η μέθοδος printStackTrace() αποκαλύπτει λεπτομέρειες για την εφαρμογή και τις τεχνολογίες που χρησιμοποιούνται. Για τη διόρθωση τέτοιων ευπαθειών πρέπει να γίνεται ανακατεύθυνση όλων των μηνυμάτων σφάλματος σε ένα log αρχείο. Αυτό μπορεί να γίνει με χρήση του Log4j. [69]

3.5.1.3 Τρόποι Διαχείρησης Εξαιρέσεων

Η Java προσφέρει δύο διαφορετικές επιλογές για τη διαχείριση των εξαιρέσεων. Ο προγραμματιστής μπορεί είτε να χρησιμοποιήσει την προσέγγιση try-catch-finally, είτε την προσέγγιση try-with-resource.

3.5.1.3.1 Try-Catch-Finally

Αυτή είναι μία κλασική προσέγγιση για τη διαχείριση των εξαιρέσεων στην Java. Περιέχει 3 βήματα:

- ένα try block που περικλείει ένα κομμάτι κώδικα που μπορεί να κάνει throw μία εξαίρεση
- ένα ή περισσότερα catch blocks που διαχειρίζονται την εξαίρεση
- και ένα finally block που εκτελείται μετά την επιτυχή εκτέλεση Το try block ή μετά τη διαχείριση μιας thrown εξαίρεσης.

Το try block είναι απαραίτητο και μπορεί να χρησιμοποιηθεί με ή χωρίς το catch ή το finally block.

Try Block

Το try block περικλείει το κομμάτι του Κώδικα που μπορεί να κάνει throw μία εξαίρεση. Αν ο κώδικας κάνει throw περισσότερες από μία εξαιρέσεις ο προγραμματιστής μπορεί να διαλέξει αν θέλει:

- να χρησιμοποιήσει ένα ξεχωριστό try block για κάθε κομμάτι που μπορεί να κάνει throw μία εξαίρεση.

---

[68] "Checked vs Unchecked Exceptions in Java - GeeksforGeeks." https://www.geeksforgeeks.org/checked-vs-unchecked-exceptions-in-java/. Πρόσβαση στις 22 Οκτ. 2020.

[69] "How to print an exception stack trace using Log4J (or ...." 5 Φεβ. 2017, https://alvinalexander.com/blog/post/java/how-print-exception-stack-trace-using-log4j-commons/. Πρόσβαση στις 22 Οκτ. 2020.

❑ να χρησιμοποιήσει ένα try block για πολλαπλά κομμάτια κώδικα που μπορεί να κάνουν throw πολλαπλές εξαιρέσεις.

Το παρακάτω παράδειγμα δείχνει ένα try block το οποίο περικλείει τρεις μεθόδους:

```
public void performBusinessOperation() {
       try {
              doSomething("A message");
              doSomethingElse();
              doEvenMore();
       }
       // see following examples for catch and finally blocks
      }

      public void doSomething(String input) throws MyBusinessException {
       // do something useful ...
       throw new MyBusinessException("A message that describes the error.");
      }

      public void doSomethingElse() {
       // do something else ...
      }

      public void doEvenMore() throws NumberFormatException{
       // do even more ...
      }
```

Φαίνεται στους ορισμούς των μεθόδων, μόνο η πρώτη και η τρίτη μέθοδος προσδιορίζουν μία εξαίρεση. Η πρώτη μπορεί να κάνει throw μία MyBusinessException, και η μέθοδος doEvenMore μπορεί να κάνει throw μία NumberFormatException.

Στο επόμενο βήμα ο προγραμματιστής μπορεί να ορίσει ένα catch block για κάθε κλάση εξαίρεσης που θέλει να διαχειριστεί και ένα finally block. Όλες οι checked εξαιρέσεις που δεν διαχειρίζονται από κανένα block πρέπει να προσδιοριστούν.

Catch Block

Σε ένα catch block μπορεί να γίνει υλοποίηση της διαχείρισης ενός ή περισσότερων τύπων εξαίρεσης.Οπότε στον επόμενο κώδικα το catch block παίρνει την εξαίρεση ως παράμετρο. επίσης μπορεί να γίνει αναφορά την εξαίρεση μέσω της παραμέτρου.

```
public void performBusinessOperation() {
       try {
              doSomething("A message");
              doSomethingElse();
              doEvenMore();
       } catch (MyBusinessException e) {
              e.printStackTrace();
       } catch (NumberFormatException e) {
              e.printStackTrace();
       }
```

Ο προηγούμενος κώδικας δείχνει δύο catch blocks. Το ένα διαχειρίζεται την εξαίρεση MyBusinessException και μία για τη διαχείριση της εξαίρεσης NumberFormatException. Και τα δύο blocks διαχειρίζονται τις εξαιρέσεις με τον ίδιο τρόπο. Από την Java 7, Ο προγραμματιστής μπορεί να κάνει το ίδιο με μόνο ένα catch block.

```
public void performBusinessOperation() {
```

```
        try {
                doSomething("A message");
                doSomethingElse();
                doEvenMore();
        } catch (MyBusinessException|NumberFormatException e) {
                e.printStackTrace();
        }
}
```

Finally Block

Το finally block εκτελείται μετά την επιτυχή εκτέλεση του try block ή εάν μετά από ένα catch block διαχειριστεί μία εξαίρεση. Για αυτό το λόγο είναι ένα καλό σημείο για να ενσωματωθεί μία λογική “καθαρισμού”, όπως για παράδειγμα το κλείσιμο μιας σύνδεσης ή ενός inputstream.

Παρακάτω φαίνεται ένα παράδειγμα λειτουργίας καθαρισμού. Το Final block θα εκτελεστεί ακόμα και αν το FileInputStream κάνει throw ένα FileNotFoundException, ή η επεξεργασία του περιεχομένου του αρχείου κάνει throw οποιαδήποτε άλλη εξαίρεση.

```
FileInputStream inputStream = null;
try {
        File file = new File("./tmp.txt");
        inputStream = new FileInputStream(file);

        // use the inputStream to read a file

} catch (FileNotFoundException e) {
        e.printStackTrace();
} finally {
        if (inputStream != null) {
                try {
                        inputStream.close();
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
}
```

Όπως φαίνεται το finally block προσφέρει μία καλή επιλογή για την αποφυγή διαρροών πληροφορίας. Ειδικά πριν την Java 7 ήταν η καλύτερη πρακτική η τοποθέτηση κώδικα καθαρισμού στο finally block.

3.5.1.3.2 Try-With-Resource

Με την εμφάνιση της Java 7 ήρθε και το try-with-resource statement. Κλείνει αυτόματα όλους τους πόρους που περιέχουν AutoCloseable interface. Κάτι που συμβαίνει στα περισσότερα Java objects αυτή τη στιγμή.

Το μόνο πράγμα που χρειάζεται να κάνει ο προγραμματιστής για να χρησιμοποιήσει αυτό το χαρακτηριστικό είναι να δημιουργήσει ένα instance του αντικειμένου μέσα στο try clause. Επίσης ο προγραμματιστής πρέπει να διαχειριστεί η να προσδιορίσει όλες τις εξαιρέσεις που μπορεί να γίνουν throw όταν κλείνει ένας πόρος.

Το παρακάτω τμήμα κώδικα δείχνει το προηγούμενο παράδειγμα με ένα statement try-with-resource αντί του statement try-catch-finally.

```
File file = new File("./tmp.txt");
try (FileInputStream inputStream = new FileInputStream(file);) {
	// use the inputStream to read a file
} catch (FileNotFoundException e) {
	e.printStackTrace();
} catch (IOException e) {
	e.printStackTrace();
}
```

Όπως φαίνεται το statement try-with-resource είναι πολύ πιο εύκολο να ενσωματωθεί και να διαβαστεί. Επίσης η διαχείριση του IOException που μπορεί να γίνει throw κατά τη διάρκεια κλεισίματος ενός FileInputStream δεν απαιτεί ένα εμφωλευμενο try-catch statement.[70]

3.5.2 Ειδικές Συμπεριφορές

3.5.2.1 Μη καταστολή ή αγνόηση checked εξαιρέσεων

Οι προγραμματιστές συχνά καταστέλλουν checked εξαιρέσεις κάνοντας catch εξαιρέσεις με ένα άδειο ή ασήμαντο catch block. Κάθε catch block πρέπει να διαβεβαιώνει ότι το πρόγραμμα συνεχίζει μόνο με έγκυρα invariants. Συνεπώς το catch block πρέπει, είτε να ανακάμπτει από την exceptional συνθήκη και να κάνει rethrow την εξαίρεση για να επιτρέψει στο επόμενο κοντινότερο catch ενός try statement να ανακάμψει ή να κάνει throw μία εξαίρεση πού είναι η κατάλληλη στο περιεχόμενο του catch block.

Οι εξαιρέσεις διακόπτουν την αναμενόμενη ροή ελέγχου της εφαρμογής. Για παράδειγμα, κανένα μέρος έκφρασης ή δήλωσης που συμβαίνει μέσα στο try block μετά από το σημείο στο οποίο η εξαίρεση γίνεται throw, ελέγχεται. Κατά συνέπεια οι εξαιρέσεις πρέπει να διαχειρίζονται κατάλληλα.

3.5.2.1.1 Παράδειγμα κώδικα που δεν συμμορφώνεται

Για παράδειγμα αν ένα Thread διακοπεί κατά τη διάρκεια sleeping ή waiting, προκαλεί μία java.lang.InterruptedException να γίνει throw. Όμως η μέθοδος run() του interface Runnable δεν μπορεί να κάνει throw μία checked εξαίρεση και πρέπει να διαχειριστεί την InterruptedException. Ο παρακάτω μη συμμορφούμενος κώδικας κάνει catch και καταστέλλει το InterruptedException.

```
class Foo implements Runnable {
  public void run() {
    try {
      Thread.sleep(1000);
    } catch (InterruptedException e) {
      // Ignore
    }
  }
}
```

Αυτός ο κώδικας αποτρέπει του callers της μεθόδου run() να προσδιορίσουν ότι έχει συμβεί μία interrupted εξαίρεση. Κατά συνέπεια η caller μέθοδοι όπως η Thread.start() δεν μπορούν να δράσουν στην εξαίρεση. Παρομοίως αν ο κώδικας καλούνταν στο δικό του thread, θα απέτρεπε το κάλεσμα το thread ξέροντας ότι το thread έχει γίνει interrupt.

---

[70] "Java Exception Handling: How to Specify and ... - Stackify." 17 Ιουλ. 2017, https://stackify.com/specify-handle-exceptions-java/. Πρόσβαση στις 22 Οκτ. 2020.

3.5.2.1.2 Συμβατή λύση

Η παρακάτω λύση κάνει catch την InterruptedException και επαναφέρει το interrupted status καλώντας την μέθοδο interrupt() στο τρέχον Thread.

```
class Foo implements Runnable {
  public void run() {
    try {
      Thread.sleep(1000);
    } catch (InterruptedException e) {
      Thread.currentThread().interrupt(); // Reset interrupted status
    }
  }
}
```

Κατά συνέπεια το κάλεσμα μεθόδων (ή κώδικα από ένα calling Thread) μπορεί να προσδιορίσει, ότι έχει συμβεί ένα interrupt.

3.5.2.1.3 Risk Assessment

Η αγνόηση η καταστολή εξαιρέσεων μπορεί να έχει ως αποτέλεσμα ασυνεπή κατάσταση προγράμματος.

Κανόνας	**Severity**	**Πιθανότητα**	**Κόστος Ανάκαμψης**	**Προτεραιότητα**	**Επίπεδο**
ERR00-J	Low	Probable	Medium	P4	L3

3.5.2.1.4 Αυτοματοποιημένη ανίχνευση

Η ανίχνευση των εξαιρέσεων που έχουν υποστεί καταστολή είναι εύκολο ζήτημα. Δεν είναι εφικτός ο σωστός προσδιορισμός των συγκεκριμένων περιπτώσεων που παραβιάζουν αυτόν τον κανόνα και οι οποίες αντιπροσωπεύουν επιτρεπόμενες εξαιρέσεις από τον κανόνα. Οι ευρετικές προσεγγίσεις μπορεί να είναι αποτελεσματικές.

Εργαλείο	**Έκδοση**	**Checker**	**Περιγραφή**
CodeSonar	5.4p0	**FB.BAD_PRACTICE.DE_MIGHT_IGNORE**	Η μέθοδος μπορεί να αγνοήσει την εξαίρεση
Coverity	7.5	**MISSING_THROW**	Ενσωματωμένο
Parasoft Jtest	10.3	**SECURITY.UEHL.LGE, UC.UCATCH**	Ενσωματωμένο
SonarQube	6.7	**S1166**	Οι διαχειριστές εξαιρέσεων πρέπει να διατηρούν τις αρχικές εξαιρέσεις

3.5.2.1.5 Σχετικές Ευπάθειες

Το AMQ-1272 περιγράφει μια ευπάθεια στην υπηρεσία ActiveMQ. Όταν το ActiveMQ λαμβάνει ένα μη έγκυρο όνομα χρήστη και κωδικό πρόσβασης από έναν Stomp client, δημιουργείται μια εξαίρεση ασφαλείας, αλλά στη συνέχεια αγνοείται, αφήνοντας τον πελάτη συνδεδεμένο με πλήρη και απεριόριστη πρόσβαση στο ActiveMQ.

3.5.2.1.6 Σχετικές Οδηγίες

MITRE CWE	CWE-390, Εντοπισμός κατάστασης σφάλματος χωρίς ενέργεια

[71]

3.5.2.2 Αποτροπή έκθεσης ευαίσθητων πληροφοριών από εξαιρέσεις

Η αποτυχία φιλτραρίσματος ευαίσθητων πληροφοριών κατά τη διάδοση εξαιρέσεων συχνά οδηγεί σε διαρροές πληροφοριών που μπορούν να βοηθήσουν τις προσπάθειες ενός attacker να αναπτύξει περισσότερα exploits. Ένας attacker μπορεί να δημιουργήσει input arguments για να εκθέσει εσωτερικές δομές και μηχανισμούς της εφαρμογής. Τόσο το κείμενο του μηνύματος εξαίρεσης όσο και ο τύπος μιας εξαίρεσης μπορούν να διαρρεύσουν πληροφορίες. Για παράδειγμα, το μήνυμα FileNotFoundException αποκαλύπτει πληροφορίες σχετικά με τη διάταξη του συστήματος αρχείων και ο τύπος εξαίρεσης αποκαλύπτει την απουσία του ζητούμενου αρχείου.

Όνομα Εξαίρεσης	**Περιγραφή Κινδύνου ή διαρροή πληροφορίας**
java.io.FileNotFoundException	Υποκείμενη δομή συστήματος αρχείων, enumeration ονομάτων χρήστη.
java.sql.SQLException	Δομή βάσης δεδομένων, enumeration ονομάτων χρήστη.
java.net.BindException	Enumeration ανοιχτών θυρών όταν ο μη αξιόπιστος client μπορεί να επιλέξει θύρα server.
java.util.ConcurrentModificationException	Μπορεί να παρέχει πληροφορίες σχετικά με μη ασφαλή κώδικα thread.
javax.naming.InsufficientResourcesException	Ανεπαρκείς πόροι server (μπορεί να βοηθήσει το DoS Attack)
java.util.MissingResourceException	Enumeration πόρων
java.util.jar.JarException	Υποκείμενη δομή συστήματος αρχείων
java.security.acl.NotOwnerException	Owner Enumeration
java.lang.OutOfMemoryError	Dos Attack

[71] "ERR00-J. Do not suppress or ignore checked ... - Confluence." https://wiki.sei.cmu.edu/confluence/display/java/ERR00-J.+Do+not+suppress+or+ignore+checked+exceptions. Πρόσβαση στις 25 Οκτ. 2020.

java.lang.StackOverflowError	Dos Attack

3.5.2.2.1 Παράδειγμα κώδικα που δεν συμμορφώνεται

Αυτό το μη συμμορφούμενο παράδειγμα κώδικα καταγράφει την εξαίρεση και κάνει throw μια custom εξαίρεση που κάνει wrap το FileNotFoundException:

```
class SecurityIOException extends IOException {/* ... */};

try {
  FileInputStream fis =
      new FileInputStream(System.getenv("APPDATA") + args[0]);
} catch (FileNotFoundException e) {
  // Log the exception
  throw new SecurityIOException();
}
```

Αυτό το πρόγραμμα αποκαλύπτει ότι δεν είναι δυνατή η ανάγνωση του καθορισμένου αρχείου. Πιο συγκεκριμένα, το πρόγραμμα αντιδρά διαφορετικά σε ανύπαρκτες διαδρομές αρχείων από ό, τι σε έγκυρες, και ένας εισβολέας μπορεί να συμπεράνει ευαίσθητες πληροφορίες σχετικά με το σύστημα αρχείων από τη συμπεριφορά αυτού του προγράμματος. Η αποτυχία περιορισμού της εισόδου χρήστη αφήνει το σύστημα ευάλωτο σε μια επίθεση brute-force στην οποία ο εισβολέας ανακαλύπτει έγκυρα ονόματα αρχείων κάνοντας requests που καλύπτουν συλλογικά το χώρο των πιθανών ονομάτων αρχείων. Τα ονόματα αρχείων που αναγκάζουν το πρόγραμμα να επιστρέψει την sanitized εξαίρεση υποδεικνύουν ανύπαρκτα αρχεία, ενώ τα ονόματα αρχείων που δεν επιστρέφουν εξαιρέσεις αποκαλύπτουν υπάρχοντα αρχεία.

3.5.2.2.2 Συμβατή λύση

Αυτή η συμβατή λύση εφαρμόζει την πολιτική ότι μόνο τα αρχεία που βρίσκονται στο c: \ homepath μπορούν να ανοιχτούν από τον χρήστη και ότι ο χρήστης δεν επιτρέπεται να ανακαλύψει τίποτα για αρχεία εκτός αυτού του directory. Η λύση εκδίδει ένα σύντομο μήνυμα σφάλματος όταν το αρχείο δεν μπορεί να ανοίξει ή το αρχείο δεν βρίσκεται στον κατάλληλο κατάλογο. Τυχόν πληροφορίες σχετικά με αρχεία εκτός του c: \ homepath είναι κρυμμένες.

Η συμβατή λύση χρησιμοποιεί επίσης τη μέθοδο File.getCanonicalFile() για να κανονικοποιήσει το αρχείο για να απλοποιήσει επόμενες συγκρίσεις ονομάτων διαδρομής (βλ. FIO16-J. Canonicalize ονόματα διαδρομών πριν από την επικύρωσή τους για περισσότερες πληροφορίες).

```
class ExceptionExample {
  public static void main(String[] args) {

    File file = null;
    try {
      file = new File(System.getenv("APPDATA") +
             args[0]).getCanonicalFile();
      if (!file.getPath().startsWith("c:\\homepath")) {
        System.out.println("Invalid file");
        return;
      }
    } catch (IOException x) {
      System.out.println("Invalid file");
```

```
      return;
    }

    try {
      FileInputStream fis = new FileInputStream(file);
    } catch (FileNotFoundException x) {
      System.out.println("Invalid file");
      return;
    }
  }
}
```

3.5.2.2.3 Risk Assessment

Εξαιρέσεις ενδέχεται να αποκαλύψουν ακούσια ευαίσθητες πληροφορίες, εκτός εάν ληφθεί μέριμνα για τον περιορισμό της αποκάλυψης πληροφοριών.

Κανόνας	**Severity**	**Πιθανότητα**	**Κόστος Ανάκαμψης**	**Προτεραιότητα**	**Επίπεδο**
ERR01-J	Medium	Probable	High	P4	L3

3.5.2.2.4 Αυτοματοποιημένη ανίχνευση

Εργαλείο	**Έκδοση**	**Checker**	**Περιγραφή**
Parasoft Jtest	10.3	**SECURITY.WSC.ACPST, SERVLET.CETS, SECURITY.ESD.ACW**	Ενσωματωμένο
SonarQube	6.7	**S1989**	Οι εξαιρέσεις δεν πρέπει γίνονται throw από τις μεθόδους servlet.

3.5.2.2.5 Σχετικές Ευπάθειες

Το CVE-2009-2897 περιγράφει διάφορες Cross-Site-Scripting επιθέσεις (XSS) σε διάφορες εκδόσεις του SpringSource Hyperic HQ. Αυτές οι ευπάθειες επιτρέπουν σε απομακρυσμένους εισβολείς να εισάγουν αυθαίρετο web script ή HTML μέσω μη έγκυρων τιμών για αριθμητικές παραμέτρους. Αποδεικνύονται από μια εξαίρεση java.lang.NumberFormatException που προκύπτει από την εισαγωγή πολλών μη έγκυρων αριθμητικών παραμέτρων στη διεπαφή ιστού.

Το CVE-2015-2080 περιγράφει μια ευπάθεια στον διακομιστή Ιστού Jetty, εκδόσεις 9.2.3 έως 9.2.8, όπου ένας παράνομος χαρακτήρας που μεταβιβάζεται σε ένα HTML request προκαλεί τον server να ανταποκριθεί με ένα μήνυμα σφάλματος που περιέχει το κείμενο με τον παράνομο χαρακτήρα. Αλλά αυτό το μήνυμα σφάλματος μπορεί επίσης να περιέχει ευαίσθητες πληροφορίες, όπως cookie από προηγούμενα web requests.

3.5.2.2.6 Σχετικές Οδηγίες

SEI CERT C++ Coding Standard	VOID ERR12-CPP. Do not allow exceptions to transmit sensitive information
MITRE CWE	CWE-209, Information Exposure through an Error Message CWE-497, Exposure of System Data to an Unauthorized Control Sphere CWE-600, Uncaught Exception in Servlet

[72]

3.5.2.3 Αποτροπή εξαιρέσεων κατα τη διάρκεια καταγραφής (logging)

Οι εξαιρέσεις που γίνονται throw, κατά τη διάρκεια της καταγραφής, μπορούν να αποτρέψουν την επιτυχή καταγραφή εκτός εάν ληφθεί ιδιαίτερη προσοχή. Η αποτυχία να ληφθούν υπόψη εξαιρέσεις κατά τη διαδικασία καταγραφής μπορεί να προκαλέσει ευπάθειες ασφαλείας, όπως το να επιτρέψετε σε έναν εισβολέα να αποκρύψει κρίσιμες εξαιρέσεις ασφαλείας, αποτρέποντάς τους να καταγραφούν. Κατά συνέπεια, τα προγράμματα πρέπει να διασφαλίζουν ότι η καταγραφή δεδομένων συνεχίζει να λειτουργεί σωστά ακόμα και όταν υπάρχουν εξαιρέσεις κατά τη διαδικασία καταγραφής.

3.5.2.3.1 Παράδειγμα κώδικα που δεν συμμορφώνεται

Αυτό το μη συμμορφούμενο παράδειγμα κώδικα γράφει μια κρίσιμη εξαίρεση ασφαλείας στην τυπική ροή σφαλμάτων:

```
try {
  // ...
} catch (SecurityException se) {
  System.err.println(se);
  // Recover from exception
}
```

Η σύνταξη τέτοιων εξαιρέσεων στην τυπική ροή σφαλμάτων είναι ανεπαρκής για σκοπούς καταγραφής. Πρώτον, η τυπική ροή σφαλμάτων μπορεί να εξαντληθεί ή να κλείσει, αποτρέποντας την καταγραφή των επακόλουθων εξαιρέσεων. Δεύτερον, το επίπεδο εμπιστοσύνης της τυπικής ροής σφαλμάτων μπορεί να είναι ανεπαρκές για την καταγραφή ορισμένων εξαιρέσεων ή σφαλμάτων που είναι κρίσιμα για την ασφάλεια χωρίς διαρροή ευαίσθητων πληροφοριών. Εάν προκύψει σφάλμα I / O κατά τη σύνταξη της εξαίρεσης ασφαλείας, το μπλοκ catch θα ρίξει μια IOException και η κρίσιμη εξαίρεση ασφαλείας θα χαθεί. Τέλος, ένας εισβολέας μπορεί να συγκαλύψει την εξαίρεση έτσι ώστε να συμβεί με πολλές άλλες αβλαβείς εξαιρέσεις.

Η χρήση του Console.printf(), System.out.print*() ή Throwable.printStackTrace() για την εξαγωγή μιας εξαίρεσης ασφαλείας συνιστά επίσης παραβίαση αυτού του κανόνα.

---

[72] "ERR01-J. Do not allow exceptions to expose ... - Confluence." https://wiki.sei.cmu.edu/confluence/display/java/ERR01-J.+Do+not+allow+exceptions+to+expose+sensitive+information. Πρόσβαση στις 25 Οκτ. 2020.

3.5.2.3.2 Συμβατή λύση

Η παρακαω συμβατή λύση χρησιμοποιεί το java.util.logging.Logger, το προεπιλεγμένο API καταγραφής που παρέχεται από το JDK 1.4 και μετά. Επιτρέπεται επίσης η χρήση άλλων συμβατών μηχανισμών καταγραφής, όπως το log4j.

```
try {
  // ...
} catch(SecurityException se) {
  logger.log(Level.SEVERE, se);
  // Recover from exception
}
```

Συνήθως, απαιτείται μόνο ένας logger για ολόκληρο το πρόγραμμα.

3.5.2.3.3 Risk Assessment

Οι εξαιρέσεις που δημιουργούνται κατά την καταγραφή δεδομένων μπορούν να προκαλέσουν απώλεια δεδομένων και να κρύψουν προβλήματα ασφαλείας.

Κανόνας	**Severity**	**Πιθανότητα**	**Κόστος Ανάκαμψης**	**Προτεραιότητα**	**Επίπεδο**
ERR02-J	Medium	Likely	High	P6	L2

3.5.2.3.4 Αυτοματοποιημένη ανίχνευση

Εργαλείο	**Έκδοση**	**Checker**	**Περιγραφή**
SonarQube	6.7	**S106**	Οι τυπικές έξοδοι δεν πρέπει να χρησιμοποιούνται απευθείας για καταγραφή.

3.5.2.3.5 Σχετικές Ευπάθειες

Το HARMONY-5981 περιγράφει μια ευπάθεια στην εφαρμογή HARMONY της Java. Σε αυτήν την εφαρμογή, η κλάση FileHandler μπορεί να λάβει μηνύματα καταγραφής, αλλά εάν ένα thread κλείσει το σχετικό αρχείο, ένα δεύτερο thread θα κάνει throw μια εξαίρεση όταν προσπαθεί να κάνει log ένα μήνυμα.

[73]

3.5.2.4 Επαναφορά προηγούμενης κατάστασης αντικειμένου σε αποτυχία μεθόδου

Τα αντικείμενα γενικά, και ειδικά τα κρίσιμα για την ασφάλεια αντικείμενα, πρέπει να διατηρούνται σε σταθερή κατάσταση ακόμη και όταν προκύπτουν εξαιρετικές συνθήκες. Οι κοινές τεχνικές για τη διατήρηση της συνοχής των αντικειμένων περιλαμβάνουν:

[73] "ERR02-J. Prevent exceptions while logging data - Confluence ...." https://wiki.sei.cmu.edu/confluence/display/java/ERR02-J.+Prevent+exceptions+while+logging+data. Πρόσβαση στις 25 Οκτ. 2020.

❏ Αξιολόγηση Εισόδου (για παράδειγμα στα arguments μεθόδου).

❏ Αναδιάταξη λογικής έτσι ώστε ο κώδικας που μπορεί να οδηγήσει στην εξαιρετική συνθήκη να εκτελείται πριν από την τροποποίηση του αντικειμένου.

❏ Χρήση επαναφοράς σε περίπτωση αποτυχίας.

❏ Εκτέλεση απαιτούμενων λειτουργιών σε προσωρινό αντίγραφο του αντικειμένου και πραγματοποίηση αλλαγών στο αρχικό αντικείμενο μόνο μετά την επιτυχή ολοκλήρωσή τους.

❏ Αποφυγή της τροποποίησης του αντικειμένου.

3.5.2.4.1 Παράδειγμα κώδικα που δεν συμμορφώνεται

Αυτό το μη συμμορφούμενο παράδειγμα κώδικα δείχνει μια κλάση Dimensions που περιέχει τρία εσωτερικά attributes: το μήκος, το πλάτος και το ύψος ενός ορθογωνίου πλαισίου. Η μέθοδος getVolumePackage() έχει σχεδιαστεί για να επιστρέφει τον συνολικό όγκο που απαιτείται για τη συγκράτηση του κουτιού αφού ληφθεί υπόψη το υλικό συσκευασίας, το οποίο προσθέτει 2 μονάδες στις διαστάσεις κάθε πλευράς. Οι μη θετικές τιμές των διαστάσεων του κουτιού (εκτός του υλικού συσκευασίας) απορρίπτονται κατά την επικύρωση της εισαγωγής. Καμία διάσταση δεν μπορεί να είναι μεγαλύτερη από 10. Επίσης, το βάρος του αντικειμένου μεταφέρεται ως όρισμα και δεν μπορεί να υπερβαίνει τις 20 μονάδες.

Εάν το βάρος είναι πάνω από 20 μονάδες, προκαλεί ένα IllegalArgumentException, το οποίο παρεμποδίζεται από τον custom error reporter. Παρόλο που η λογική επαναφέρει την αρχική κατάσταση του αντικειμένου ελλείψει αυτής της εξαίρεσης, ο κώδικας επαναφοράς δεν μπορεί να εκτελεστεί σε περίπτωση εξαίρεσης. Κατά συνέπεια, οι επακόλουθες επικλήσεις του getVolumePackage() παράγουν λανθασμένα αποτελέσματα.

```
class Dimensions {
  private int length;
  private int width;
  private int height;
  static public final int PADDING = 2;
  static public final int MAX_DIMENSION = 10;

  public Dimensions(int length, int width, int height) {
    this.length = length;
    this.width = width;
    this.height = height;
  }

  protected int getVolumePackage(int weight) {
    length += PADDING;
    width  += PADDING;
    height += PADDING;
    try {
      if (length <= PADDING || width <= PADDING || height <= PADDING ||
        length > MAX_DIMENSION + PADDING || width > MAX_DIMENSION + PADDING ||
        height > MAX_DIMENSION + PADDING || weight <= 0 || weight > 20) {
        throw new IllegalArgumentException();
      }

      int volume = length * width * height;
      length -= PADDING; width -= PADDING; height -= PADDING; // Revert
```

```
      return volume;
    } catch (Throwable t) {
      MyExceptionReporter mer = new MyExceptionReporter();
      mer.report(t); // Sanitize
      return -1; // Non-positive error code
    }
  }

  public static void main(String[] args) {
    Dimensions d = new Dimensions(8, 8, 8);
    System.out.println(d.getVolumePackage(21)); // Prints -1 (error)
    System.out.println(d.getVolumePackage(19));
    // Prints 1728 (12x12x12) instead of 1000 (10x10x10)
  }
}
```

3.5.2.4.2 Συμβατή λύση

Rollback

Αυτή η συμβατή λύση αντικαθιστά το μπλοκ catch στη μέθοδο getVolumePackage() με κώδικα που επαναφέρει την προηγούμενη κατάσταση αντικειμένου σε περίπτωση εξαίρεσης:

```
// ...

} catch (Throwable t) {
  MyExceptionReporter mer = new MyExceptionReporter();
  mer.report(t); // Sanitize
  length -= PADDING; width -= PADDING; height -= PADDING; // Revert
  return -1;
}
```

finally Clause

Αυτή η συμβατή λύση χρησιμοποιεί ένα finally clause για την εκτέλεση επαναφοράς, διασφαλίζοντας ότι η επαναφορά πραγματοποιείται είτε παρουσιάζεται σφάλμα είτε όχι:

```
protected int getVolumePackage(int weight) {
  length += PADDING;
  width  += PADDING;
  height += PADDING;
  try {
    if (length <= PADDING || width <= PADDING || height <= PADDING ||
      length > MAX_DIMENSION + PADDING ||
      width > MAX_DIMENSION + PADDING ||
      height > MAX_DIMENSION + PADDING ||
      weight <= 0 || weight > 20) {
      throw new IllegalArgumentException();
    }

    int volume = length * width * height;
    return volume;
  } catch (Throwable t) {
    MyExceptionReporter mer = new MyExceptionReporter();
```

```
    mer.report(t); // Sanitize
    return -1; // Non-positive error code
  } finally {
    // Revert
    length -= PADDING; width -= PADDING; height -= PADDING;
  }
}
```

Αξιολόγηση Εισόδου

Αυτή η συμβατή λύση βελτιώνεται από την προηγούμενη λύση εκτελώντας επικύρωση εισόδου πριν τροποποιηθεί η κατάσταση του αντικειμένου. Να σημειωθεί ότι το try μπλοκ περιέχει μόνο εκείνα τα statements που θα μπορούσαν να κάνουν throw την εξαίρεση. Όλα τα υπόλοιπα έχουν μετακινηθεί εκτός του try block.

```
protected int getVolumePackage(int weight) {
  try {
    if (length <= 0 || width <= 0 || height <= 0 ||
        length > MAX_DIMENSION || width > MAX_DIMENSION || height > MAX_DIMENSION ||
        weight <= 0 || weight > 20) {
      throw new IllegalArgumentException(); // Validate first
    }
  } catch (Throwable t) { MyExceptionReporter mer = new MyExceptionReporter();
    mer.report(t); // Sanitize
    return -1;
  }

  length += PADDING;
  width  += PADDING;
  height += PADDING;

  int volume = length * width * height;
  length -= PADDING; width -= PADDING; height -= PADDING;
  return volume;
}
```

Μη τροποποιημενο Object

Αυτή η συμβατή λύση αποφεύγει την ανάγκη τροποποίησης του αντικειμένου. Η κατάσταση του αντικειμένου δεν μπορεί να γίνει inconsistent και συνεπώς η επαναφορά είναι περιττή. Αυτή η προσέγγιση προτιμάται από λύσεις που τροποποιούν το αντικείμενο, αλλά ενδέχεται να μην είναι εφικτές για σύνθετο κώδικα.

```
protected int getVolumePackage(int weight) {
  try {
    if (length <= 0 || width <= 0 || height <= 0 ||
        length > MAX_DIMENSION || width > MAX_DIMENSION ||
        height > MAX_DIMENSION || weight <= 0 || weight > 20) {
      throw new IllegalArgumentException(); // Validate first
    }
  } catch (Throwable t) {
    MyExceptionReporter mer = new MyExceptionReporter();
    mer.report(t); // Sanitize
    return -1;
  }
```

```
  int volume = (length + PADDING) * (width + PADDING) *
               (height + PADDING);
  return volume;
}
```

3.5.2.4.3 Risk Assessment

Η αποτυχία επαναφοράς της κατάστασης προηγούμενου αντικειμένου κατά την αποτυχία της μεθόδου μπορεί να αφήσει το αντικείμενο σε inconsistent κατάσταση και μπορεί να παραβιάσει τις απαιτούμενες state invariants.

Κανόνας	Severity	Πιθανότητα	Κόστος Ανάκαμψης	Προτεραιότητα	Επίπεδο
ERR03-J	Low	Probable	High	**P2**	**L3**

3.5.2.4.4 Σχετικές Ευπάθειες

Το CVE-2008-0002 περιγράφει μια ευπάθεια σε διάφορες εκδόσεις του Apache Tomcat. Εάν προκύψει εξαίρεση κατά την επεξεργασία παραμέτρων, το πρόγραμμα μπορεί να παραμείνει στο πλαίσιο λανθασμένου αιτήματος, το οποίο θα μπορούσε να επιτρέψει σε απομακρυσμένους attackers να λάβουν ευαίσθητες πληροφορίες. Μια εξαίρεση μπορεί να προκληθεί αποσυνδέοντας από το Tomcat κατά τη διάρκεια αυτής της επεξεργασίας.

3.5.2.4.5 Σχετικές Οδηγίες

MITRE CWE	CWE-460, Improper Cleanup on Thrown Exception

[74]

3.5.2.5 Μη ολοκλήρωση του finally block απότομα

Ποτέ δεν πρέπει να χρησιμοποιούνται τα statements return, break, continue ή throw σε finally μπλοκ. Όταν η εκτέλεση του προγράμματος εισέρχεται σε ένα try μπλοκ που έχει finally μπλοκ, το finally μπλοκ εκτελείται πάντα ανεξάρτητα από το εάν το try μπλοκ (ή τυχόν συσχετισμένα catch μπλοκ) εκτελείται στην κανονική ολοκλήρωση. Τα statements που προκαλούν την απότομη ολοκλήρωση του μπλοκ προκαλούν επίσης την απότομη ολοκλήρωση του try μπλοκ και κατά συνέπεια καταστέλλει κάθε exception που γίνεται throw από τα try ή τα catch μπλοκ.

3.5.2.5.1 Παράδειγμα κώδικα που δεν συμμορφώνεται

Σε αυτό το μη συμμορφούμενο παράδειγμα κώδικα, το finally μπλοκ ολοκληρώνεται απότομα λόγω ενός return statement στο μπλοκ:

[74] "ERR03-J. Restore prior object state on method ... - Confluence." https://wiki.sei.cmu.edu/confluence/display/java/ERR03-J.+Restore+prior+object+state+on+method+failure. Πρόσβαση στις 25 Οκτ. 2020.

```
class TryFinally {
  private static boolean doLogic() {
    try {
      throw new IllegalStateException();
    } finally {
      System.out.println("logic done");
      return true;
    }
  }
}
```

Το IllegalStateException καταστέλλεται από την απότομη ολοκλήρωση του finally μπλοκ που προκαλείται από το return statement.

3.5.2.5.2 Συμβατή λύση

Αυτή η συμβατή λύση αφαιρεί το return statement από το finally block:

```
class TryFinally {
  private static boolean doLogic() {
    try {
      throw new IllegalStateException();
    } finally {
      System.out.println("logic done");
    }
    // Any return statements must go here;
    // applicable only when exception is thrown conditionally
  }
}
```

3.5.2.5.3 Risk Assessment

Η απότομη ολοκλήρωση ενός finally μπλοκ αποκρύπτει τυχόν εξαιρέσεις που γίνονται throw στο σχετικό try ή catch μπλοκ.

Κανόνας	**Severity**	**Πιθανότητα**	**Κόστος Ανάκαμψης**	**Προτεραιότητα**	**Επίπεδο**
ERR04-J	Low	Probable	Medium	**P4**	**L3**

3.5.2.5.4 Αυτοματοποιημένη ανίχνευση

Εργαλείο	**Έκδοση**	**Checker**	**Περιγραφή**
CodeSonar	5.4p0	**PMD.Strict-Exceptions.DoNotThrowExceptionInFinallyE**	Να μην γίνεται throw της εξαίρεσης στο finally block
Coverity	7.5	**PW.ABNORMAL_TERMINATION_ OF_FINALLY_BLOCK**	Ενσωματωμένο

Parasoft Jtest	10.3	**PB.CUB.ARCF, PB.CUB.ATSF**	Ενσωματωμένο
SonarQube	6.7	**S1143**	Τα jump statements δεν πρέπει να εμφανίζονται σε finally μπλοκ

3.5.2.5.5 Σχετικές Οδηγίες

MITRE CWE	CWE-459, Incomplete Cleanup CWE-584, Return Inside finally Block

[75]

3.5.2.6 Αποτροπή checked exceptions να κάνουν escape από finally block

Οι μέθοδοι που καλούνται μέσα από ένα finally μπλοκ μπορούν να κάνουν throw μια εξαίρεση. Η αποτυχία εντοπισμού και διαχείρισης τέτοιων εξαιρέσεων οδηγεί στον απότομο τερματισμό ολόκληρου του try μπλο. Ο απότομος τερματισμός προκαλεί απώλεια οποιασδήποτε εξαίρεσης που γίνεται throw στο try μπλο, εμποδίζοντας κάθε πιθανή μέθοδο ανάκτησης να διαχειριστεί αυτό το συγκεκριμένο πρόβλημα. Επιπλέον, η μεταβίβαση ελέγχου που σχετίζεται με την εξαίρεση ενδέχεται να αποτρέψει την εκτέλεση τυχόν εκφράσεων ή δηλώσεων που προκύπτουν μετά το σημείο στο finally μπλοκ από το οποίο η εξαίρεση γίνεται throw. Κατά συνέπεια, τα προγράμματα πρέπει να χειρίζονται κατάλληλα τις checked εξαιρέσεις που απορρίπτονται από ένα finally μπλοκ. Επίσης το να επιτρέπονται οι checked εξαιρέσεις να ξεφεύγουν από το finally block παραβιάζει επίσης τη 3.5.2.5 Μη ολοκλήρωση του finally block απότομα.

3.5.2.6.1 Παράδειγμα κώδικα που δεν συμμορφώνεται

Αυτό το μη συμμορφούμενο παράδειγμα κώδικα περιέχει ένα finally μπλοκ που κλείνει το reader object. Ο προγραμματιστής υποθέτει εσφαλμένα ότι οι δηλώσεις στο finally μπλοκ δεν μπορούν να κάνουν throw εξαιρέσεις και κατά συνέπεια αποτυγχάνει να χειριστεί κατάλληλα οποιαδήποτε εξαίρεση που μπορεί να προκύψει.

```
public class Operation {
  public static void doOperation(String some_file) {
    // ... Code to check or set character encoding ...
    try {
      BufferedReader reader =
          new BufferedReader(new FileReader(some_file));
      try {
        // Do operations
      } finally {
        reader.close();
        // ... Other cleanup code ...
      }
    } catch (IOException x) {
      // Forward to handler
```

[75] **"ERR04-J. Do not complete abruptly from a finally ... - Confluence." https://wiki.sei.cmu.edu/confluence/display/java/ERR04-J.+Do+not+complete+abruptly+from+a+finally+block. Πρόσβαση στις 25 Οκτ. 2020.**

```
    }
  }
}
```

Η μέθοδος close() μπορεί να κάνει throw ένα IOException, το οποίο, εάν γίνει throw, θα αποτρέψει την εκτέλεση τυχόν μεταγενέστερων δηλώσεων cleanup. Αυτό το πρόβλημα δεν θα διαγνωστεί από τον μεταγλωττιστή επειδή τυχόν IOException θα γινόταν catch από το εξωτερικό catch μπλοκ. Επίσης, μια εξαίρεση που γίνεται throw από τη μέθοδο close() μπορεί να κάνει mask οποιαδήποτε εξαίρεση που γίνεται throw κατά την εκτέλεση του Do operations block, αποτρέποντας τη σωστή ανάκτηση.

3.5.2.6.2 Συμβατή λύση

Η Java SE 7 εισήγαγε μια δυνατότητα που ονομάζεται try-with-resources που μπορεί να κλείσει αυτόματα ορισμένους πόρους σε περίπτωση σφάλματος. Αυτή η συμβατή λύση χρησιμοποιεί try-with-resources για να κλείσει σωστά το αρχείο.

```
public class Operation {
  public static void doOperation(String some_file) {
    // ... Code to check or set character encoding ...
    try ( // try-with-resources
      BufferedReader reader =
          new BufferedReader(new FileReader(some_file))) {
      // Do operations
    } catch (IOException ex) {
      System.err.println("thrown exception: " + ex.toString());
      Throwable[] suppressed = ex.getSuppressed();
      for (int i = 0; i < suppressed.length; i++) {
        System.err.println("suppressed exception: "
            + suppressed[i].toString());
      }
      // Forward to handler
    }
  }

  public static void main(String[] args) {
    if (args.length < 1) {
      System.out.println("Please supply a path as an argument");
      return;
    }
    doOperation(args[0]);
  }
}
```

Όταν εμφανιστεί ένα IOException στο try μπλοκ της μεθόδου doOperation(), γίνεται catch από το catch μπλοκ και εκτυπώνεται ως thrown εξαίρεση. Επίσης περιλαμβάνονται εξαιρέσεις που προκύπτουν κατά τη δημιουργία του BufferedReader. Όταν συμβαίνει ένα IOException κατά το κλείσιμο του reader, αυτή η εξαίρεση γίνεται catch επίσης από το catch μπλοκ και εκτυπώνεται ως thrown εξαίρεση. Εάν τόσο το try μπλοκ όσο και το κλείσιμο του reader κάνει throw μια εξαίρεση IO, το catch clause κάνει catch και τις δύο εξαιρέσεις και εκτυπώνει την εξαίρεση του try μπλοκ ως thrown εξαίρεση. Η close εξαίρεση καταστέλλεται και εκτυπώνεται. Σε όλες τις περιπτώσεις, ο reader κλείνει με ασφάλεια.

3.5.2.6.3 Risk Assessment

Η αποτυχία χειρισμού εξαίρεσης σε finally μπλοκ μπορεί να έχει απρόσμενα αποτελέσματα.

Κανόνας	Severity	Πιθανότητα	Κόστος Ανάκαμψης	Προτεραιότητα	Επίπεδο
ERR05-J	Low	Unlikely	Medium	**P2**	**L3**

3.5.2.6.4 Αυτοματοποιημένη ανίχνευση

Εργαλείο	Έκδοση	Checker	Περιγραφή
Coverity	7.5	**PW.ABNORMAL_TERMINATION_OF_FINALLY_BLOCK**	Ενσωματωμένο
Parasoft Jtest	10.3	**PB.CUB.ARCF, PB.CUB.ATSF**	Ενσωματωμένο
SonarQube	6.7	**S1163**	Οι εξαιρέσεις δεν πρέπει να γίνονται throw σε finally blocks.

3.5.2.6.5 Σχετικές Οδηγίες

MITRE CWE	CWE-248, Uncaught Exception CWE-460, Improper Cleanup on Thrown Exception CWE-584, Return inside finally Block CWE-705, Incorrect Control Flow Scoping CWE-754, Improper Check for Unusual or Exceptional Conditions

[76]

3.5.2.7 Να μην γίνονται throw μη δηλωμένες checked εξαιρέσεις

Η Java απαιτεί ότι κάθε μέθοδος αντιμετωπίζει κάθε checked εξαίρεση που μπορεί να γίνει throw κατά την εκτέλεση της, είτε διαχειρίζοντας την εξαίρεση σε ένα try μπλοκ ή δηλώνοντας ότι η εξαίρεση μπορεί να εξαπλωθεί εκτός της μεθόδου (μέσω του throw clause). Δυστυχώς, υπάρχουν μερικές τεχνικές που επιτρέπουν σε μη δηλωμένες checked εξαίρεση να γίνουν throw κατά το χρόνο εκτέλεσης. Τέτοιες τεχνικές χάνουν την ικανότητα των caller μεθόδων να χρησιμοποιούν τα throw clauses για να προσδιορίσουν το πλήρες σύνολο checked εξαιρέσεων που θα μπορούσαν να διαδίδονται από μια invoked μέθοδο. Κατά συνέπεια, τέτοιες τεχνικές δεν πρέπει να χρησιμοποιούνται για το throw μη δηλωμένων checked εξαιρέσεων.

[76] "ERR05-J. Do not let checked exceptions escape ... - Confluence." https://wiki.sei.cmu.edu/confluence/display/java/ERR05-J.+Do+not+let+checked+exceptions+escape+from+a+finally+block. Πρόσβαση στις 26 Οκτ. 2020.

3.5.2.7.1 Παράδειγμα κώδικα που δεν συμμορφώνεται

Όταν ο προγραμματιστής επιθυμεί να κάνει catch και να χειριστεί τις πιθανές αδήλωτες checked εξαιρέσεις, ο μεταγλωττιστής αρνείται να πιστέψει ότι μπορούν να γίνουν throw στο συγκεκριμένο πλαίσιο. Αυτό το μη συμμορφούμενο παράδειγμα κώδικα επιχειρεί να κάνει catch μη δηλωμένες checked εξαιρέσεις που γίνονται throw από το Class.newInstance(). Κάνει catch την εξαίρεση και ελέγχει δυναμικά εάν η catched εξαίρεση είναι μια περίπτωση της πιθανής checked εξαίρεσης (προσεκτικά rethrowing όλες τις άλλες εξαιρέσεις).

```
public static void main(String[] args) {
  try {
    NewInstance.undeclaredThrow(
        new IOException("Any checked exception"));
  } catch (Throwable e) {
    if (e instanceof IOException) {
      System.out.println("IOException occurred");
    } else if (e instanceof RuntimeException) {
      throw (RuntimeException) e;
    } else {
      // Forward to handler
    }
  }
}
```

3.5.2.7.2 Συμβατή λύση

Αυτή η συμβατή λύση χρησιμοποιεί το java.lang.reflect.Constructor.newInstance() και όχι Class.newInstance (). Η μέθοδος Constructor.newInstance () περιτυλίγει τυχόν εξαιρέσεις που γίνονται throw από τον constructor σε μια checked εξαίρεση που ονομάζεται InvocationTargetException.

```
public static synchronized void undeclaredThrow(Throwable throwable) {
  // These exceptions should not be passed
  if (throwable instanceof IllegalAccessException ||
      throwable instanceof InstantiationException) {
    // Unchecked, no declaration required
    throw new IllegalArgumentException();
  }

  NewInstance.throwable = throwable;
  try {
    Constructor constructor =
        NewInstance.class.getConstructor(new Class<?>[0]);
    constructor.newInstance();
  } catch (InstantiationException e) { /* Unreachable */
  } catch (IllegalAccessException e) { /* Unreachable */
  } catch (InvocationTargetException e) {
    System.out.println("Exception thrown: "
        + e.getCause().toString());
  } finally { // Avoid memory leak
    NewInstance.throwable = null;
  }
}
```

3.5.2.7.3 Risk Assessment

Η αποτυχία χειρισμού εξαίρεσης σε finally μπλοκ μπορεί να έχει απρόσμενα αποτελέσματα.

Κανόνας	Severity	Πιθανότητα	Κόστος Ανάκαμψης	Προτεραιότητα	Επίπεδο
ERR06-J	Low	Unlikely	High	**P1**	**L3**

3.5.2.7.4 Σχετικές Οδηγίες

MITRE CWE	CWE-703, Improper Check or Handling of Exceptional Conditions CWE-248, Uncaught Exception

[77]

3.5.2.7 Να μην γίνονται throw RunTimeException, Exception, ή Throwable

Οι μέθοδοι δεν πρέπει να κάνουν throw RuntimeException, Exception ή Throwable. Ο χειρισμός αυτών των εξαιρέσεων απαιτεί το catching του RuntimeException. Δεν πρέπει να γίνεται catch του NullPointerException ή οποιονδήποτε από τους προγόνους του. Επιπλέον, το throw ενός RuntimeException μπορεί να οδηγήσει σε λάθη. Για παράδειγμα, ένας caller δεν μπορεί να εξετάσει την εξαίρεση και να προσδιορίσει γιατί έγινε throw και συνεπώς δεν μπορεί να επιχειρήσει ανάκτηση.

Οι μέθοδοι μπορούν να κάνουν throw μια συγκεκριμένη εξαίρεση υποκατηγορίας της Exception ή της RuntimeException. Να σημειωθεί ότι επιτρέπεται η κατασκευή μιας κατηγορίας εξαίρεσης ειδικά για μια δήλωση μεμονωμένου throw.

3.5.2.7.1 Παράδειγμα κώδικα που δεν συμμορφώνεται

Η μέθοδος isCapitalized() σε αυτό το μη συμμορφούμενο παράδειγμα κώδικα δέχεται μια συμβολοσειρά και επιστρέφει true όταν η συμβολοσειρά αποτελείται από ένα κεφαλαίο γράμμα ακολουθούμενο από πεζά γράμματα. Η μέθοδος κάνει επίσης throw ένα RuntimeException όταν περαστεί ένα όρισμα συμβολοσειράς null.

```
boolean isCapitalized(String s) {
  if (s == null) {
    throw new RuntimeException("Null String");
  }
  if (s.equals("")) {
    return true;
  }
  String first = s.substring(0, 1);
  String rest = s.substring(1);
  return (first.equals(first.toUpperCase()) &&
```

[77] **"ERR06-J. Do not throw undeclared checked ... - Confluence." https://wiki.sei.cmu.edu/confluence/display/java/ERR06-J.+Do+not+throw+undeclared+checked+exceptions. Πρόσβαση στις 26 Οκτ. 2020.**

```
          rest.equals(rest.toLowerCase()));
}
```

3.5.2.7.2 Συμβατή λύση

Αυτή η συμβατή λύση κάνει throw το NullPointerException για να δηλώσει τη συγκεκριμένη exceptional κατάσταση

```
boolean isCapitalized(String s) {
  if (s == null) {
    throw new NullPointerException();
  }
  if (s.equals("")) {
    return true;
  }
  String first = s.substring(0, 1);
  String rest = s.substring(1);
  return (first.equals(first.toUpperCase()) &&
          rest.equals(rest.toLowerCase()));
}
```

3.5.2.7.3 Risk Assessment

Η αποτυχία χειρισμού εξαίρεσης σε finally μπλοκ μπορεί να έχει απρόσμενα αποτελέσματα.

Κανόνας	Severity	Πιθανότητα	Κόστος Ανάκαμψης	Προτεραιότητα	Επίπεδο
ERR07-J	Low	Likely	Medium	P6	L2

3.5.2.7.4 Αυτοματοποιημένη ανίχνευση

Εργαλείο	Έκδοση	Checker	Περιγραφή
Parasoft Jtest	10.3	**CODSTD.BP.NTX, EXCEPT.NTERR**	Ενσωματωμένο
SonarQube	6.7	**S112**	Οι γενικές εξαιρέσεις δεν πρέπει ποτέ να γίνονται throw.

3.5.2.7.5 Σχετικές Οδηγίες

MITRE CWE	CWE-397, Declaration of Throws for Generic Exception

[78]

3.5.2.8 Δεν πρέπει να γίνεται catch της NullPointerException ή ενός προγόνου της

Τα προγράμματα δεν πρέπει να κάνουν catch το java.lang.NullPointerException. Μια εξαίρεση NullPointerException που γίνεται throw κατά το χρόνο εκτέλεσης υποδεικνύει την ύπαρξη υποκείμενης dereference δείκτη null που πρέπει να διορθωθεί στον κώδικα. Ο χειρισμός της υποκείμενης null pointer dereference κάνοντας catch τη NullPointerException, αντί να διορθωθεί το υποκείμενο πρόβλημα είναι ακατάλληλο για διάφορους λόγους. Πρώτον, το catching του NullPointerException προσθέτει σημαντικό performance overhead από την απλή προσθήκη των απαραίτητων null ελέγχων. Δεύτερον, όταν πολλές εκφράσεις σε ένα try μπλοκ είναι ικανές να κάνουν throw ένα NullPointerException, είναι δύσκολο ή αδύνατο να προσδιοριστεί ποια έκφραση είναι υπεύθυνη για την εξαίρεση, επειδή το catch μπλοκ NullPointerException χειρίζεται οποιοδήποτε NullPointerException που γίνεται throw από οποιαδήποτε τοποθεσία στο try μπλοκ. Τρίτον, τα προγράμματα σπάνια παραμένουν σε αναμενόμενη και χρησιμοποιήσιμη κατάσταση μετά το NullPointerException. Οι προσπάθειες για συνέχιση της εκτέλεσης μετά την πρώτη σύλληψη και καταγραφή (ή χειρότερα, καταστολή) η εξαίρεση σπάνια επιτυγχάνεται.

3.5.2.8.1 Παράδειγμα κώδικα που δεν συμμορφώνεται

Αυτό το μη συμμορφούμενο παράδειγμα κώδικα ορίζει μια μέθοδο isName() που παίρνει ένα όρισμα String και επιστρέφει true, αν η δεδομένη συμβολοσειρά είναι έγκυρο όνομα. Ένα έγκυρο όνομα ορίζεται ως δύο λέξεις με κεφαλαία διαχωρισμένα με ένα ή περισσότερα κενά. Αντί να ελέγχει αν η δεδομένη συμβολοσειρά είναι μηδενική, η μέθοδος κάνει catch το NullPointerException και επιστρέφει false.

```
boolean isName(String s) {
  try {
    String names[] = s.split(" ");

    if (names.length != 2) {
      return false;
    }
    return (isCapitalized(names[0]) && isCapitalized(names[1]));
  } catch (NullPointerException e) {
    return false;
  }
}
```

3.5.2.8.2 Συμβατή λύση

Αυτή η συμβατή λύση ελέγχει ρητά το όρισμα String για null αντί να κάνει catch το NullPointerException:

```
boolean isName(String s) {
  if (s == null) {
    return false;
  }
  String names[] = s.split(" ");
  if (names.length != 2) {
```

[78] **"ERR07-J. Do not throw RuntimeException ... - Confluence." https://wiki.sei.cmu.edu/confluence/display/java/ERR07-J.+Do+not+throw+RuntimeException%2C+Exception%2C+or+Throwable. Πρόσβαση στις 26 Οκτ. 2020.**

```
    return false;
  }
  return (isCapitalized(names[0]) && isCapitalized(names[1]));
}
```

3.5.2.8.3 Risk Assessment

Το Catching του NullPointerException μπορεί να κάνει mask μια υποκείμενη null dereference, να υποβαθμίσει την απόδοση της εφαρμογής και να οδηγήσει σε κώδικα που είναι δύσκολο να κατανοηθεί και να διατηρηθεί. Ομοίως, το catching των RuntimeException, Exception ή Throwable μπορεί να παγιδεύσει ακούσια άλλους τύπους εξαιρέσεων και να τους αποτρέψει από το σωστό χειρισμό.

Κανόνας	**Severity**	**Πιθανότητα**	**Κόστος Ανάκαμψης**	**Προτεραιότητα**	**Επίπεδο**
ERR08-J	Medium	Likely	Medium	**P12**	**L1**

3.5.2.8.4 Αυτοματοποιημένη ανίχνευση

Εργαλείο	**Έκδοση**	**Checker**	**Περιγραφή**
CodeSonar	5.4p0	**PMD.Strict-Exceptions.AvoidCatchingThrowable**	Αποφυγή catching Throwable
Parasoft Jtest	10.3	**EXCEPT.NCNPE**	Ενσωματωμένο
SonarQube	6.7	**S1181** **S1696**	Τα Throwable, οι NullPointerException και τα Error δεν πρέπει να γίνονται catch.

[79]

3.5.2.9 Αποτροπή μη αξιόπιστου κώδικα να τερματίσει το JVM

Η επίκληση του System.exit() τερματίζει την Java Virtual Machine (JVM), τερματίζοντας κατά συνέπεια όλα τα τρέχοντα προγράμματα και threads. Αυτό μπορεί να οδηγήσει σε επιθέσεις άρνησης υπηρεσίας (DoS). Για παράδειγμα, μια κλήση στο System.exit() που είναι ενσωματωμένη στον κώδικα Java Server Pages (JSP) μπορεί να προκαλέσει τον τερματισμό ενός web server, εμποδίζοντας την περαιτέρω εξυπηρέτηση των χρηστών. Τα προγράμματα πρέπει να αποτρέπουν τόσο ακούσιες όσο και κακόβουλες κλήσεις στο System.exit(). Επιπλέον, τα προγράμματα θα πρέπει να εκτελούν τις απαραίτητες ενέργειες εκκαθάρισης όταν τερματίζονται βίαια (για παράδειγμα, χρησιμοποιώντας τον Windows Task Manager, την εντολή POSIX kill ή άλλους μηχανισμούς).

[79] "ERR08-J. Do not catch NullPointerException or any of its ...." https://wiki.sei.cmu.edu/confluence/display/java/ERR08-J.+Do+not+catch+NullPointerException+or+any+of+its+ancestors . Πρόσβαση στις 26 Οκτ. 2020.

3.5.2.9.1 Παράδειγμα κώδικα που δεν συμμορφώνεται

Αυτό το μη συμμορφούμενο παράδειγμα κώδικα χρησιμοποιεί τη System.exit() για να κλείσει δυναμικά το JVM και να τερματίσει τη διαδικασία που εκτελείται. Το πρόγραμμα στερείται διαχειριστή ασφαλείας. Κατά συνέπεια, δεν έχει τη δυνατότητα να ελέγξει εάν επιτρέπεται στον caller να καλέσει το System.exit().

```
public class InterceptExit {
  public static void main(String[] args) {
    // ...
    System.exit(1);  // Abrupt exit
    System.out.println("This never executes");
  }
}
```

3.5.2.9.2 Συμβατή λύση

Αυτή η συμβατή λύση εγκαθιστά έναν custom διαχειριστή ασφαλείας PasswordSecurityManager που κάνει override τη μέθοδο checkExit() που ορίζεται στην κλάση SecurityManager. Αυτό το override απαιτείται για να ενεργοποιηθεί η επίκληση του κώδικα εκκαθάρισης προτού επιτραπεί η έξοδος. Η προεπιλεγμένη μέθοδος checkExit() στην κλάση SecurityManager δεν διαθέτει αυτήν τη δυνατότητα.

```
class PasswordSecurityManager extends SecurityManager {
  private boolean isExitAllowedFlag;

  public PasswordSecurityManager(){
    super();
    isExitAllowedFlag = false;
  }

  public boolean isExitAllowed(){
    return isExitAllowedFlag;
  }

  @Override
  public void checkExit(int status) {
    if (!isExitAllowed()) {
      throw new SecurityException();
    }
    super.checkExit(status);
  }

  public void setExitAllowed(boolean f) {
    isExitAllowedFlag = f;
  }
}

public class InterceptExit {
  public static void main(String[] args) {
    PasswordSecurityManager secManager =
        new PasswordSecurityManager();
    System.setSecurityManager(secManager);
```

```
    try {
      // ...
      System.exit(1);  // Abrupt exit call
    } catch (Throwable x) {
      if (x instanceof SecurityException) {
        System.out.println("Intercepted System.exit()");
        // Log exception
      } else {
        // Forward to exception handler
      }
    }

    // ...
    secManager.setExitAllowed(true);  // Permit exit
    // System.exit() will work subsequently
    // ...
  }
}
```

Αυτή η εφαρμογή χρησιμοποιεί μια εσωτερική σημαία για να παρακολουθεί εάν επιτρέπεται η έξοδος. Η μέθοδος setExitAllowed() ορίζει αυτήν τη σημαία. Η μέθοδος checkExit() κάνει throw μία SecurityException όταν δεν έχει οριστεί η σημαία (δηλαδή, false). Επειδή αυτή η σημαία δεν έχει οριστεί αρχικά, η κανονική επεξεργασία εξαίρεσης παρακάμπτει την αρχική κλήση στο System.exit(). Το πρόγραμμα κάνει catch το SecurityException και εκτελεί υποχρεωτικές λειτουργίες καθαρισμού, συμπεριλαμβανομένης της καταγραφής της εξαίρεσης. Η μέθοδος System.exit() ενεργοποιείται μόνο μετά την ολοκλήρωση της εκκαθάρισης.

3.5.2.9.3 Risk Assessment

Η αποδοχή μη εξουσιοδοτημένων κλήσεων στη System.exit() μπορεί να οδηγήσει σε άρνηση υπηρεσιών DOS.

Κανόνας	Severity	Πιθανότητα	Κόστος Ανάκαμψης	Προτεραιότητα	Επίπεδο
ERR09-J	Low	Unlikely	Medium	**P2**	**L3**

3.5.2.9.4 Αυτοματοποιημένη ανίχνευση

Εργαλείο	Έκδοση	Checker	Περιγραφή
CodeSonar	5.4p0	**PMD.J2EE.DoNotCallSystemExit** **FB.BAD_PRACTICE.DM_EXIT**	Να μην καλείται η System.exit()
Coverity	7.5	**DC.CODING_STYLE** **FB.DM_EXIT**	Ενσωματωμένο
Parasoft Jtest	10.3	**CODSTA.BP.EXIT, SECURITY.EAB.JVM**	Ενσωματωμένο

SonarQube	6.7	**S1147**	Οι μέθοδοι εξόδου δεν πρέπει να καλούνται.

[80]

3.5.3 Διαχείριση Σφαλμάτων και JSF

3.5.3.1 Custom Σελίδα Σφάλματος

Όταν μία εφαρμογή τρέχει στο στάδιο ανάπτυξης και συναντάται ένα σφάλμα, το σφάλμα βοηθάει τον προγραμματιστή να καταλάβει τι δεν πάει καλά. Παρ'όλα αυτά δεν είναι σωστό ένας χρήστης να βλέπει αυτά τα σφάλματα, καθώς μπορεί να υπάρξει διαρροή σημαντικών πληροφοριών. Για να εμφανιστεί στο χρήστη το σωστό μήνυμα σφάλματος που δεν θα θέτει σε κίνδυνο την εφαρμογή, είναι καλό να δημιουργηθούν custom σελίδες σφάλματος, όπου θα αποκαλύπτουν στον χρήστη μόνο όσα πρέπει να μάθει. Για να γίνει αντικατάσταση της σελίδας σφάλματος πρέπει να χρησιμοποιηθεί το error page tag στο αρχείο web.xml, στο οποίο μπορεί να οριστεί μία εξαίρεση Java ή ένα HTTP error code. Έτσι σε περίπτωση που γίνει throw μια εξαίρεση η οποία γίνει match με τον τύπο που ορίστηκε στο web-xml exception-type ή παραχθεί error code από τον server που αντιστοιχίζεται στο error-code που συμπεριλήφθηκε στο web.xml, το JSF Framework θα το διαχειριστεί προωθώντας τον χρήστη στο επιθυμητό view που ορίστηκε από τον προγραμματιστή για τα αντίστοιχα σφάλματα ή εξαιρέσεις.

3.5.3.1.1 Deployment Descriptor

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5" metadata-complete="true">
        <error-page>
            <error-code>404</error-code>
            <location>/faces/error.xhtml</location>
        </error-page>
        <error-page>
            <error-code>500</error-code>
            <location>/faces/error.xhtml</location>
        </error-page>
        <error-page>
            <exception-type>java.lang.Exception</exception-type>
            <location>/faces/error.xhtml</location>
        </error-page>
        <context-param>
            <description>State saving method: 'client' or 'server'
                      (=default). See JSF Specification 2.5.2
            </description>
            <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
            <param-value>server</param-value>
        </context-param>
```

[80] **"ERR09-J. Do not allow untrusted code to terminate the JVM." https://wiki.sei.cmu.edu/confluence/display/java/ERR09-J.+Do+not+allow+untrusted+code+to+terminate+the+JVM. Πρόσβαση στις 26 Οκτ. 2020.**

```
        <context-param>
            <param-name>javax.faces.application.CONFIG_FILES</param-name>
            <param-value>/WEB-INF/faces-config.xml</param-value>
        </context-param>
        <servlet>
            <servlet-name>Faces Servlet</servlet-name>
            <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
            <load-on-startup>1</load-on-startup>
        </servlet>
            <servlet-mapping>
            <servlet-name>Faces Servlet</servlet-name>
            <url-pattern>/faces/*</url-pattern>
        </servlet-mapping>
        <servlet-mapping>
            <servlet-name>Faces Servlet</servlet-name>
            <url-pattern>*.xhtml</url-pattern>
        </servlet-mapping>
        <listener>
  <listener-class>com.sun.faces.config.ConfigureListener</listener-class>
        </listener>
</web-app>
```

Τα error codes που γίνονται handle από αυτό το web.xml είναι το **500** και το **400**. Οι εξαιρέσεις που γίνονται handle είναι η ρίζα των εξαιρέσεων που μπορούν να γίνουν throw από το java.lang.Exception. Όλες οι ορισμένες εξαιρέσεις και error codes πρέπει να γίνουν handle σε μια σελίδα σφάλματος **error.xhtml**.

3.5.3.1.2 Η σελίδα σφάλματος

error.xhtml

```
<html          xmlns="http://www.w3.org/1999/xhtml"          xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:h="http://java.sun.com/jsf/html" xmlns:f="http://java.sun.com/jsf/core">
        <h1>JavaBeat JSF 2.2 Examples</h1>
        <h2>JSF2 - Error Handling</h2>
        <br />
        <h:outputText value="Error Code: #{errorHandler.statusCode}"></h:outputText>
        <br />
        <h:outputText value="Error Desscription: #{errorHandler.message}"></h:outputText>
        <br />
        <h:outputText value="Exception Type: #{errorHandler.exceptionType}"></h:outputText>
        <br />
        <h:outputText value="Exception Calss: #{errorHandler.exception}"></h:outputText>
        <br />
        <h:outputText value="Request URI : #{errorHandler.requestURI}"></h:outputText>
</html>
```

3.5.3.1.3 ErrorHandler RequestScoped Bean

ErrorHandler.java

```
package net.javabeat.jsf.error;
import javax.faces.bean.ManagedBean;
```

```
import javax.faces.bean.RequestScoped;
import javax.faces.context.FacesContext;

@ManagedBean
@RequestScoped

public class ErrorHandler {
       public String getStatusCode(){
       String val = String.valueOf((Integer)FacesContext.getCurrentInstance().getExternalContext().
      getRequestMap().get("javax.servlet.error.status_code"));
       return val;
}
       public String getMessage(){
       String val =  (String)FacesContext.getCurrentInstance().getExternalContext().
      getRequestMap().get("javax.servlet.error.message");
       return val;
       }
       public String getExceptionType(){
       String val = FacesContext.getCurrentInstance().getExternalContext().
     getRequestMap().get("javax.servlet.error.exception_type").toString();
       return val;
}
       public String getException(){
       String val = (String)((Exception)FacesContext.getCurrentInstance().getExternalContext().
              getRequestMap().get("javax.servlet.error.exception")).toString();
       return val;
       }
       public String getRequestURI(){
       return (String)FacesContext.getCurrentInstance().getExternalContext().
      getRequestMap().get("javax.servlet.error.request_uri");
       }
       public String getServletName(){
       return (String)FacesContext.getCurrentInstance().getExternalContext().
      getRequestMap().get("javax.servlet.error.servlet_name");
       }
}
```

Το error handler bean ορίζεται ως RequestScoped. Πολλαπλά objects που σχετίζονται με το σφάλμα τοποθετούνται στο RequestMap και θεωρούνται ως **Servlet Exception Attributes**.

3.5.3.1.4 Σελίδα επιρρεπής σε σφάλματα

index.xhtml

```
<html          xmlns="http://www.w3.org/1999/xhtml"          xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:h="http://java.sun.com/jsf/html" xmlns:f="http://java.sun.com/jsf/core">
       <f:view>
       <h:form prependId="false">
              <h1>JavaBeat JSF 2.2 Examples</h1>
              <h2>JSF2 - Error Handling</h2>
              <br />
              <h:outputText value="#{indexBean.message}"></h:outputText>
              <h:commandButton value="Throws Exception" action="#{indexBean.navigate}" />
```

```
        </h:form>
        </f:view>
</html>
```

3.5.3.1.5 Managed Bean επιρρεπές σε σφάλματα

IndexBean.java

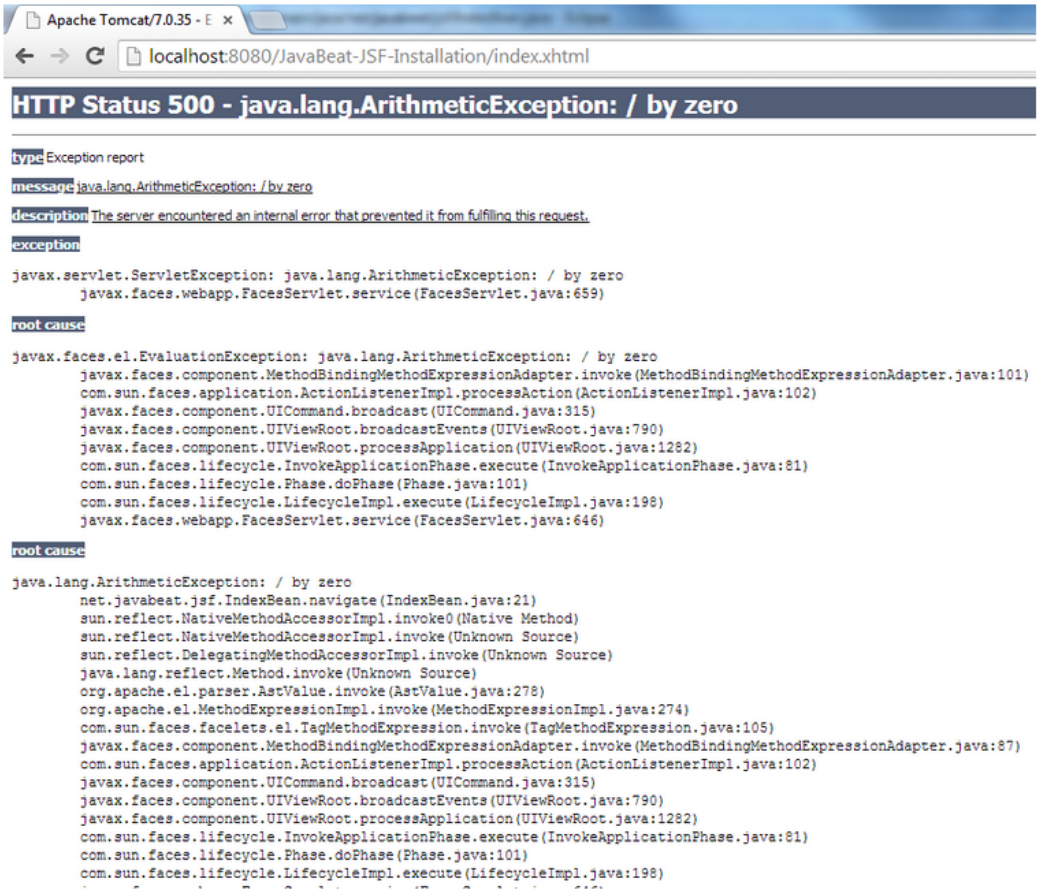
```
package net.javabeat.jsf;
    import javax.faces.bean.ManagedBean;
    import javax.faces.bean.SessionScoped;

    @ManagedBean
    @SessionScoped

    public class IndexBean {
       private String message;
       public String getMessage() {
                     return message;
       }
       public void setMessage(String message) {
                     this.message = message;
       }
       public String navigate(){
              // Assume an exception has been thrown by some business logic
                     System.out.println(10/0);
                     return "anonymousView";
       }
    }
```

3.5.3.1.6 Παρουσίαση διαχείρισης σφάλματος σε JSF

Η παρακάτω εικόνα δείχνει πώς μπορεί ένα thrown exception μπορεί να διαχειριστεί σε ένα view.

3.5.3.1.7 Διαχείριση Σφάλματος χωρίς custom σελίδα σφάλματος

Παρακάτω φαίνεται πώς θα ήταν η σελίδα χωρίς custom error page.



81

[81] "JSF Custom Error Pages - JavaBeat." 10 Απρ. 2014, https://javabeat.net/jsf-custom-error-pages/. Πρόσβαση στις 28 Οκτ. 2020.

3.5.3.2 Διαχείρηση Σφάλματος με Ajax

Το Ajax framework με το JSF 2.0 framework προσφέρει το attribute onerror για την διαχείριση των σφαλμάτων που μπορεί να γίνουν throw κατα τη διάρκεια που το JSF Framework εκτελεί τα στοιχεία που έχουν συμπεριληφθεί στο **f:ajax's execute** attribute. Η τιμή του onerror attribute είναι μία JavaScript function. Το JSF καλεί αυτό το function όταν υπάρχει ένα σφάλμα κατα τη διάρκεια επεξεργασίας ενός Ajax request. Όπως το f:ajax's onevent attribute, το JSF το onerror function data object. Οι τιμές για αυτό το object είναι ίδιες με τις τιμές για το data object που το JSF περνάει στο event function όπως αναγράφεται:

- **httpError**: Response status null ή undefined ή status <200 ή status >=300.
- **emptyResponse**: Δεν υπήρξε response από τον server.
- **maleformed**: Το response δεν ήταν σωστά δομημένο XML.
- **serveError**: Το Ajax response περιέχει ένα error element από το server.

Για σφάλματα, το αντικείμενο δεδομένων περιέχει επίσης τρεις ιδιότητες που δεν υπάρχουν για συμβάντα:

- **description**
- **errorName**
- **errorMessage**

3.5.3.2.1 To View

index.xhtml

```
<html          xmlns="http://www.w3.org/1999/xhtml"          xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:h="http://java.sun.com/jsf/html" xmlns:f="http://java.sun.com/jsf/core">
       <h:head>
       <script>
              if(!net) var net = {}
              if(!net.javabeat){
              net.javabeat = {
                     monitor:
                     function (data){
                     var loading = document.getElementById("image");
                     if(data.status == "begin"){
                            loading.style.display = "block";
                     }
                     else if(data.status == "success"){
                            loading.style.display = "none";
                     }
                     },
                     handle:
                     function(data){
                     console.log("Error Description: "+data.description);
                     console.log("Error Name: "+data.errorName);
                     console.log("Error errorMessage: "+data.errorMessage);
                     console.log("httpError: "+data.httpError);
                     console.log("emptyResponse: "+data.emptyResponse);
```

```
                    console.log("maleformed: "+data.maleformed);
                    console.log("serverError: "+data.serverError);
                    }
            }
            }
        </script>
        <h:outputScript library="javax.faces" name="jsf.js" />
        </h:head>
        <h:body>
        <f:view>
            <h1>JavaBeat JSF 2.2 Examples</h1>
            <h2>JSF2 Using JavaScript Namespace Example</h2>
            <h:form prependId="false">
            <h:inputText id="input" value="#{indexBean.message}" />
            #{' '}
            <h:commandButton value="Display Text" action="#{indexBean.action}"><f:ajax execute="@this
input"                          render="output"                          onevent="net.javabeat.monitor"
onerror="net.javabeat.handle"></f:ajax></h:commandButton>
            #{' '}
            <h:outputText   id="output"   value="#{indexBean.message}"></h:outputText><h:graphicImage
id="image"                                              value="#{resource['images:ajax-loader.gif']}"
style="display:none;"></h:graphicImage></h:form>
        </f:view>
        </h:body>
</html>
```

3.5.3.2.2 Managed Bean

IndexBean.java

```
package net.javabeat.jsf;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
@ManagedBean
@SessionScoped
public class IndexBean {
        private String message;
        public String getMessage() {
          return message;
        }
        public void setMessage(String message) {
          this.message = message;
        }
        public String action() throws Exception {
          Thread.sleep(6000);
          System.out.println(5 + 10 / 0); // This is an assumption
          System.out.println(message);
          return "";
        }
}
```

3.5.3.2.3 Deployment Descriptor

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"             id="WebApp_ID"             version="2.5"
metadata-complete="true">
       <context-param>
          <description>State saving method: 'client' or 'server'
                              (=default). See JSF Specification 2.5.2
         </description>
         <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
          <param-value>client</param-value>
       </context-param>
       <context-param>
          <param-name>javax.faces.application.CONFIG_FILES</param-name>
          <param-value>/WEB-INF/faces-config.xml</param-value>
       </context-param>
       <servlet>
          <servlet-name>Faces Servlet</servlet-name>
          <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
          <load-on-startup>1</load-on-startup>
       </servlet>
       <servlet-mapping>
          <servlet-name>Faces Servlet</servlet-name>
          <url-pattern>/faces/*</url-pattern>
       </servlet-mapping>
       <servlet-mapping>
          <servlet-name>Faces Servlet</servlet-name>
          <url-pattern>*.xhtml</url-pattern>
       </servlet-mapping>
       <listener>
      <listener-class>com.sun.faces.config.ConfigureListener</listener-class>
       </listener>
</web-app>
```

3.5.3.2.4 Παρουσίαση Διαχείρισης Σφαλμάτων με Ajax στο JSF 2

Η παρακάτω εικόνα δείχνει πώς γίνονται handle τα σφάλματα που μπορεί να συμβούν κατά την επεξεργασία ενός Ajax request.

[82]

[82] "JSF 2 Ajax Error Handling Example - JavaBeat." 2 Απρ. 2014, https://javabeat.net/jsf-2-ajax-error-handling/. Πρόσβαση στις 28 Οκτ. 2020.

4.0 Static and Dynamic Security Testing (SAST & DAST)

4.1 Static Application Security Testing (SAST)

4.1.1 Εισαγωγή

Ο στατικός έλεγχος ασφάλειας εφαρμογών είναι ένα σύνολο τεχνολογιών που έχουν σχεδιαστεί για να αναλύουν τον πηγαίο κώδικα μιας εφαρμογής, το bytecode και τα binary αρχεία για συνθήκες που υποδεικνύουν ευπάθειες ασφαλείας. Οι λύσεις SAST αναλύουν μια εφαρμογή από "μέσα προς τα έξω" σε κατάσταση μη εκτέλεσης. Με χρήση των εργαλείων οι testers διαβάζουν τον πηγαίο κώδικα και προσπαθούν να βρουν λογικά σφάλματα, όπως για παράδειγμα loopholes στον έλεγχο data, κάτι που μπορεί να χρησιμοποιήσει ένας επιτιθέμενος για να πάρει πρόσβαση στο σύστημα. Το SAST αποκαλείται και white box testing επειδή ο πηγαίος κώδικας της εφαρμογής είναι διαθέσιμος και διαφανής. Τα εργαλεία του SAST επιστρέφουν στον tester μία μακροσκελή λίστα από ευρήματα, που πρέπει ο tester να ελέγξει ένα ένα, διότι πολλά από αυτά είναι false positives, ή δεν γίνεται να γίνουν exploit από έξω. [83]

Το SAST είναι πολύ σημαντικό για την ασφάλεια μιας εφαρμογής. Οι προγραμματιστές είναι πολύ περισσότεροι από το προσωπικό ασφαλείας. Μπορεί να είναι δύσκολο για μια επιχείρηση να βρει πόρους για την διεξαγωγή reviews κώδικα ακόμα και σε ένα μικρό μέρος της εφαρμογής τους. Τα εργαλεία SAST είναι πολύ πιο γρήγορα από τον χειροκίνητο έλεγχο κώδικα από ανθρώπους. Αυτά τα εργαλεία μπορούν να ελέγξουν εκατομμύρια γραμμές κώδικα σε μικρό χρονικό διάστημα. Αναγνωρίζουν αυτόματα κρίσιμες ευπάθειες όπως buffer overflows, SQL Injections, Cross-Site-Scripting επιθέσεις και άλλες, με υψηλή ακρίβεια. Γι'αυτό η ενσωμάτωση στατικής ανάλυσης στο SDLC μπορεί να οδηγήσει σε τεράστια βελτίωση της ποιότητας του κώδικα που αναπτύσσεται.[84]

4.1.2 Βήματα για την αποτελεσματική εκτέλεση του SAST

Απαιτούνται έξι απλά βήματα για την αποτελεσματική εκτέλεση του SAST σε οργανισμούς που διαθέτουν πολύ μεγάλο αριθμό εφαρμογών που έχουν δημιουργηθεί με διαφορετικές γλώσσες, πλαίσια και πλατφόρμες.

1. **Οριστικοποίηση εργαλείου**: Πρέπει να επιλεγεί ένα εργαλείο στατικής ανάλυσης που διεξάγει ελέγχους κώδικα σε εφαρμογές που είναι γραμμένες στις γλώσσες προγραμματισμού που έχουν χρησιμοποιηθεί. Το εργαλείο θα πρέπει επίσης να μπορεί να συμπεριλαμβάνει το framework που χρησιμοποιείται από το λογισμικό της εφαρμογής.

2. **Δημιουργία της υποδομής σάρωσης και εφαρμογή του εργαλείου**: Αυτό το βήμα περιλαμβάνει τον χειρισμό των απαιτήσεων αδειοδότησης, τη ρύθμιση ελέγχου πρόσβασης και εξουσιοδότησης και την προμήθεια των πόρων που απαιτούνται (π.χ. διακομιστές και βάσεις δεδομένων) για την ανάπτυξη του εργαλείου.

3. **Παραμετροποίηση του εργαλείου**: Το εργαλείο πρέπει να παραμετροποιηθεί ως προς τις ανάγκες του οργανισμού. Για παράδειγμα, μπορεί να διαμορφωθεί έτσι ώστε να μειωθούν τα false positives ή να βρεθούν επιπλέον ευπάθειες ασφαλείας γράφοντας νέους κανόνες ή ενημερώνοντας τους υπάρχοντες. Πρέπει να ενσωματωθεί το εργαλείο στο περιβάλλον δημιουργίας, να δημιουργηθούν πίνακες ελέγχου για την παρακολούθηση αποτελεσμάτων σάρωσης και να δημιουργηθούν προσαρμοσμένες αναφορές.

4. **Προτεραιότητα και ενσωμάτωση εφαρμογών**: Όταν το εργαλείο είναι έτοιμο πρέπει να ενσωματωθούν οι εφαρμογές που πρέπει να ελεγχθούν. Αν υπάρχει μεγάλος αριθμός εφαρμογών πρέπει να μπουν σε προτεραιότητα οι εφαρμογές υψηλού ρίσκου. Στο τέλος όλες οι εφαρμογές πρέπει να ενσωματωθούν και να γίνεται σάρωση σε αυτές τακτικά, συγχρονίζοντας τις σαρώσεις με τους κύκλους νέων releases, καθημερινά μηνιαία ή με την ενσωμάτωση νέου κώδικα.

---

[83] "Definition of Static Application Security Testing (SAST) - Gartner." https://www.gartner.com/en/information-technology/glossary/static-application-security-testing-sast. Πρόσβαση στις 2 Νοε. 2020.

[84] "What Is SAST and How Does Static Code Analysis ... - Synopsys." https://www.synopsys.com/glossary/what-is-sast.html. Πρόσβαση στις 2 Νοε. 2020.

5. **Ανάλυση αποτελεσμάτων σάρωσης**: Αυτό το βήμα περιλαμβάνει τον έλεγχο των αποτελεσμάτων της σάρωσης για την αφαίρεση false positives. Μόλις ολοκληρωθεί το σύνολο των προβλημάτων, θα πρέπει να παρακολουθούνται και να παρέχονται στις ομάδες ανάπτυξης για σωστή και έγκαιρη αποκατάσταση.

6. **Παροχή διακυβέρνησης και κατάρτισης**: Η σωστή διακυβέρνηση διασφαλίζει ότι οι ομάδες ανάπτυξης χρησιμοποιούν σωστά τα εργαλεία σάρωσης. Οι τεχνικές του SAST θα πρέπει να υπάρχουν εντός του SDLC. Το SAST πρέπει να ενσωματωθεί ως μέρος της διαδικασίας ανάπτυξης της εφαρμογής.[85]

4.1.3 Αυτοματοποιημένη Ανάλυση Στατικού Κώδικα

Η ανάλυση πηγαίου κώδικα είναι ο αυτοματοποιημένος έλεγχος του πηγαίου κώδικα με σκοπό τον εντοπισμό σφαλμάτων ενός προγράμματος υπολογιστή ή μιας εφαρμογής πριν διανεμηθεί ή πωληθεί. Ο πηγαίος κώδικας αποτελείται από statements που δημιουργούνται με ένα πρόγραμμα επεξεργασίας κειμένου ή ένα οπτικό εργαλείο προγραμματισμού και στη συνέχεια αποθηκεύονται σε ένα αρχείο. Ο πηγαίος κώδικας είναι η πιο μόνιμη μορφή ενός προγράμματος, παρόλο που το πρόγραμμα μπορεί αργότερα να τροποποιηθεί, να βελτιωθεί ή να αναβαθμιστεί.

Η ανάλυση πηγαίου κώδικα μπορεί να είναι είτε στατική είτε δυναμική. Στη στατική ανάλυση, ο εντοπισμός σφαλμάτων γίνεται εξετάζοντας τον κώδικα χωρίς να εκτελέσετε πραγματικά το πρόγραμμα. Αυτό μπορεί να αποκαλύψει σφάλματα σε πρώιμο στάδιο στην ανάπτυξη προγραμμάτων, συχνά εξαλείφοντας την ανάγκη για πολλαπλές αναθεωρήσεις αργότερα. Αφού γίνει στατική ανάλυση, η δυναμική ανάλυση πραγματοποιείται σε μια προσπάθεια να αποκαλυφθούν πιο ανεπαίσθητα ελαττώματα ή ευπάθειες. Η δυναμική ανάλυση αποτελείται από δοκιμές προγράμματος σε πραγματικό χρόνο.

Ένα σημαντικό πλεονέκτημα αυτής της μεθόδου είναι το γεγονός ότι δεν απαιτεί από τους προγραμματιστές να κάνουν εικασίες σε καταστάσεις που ενδέχεται να προκαλέσουν σφάλματα. Άλλα πλεονεκτήματα περιλαμβάνουν την εξάλειψη περιττών στοιχείων του προγράμματος και τη διασφάλιση ότι το υπό δοκιμή πρόγραμμα είναι συμβατό με άλλα προγράμματα που ενδέχεται να εκτελούνται ταυτόχρονα.[86]
https://techexpert.tips/sonarqube/sonarqube-docker-installation/

4.1.3.1 Στατική Ανάλυση Κώδικα με χρήση του SonarQube

Το SonarQube είναι μια πλατφόρμα ανοιχτού κώδικα για συνεχή έλεγχο της ποιότητας κώδικα. Χρησιμοποιώντας ανάλυση στατικού κώδικα, προσπαθεί να εντοπίσει σφάλματα, “code smells” και ευπάθειες ασφαλείας. Το SonarQube υποστηρίζει πολλές γλώσσες μέσω ενσωματωμένων συνόλων κανόνων και μπορεί επίσης να επεκταθεί με διάφορα πρόσθετα.

Μερικά πλεονεκτήματα του SonarQube είναι τα ακόλουθα:

- ❏ Αναπτύσσεται συνεχώς και είναι καλά υλοποιημένο. Πολλά πρόσθετα είναι διαθέσιμα για χρήση ως continuous integration pipelines, συμπεριλαμβανομένων των Maven, Jenkins και GitHub.

- ❏ Τα ενσωματωμένα σύνολα κανόνων μπορούν να επεκταθούν με προσθήκες που είναι περισσότερο προσανατολισμένες στην ασφάλεια. Για παράδειγμα, αν χρησιμοποιηθεί το πρόσθετο FindBugs θα υπάρξει η εφαρμογή των κανόνων του FindBugs.

- ❏ Μπορεί επίσης να αναφέρει γεγονότα όπως ο διπλότυπος κώδικα, κάλυψη κώδικα ή πρότυπα κωδικοποίησης.

4.1.3.1.1 Εγκατάσταση

Για να γίνει η διαδικασία πιο εύκολη θα χρησιμοποιηθεί το παρακάτω Docker Compose YAML αρχείο.

---

[85] "What Is SAST and How Does Static Code Analysis ... - Synopsys." https://www.synopsys.com/glossary/what-is-sast.html. Πρόσβαση στις 2 Νοε. 2020.

[86] "What is source code analysis? - Definition from WhatIs.com." https://searchsoftwarequality.techtarget.com/definition/source-code-analysis. Πρόσβαση στις 5 Νοε. 2020.

```
version: '3'
services:
  mysql:
    image: mysql
    container_name: "mysql-thibaut"
    volumes:
      - ./custom-mysql.cnf:/etc/mysql/conf.d/custom-mysql.cnf
    environment:
      - MYSQL_ROOT_PASSWORD=sonarqube
      - MYSQL_DATABASE=sonarqube
      - MYSQL_USER=sonarqube
      - MYSQL_PASSWORD=sonarqube
  sonarqube:
    image: sonarqube:6.7
    container_name: "sonarqube-thibaut"
    hostname: "sonarqube-thibaut"
    links:
      - mysql
    ports:
      - "9000:9000"
    volumes:
      - ./start.sh:/opt/sonarqube/start.sh
                                                                                                -
./sonar-dependency-check-plugin-1.1.0.jar:/opt/sonarqube/extensions/plugins/sonar-dependency-check-plugi
n-1.1.0.jar
    entrypoint: /opt/sonarqube/start.sh
    environment:
      - USER_LOGIN=<your-login>
      - USER_NAME=<your-name>
      - USER_PASSWORD=<your-password>
      - SONARQUBE_JDBC_USERNAME=sonarqube
      - SONARQUBE_JDBC_PASSWORD=sonarqube
                                                                                                -
SONARQUBE_JDBC_URL=jdbc:mysql://mysql:3306/sonarqube?useUnicode=true&characterEncoding=utf8&rewriteBatch
edStatements=true
```

Παρακάτω είναι το σχετικό script start.sh που χρησιμεύει ως σημείο εισόδου. Είναι κυρίως υπεύθυνο για τη δημιουργία ενός νέου χρήστη του οποίου τα διαπιστευτήρια ορίστηκαν παραπάνω και με υψηλά προνόμια. Προσθέτουμε επίσης το αρχείο JAR για το OWASP Dependency Check που θα παρουσιαστεί αργότερα.

```
#!/bin/bash

# Wait for MySQL database to be running
sleep 20

# Start SonarQube
./bin/run.sh &

curlAdmin() {
    curl -u admin:admin $@
}

URL_API=http://localhost:9000/api
```

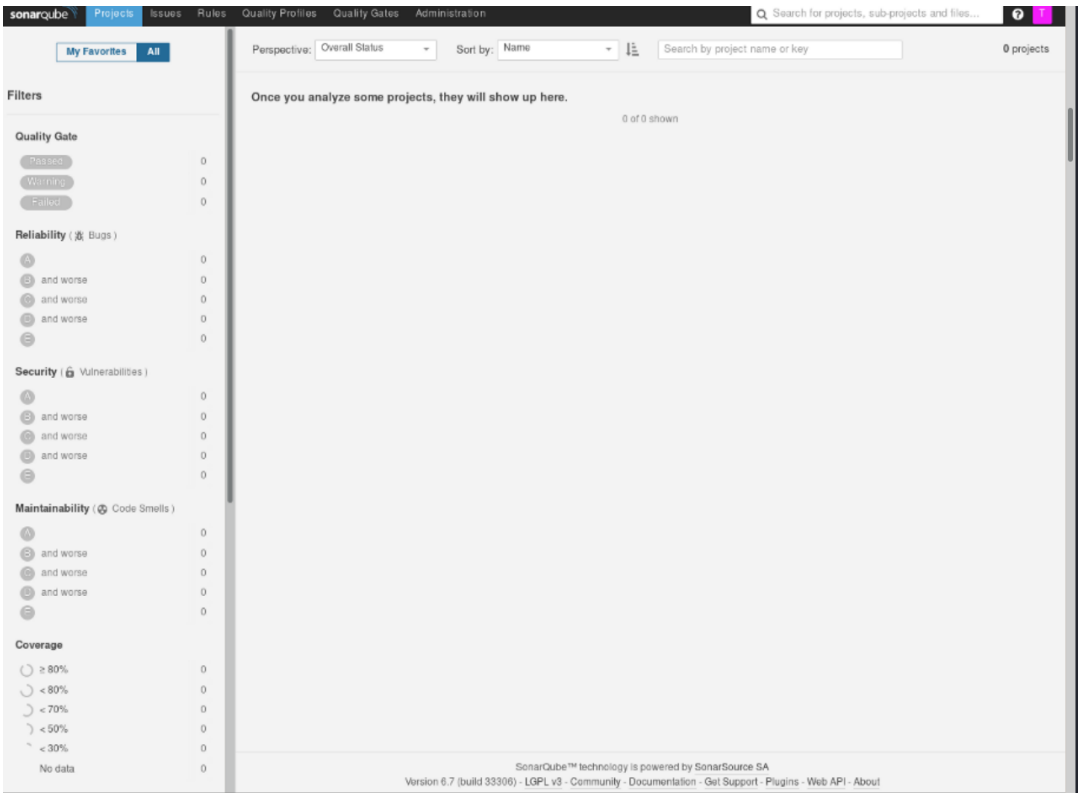
```
isUp() {
    curl -s -u admin:admin -f "$URL_API/system/info"
}

# Wait for server to be up
while [ -z "$(isUp)" ] ; do sleep 5 ; done

# Provision an admin user
if [ "$USER_LOGIN" ] && [ "$USER_NAME" ] && [ "$USER_PASSWORD" ]
then
    curlAdmin "$URL_API/users/create" \
                    --data-urlencode "login=$USER_LOGIN" \
                    --data-urlencode "name=$USER_NAME" \
                    --data-urlencode "password=$USER_PASSWORD"
    #RIGHTS=( "admin" "profileadmin" ) # Other rights: gateadmin, scan, provisioning
    #for right in "${RIGHTS[@]}" ; do
    #    curlAdmin "$URL_API/permissions/add_user" \
    #                 --data-urlencode "login=$USER_LOGIN" \
    #                 --data-urlencode "permission=$right"
    #done

    # Above rights are only for general administration of SonarQube. We need
    # to be a member of the sonar-administrators group to be able to manage
    # projects permissions.
    curlAdmin "$URL_API/user_groups/add_user" \
                        --data-urlencode "name=sonar-administrators" \
                        --data-urlencode "login=$USER_LOGIN"
    curl -u $USER_LOGIN:$USER_PASSWORD "$URL_API/users/deactivate" \
                    --data-urlencode "login=admin"
fi

wait
```

Πρέπει να σημειωθεί ότι χρησιμοποιούμε μια εξωτερική βάση δεδομένων MySQL αντί για την προεπιλεγμένη ενσωματωμένη. Αυτό είναι σημαντικό για διάφορους λόγους, συμπεριλαμβανομένης της δυνατότητας μετεγκατάστασης όλων των δεδομένων μέσω αναβαθμίσεων SonarQube. Επιπλέον, εδώ είναι το πρόσθετο snippet διαμόρφωσης που απαιτείται για την αποφυγή σφαλμάτων αργότερα όταν ο Maven θα στέλνει αναφορές ανάλυσης στο SonarQube:

```
[mysqld]
max_allowed_packet = 1073741824
```

Όσο για το αρχείο OWASP Dependency Check JAR, μπορεί κάποιος να το κατεβάσει εδώ.

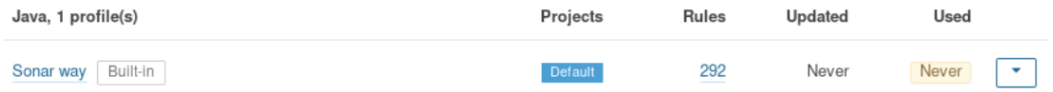
Έπειτα είναι εφικτό να συνδεθούμε στη διεπαφή Ιστού SonarQube στη θύρα 9000 μέσω ενός web browser όπου θα υπάρχει η ακόλουθη σελίδα:



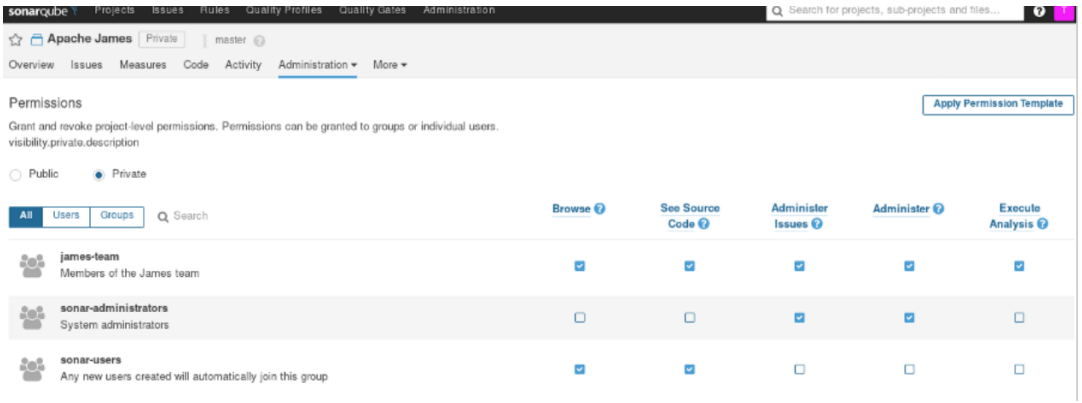
Όπως ήταν αναμενόμενο, η σελίδα είναι κενή και πρέπει να δημιουργηθεί ένα νέο project. Πριν από αυτό, υπάρχει η δυνατότητα της εξερεύνησης του Administration section για τη διαμόρφωση του instance του SonarQube. Για παράδειγμα, στην ενότητα General Settings -> Security, μπορεί ο χρήστης να ενεργοποιήσει την επιλογή Force user authentication. Επίσης μπορούν να διαμορφωθούν webhooks, να ρυθμιστεί το πρόγραμμα καθαρισμού βάσης δεδομένων ή να διαμορφωθεί ένας λογαριασμός email για την αποστολή ειδοποιήσεων. Στην καρτέλα System, μπορεί επίσης να ελεγχθεί η κατάσταση του JVM, της βάσης δεδομένων MySQL ή των μηχανών υπολογισμού και αναζήτησης. Στο Administration -> Security, μπορεί ο χρήστης επίσης να διαχειριστεί τα δικαιώματα χρήστη, τις ομάδες χρηστών και να δημιουργηθούν πρότυπα δικαιωμάτων.

4.1.3.1.2 Δημιουργία νέου Project

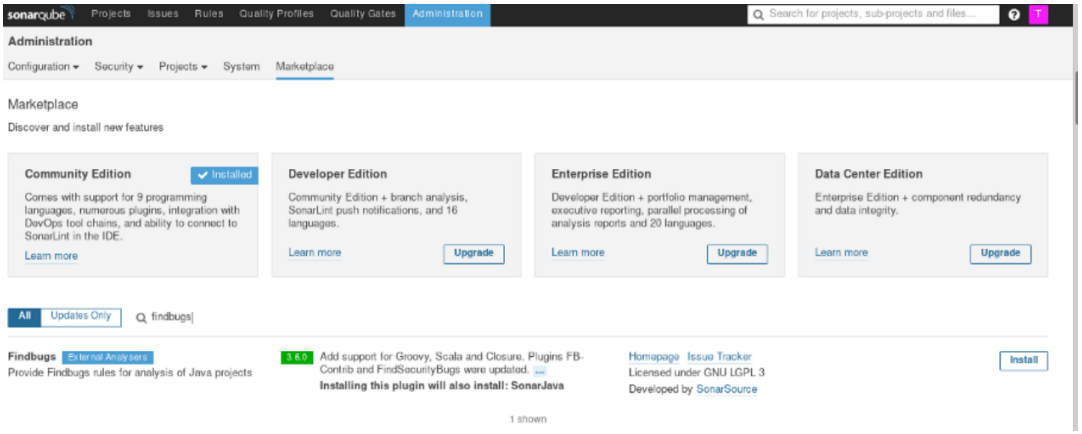
Ο στόχος είναι να εκτελεστεί μια ανάλυση ασφαλείας για τον πηγαίο κώδικα του James mail server. Πριν εκτελεστεί μια ανάλυση, πρέπει να ρυθμιστεί ένα νέο έργο και να του εκχωρηθεί ένα **quality profile**. Ένα quality profile είναι μια συλλογή κανόνων που θα εφαρμοστούν κατά τη διάρκεια μιας ανάλυσης. Υπάρχει ένα προεπιλεγμένο για κάθε γλώσσα. Για παράδειγμα, εάν δεν γίνει περαιτέρω διαμόρφωση και γίνει ανάλυση άμεσα στο project James, θα χρησιμοποιηθεί το ενσωματωμένο quality profile Java με το όνομα Sonar way.



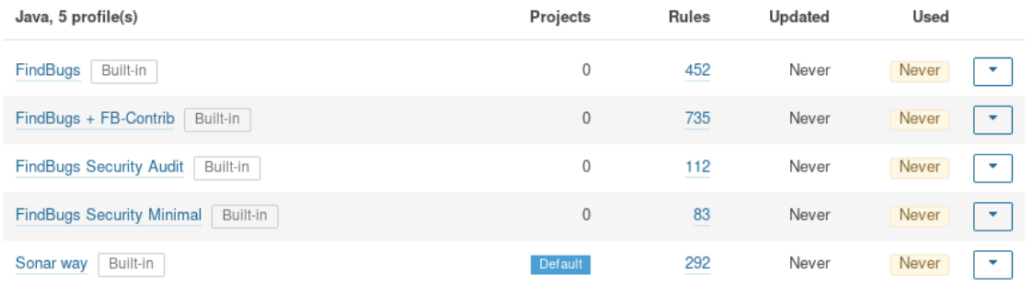
Για να δημιουργηθεί το νέο project Apache James, ακολουθείται η ενότητα Projects -> Management και επιλέγεται η δημιουργία νέου project. Μπορεί να επιλεγεί από το χρήστη το όνομα του project καθώς και ένα project key, το οποίο πρέπει να είναι μοναδικό. Στη συνέχεια, μια καλή πρακτική θα ήταν για παράδειγμα να δημιουργηθεί ένα νέο group με πλήρη δικαιώματα σε αυτό το project. Πρώτα πρέπει να δημιουργηθεί η ομάδα από το general administration panel και μετά να δοθούν δικαιώματα από το Apache James project administration tab. Στην παρακάτω περίπτωσή αυτή, δημιουργήθηκε η ομάδα james-team για το project Apache James:



Στη συνέχεια, πρέπει να επιλεγεί ένα διαφορετικό quality profile για το έργο. Θα χρησιμοποιηθεί το Findbugs Security Audit quality profile που διαθέτει πολλούς διαθέσιμους κανόνες ασφαλείας. Για αυτό, πρέπει να εγκατασταθεί το plugin Findbugs. Στην καρτέλα Marketplace του Administration section ο χρήστης πρέπει να κάνει αναζήτηση με τον όρο **findbugs**. Το πρώτο και μοναδικό αποτέλεσμα είναι καλό και πρέπει να γίνει κλικ στο Install.

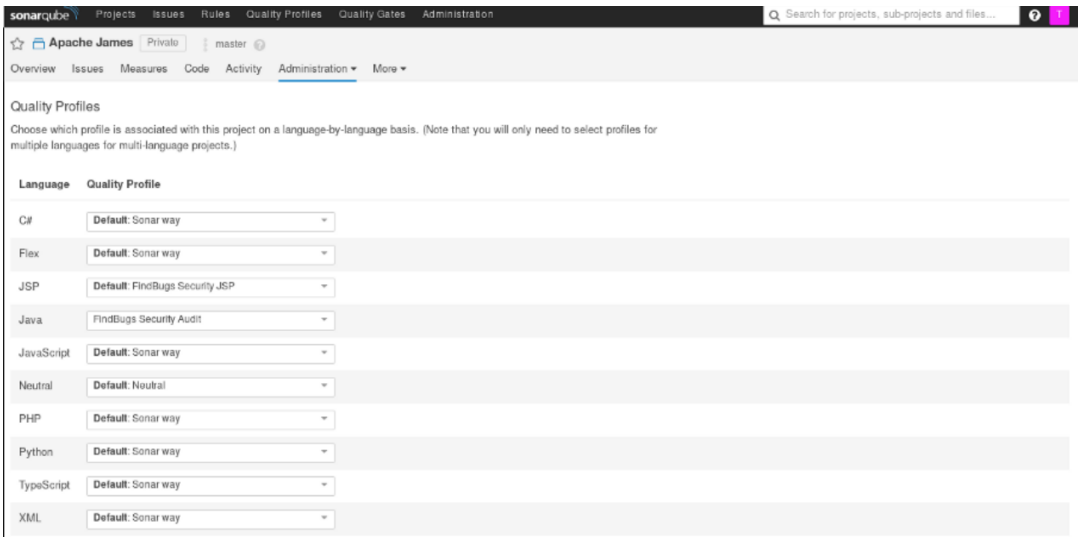


Θα ζητηθεί η επανεκκίνηση του SonarQube για να ολοκληρωθεί η εγκατάσταση του plugin. Μόλις γίνει αυτό, μπορεί να δει κανείς ότι είναι διαθέσιμα νέα quality profiles:



Μπορεί να γίνει έλεγχος των νέα quality profiles και να δει ο χρήστης ποιοι κανόνες εισάγονται και χρησιμοποιούνται. Στο SonarQube, οι κανόνες χωρίζονται σε τρεις self-explaining κατηγορίες: σφάλματα, ευπάθειες και code smells. Από προεπιλογή, ορισμένα είναι ενεργά και μερικά όχι. Αυτό φυσικά μπορεί να αλλάξει.

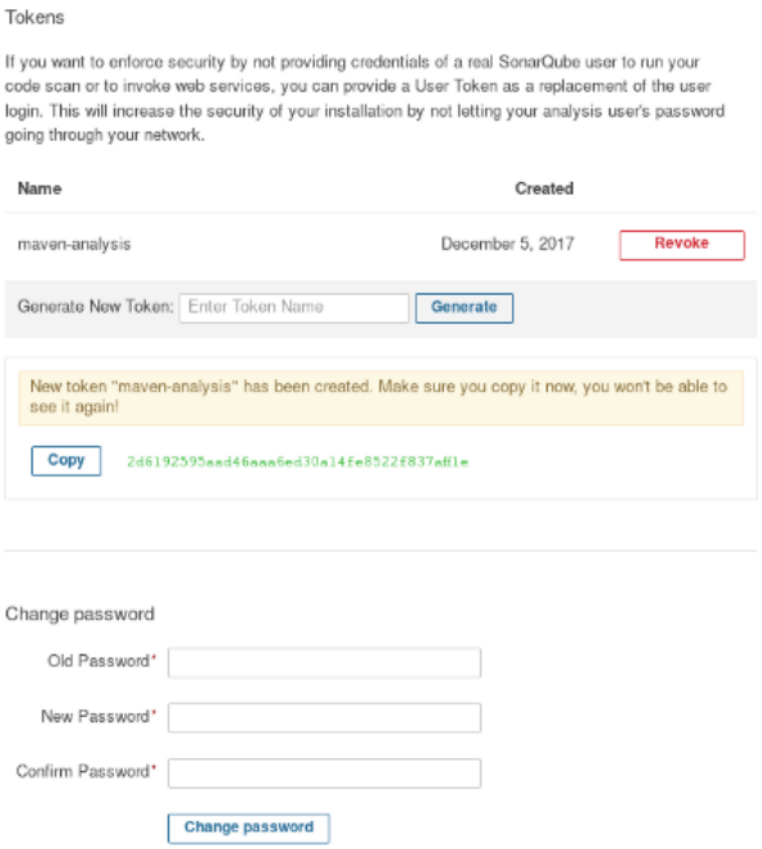
Πίσω στην ενότητα project του Apache James, μπορεί τώρα ο χρήστης να ορίσει το quality profile που θέλει να χρησιμοποιήσει. Δεδομένου ότι δεν έχει γίνει ακόμη ανάλυση, το SonarQube δεν γνωρίζει ότι το έργο περιέχει μόνο Java και, ως εκ τούτου, είναι ακόμα δυνατό να επιλέξετε ένα διαφορετικό quality profile για κάθε γλώσσα.



Ήρθε η ώρα να πραγματοποιηθεί η πρώτη ανάλυση του έργου Apache James.

4.1.3.1.3 Εκτέλεση Ανάλυσης

Το project χρησιμοποιεί το Maven σαν αυτοματοποιημένο εργαλείο κατασκευής. Το Maven λειτουργεί καλά με το SonarQube Scanner plugin. Το μόνο που χρειάζεται είναι η προσθήκη μερικών ρυθμίσεων στο configuration του Maven και μετά θα υπάρχει η δυνατότητα εκτέλεσης μιας απλής mvn εντολής από το πηγαίο repository του project James. Πρώτα απ όλα, επειδή χρειάζεται αυθεντικοποίηση πρέπει να παραχθεί ένα security token. Γι' αυτό πρέπει να γίνει πλοήγηση στο My Account section του χρήστη και πρέπει να γίνει χρήση της εκτέλεσης της ανάλυσης, ώστε να δημιουργηθεί ένα νέο token:



Η ανάλυση του project Maven αποτελείται από την εκτέλεση sonar:sonar στο source directory του project. Πρώτα πρέπει να προστεθεί το ακόλουθο Profile στο Maven configuration (συνήθως στο conf/settings.xml) ή στο αρχείο pom.xml του project:

```
<settings>
	<pluginGroups>
	<pluginGroup>org.sonarsource.scanner.maven</pluginGroup>
	</pluginGroups>
	<profiles>
	<profile>
		<id>sonar</id>
		<activation>
		<activeByDefault>true</activeByDefault>
		</activation>
		<properties>
		<sonar.host.url>http://myserver:9000</sonar.host.url>
		<sonar.login>2d6192595aad46aaa6ed30a14fe8522f837aff1e</sonar.login>
		<sonar.projectName>Apache James</sonar.projectName>
		<sonar.projectVersion>master</sonar.projectVersion>
		<sonar.test.exclusions>**/test/**/*.*</sonar.test.exclusions>

<sonar.exclusions>**/ai/**/*.*,**/jdbc/**/*.*,**/mpt/**/*.*,**/jcr/**/*.*,**/JDBC*</sonar.exclusions>
		</properties>
	</profile>
	</profiles>
</settings>
```

Φαίνεται το security token που παράχθηκε παραπάνω. Επίσης μπορεί κάποιος να αναρωτηθεί γιατί δεν έχει ρυθμιστεί το project key. Βασικά το maven χρησιμοποιεί αυτόματα το ακόλουθο μοτίβο για ένα project key: <groupId>:<artifactId>. Δεν μπορεί να γίνει κάτι γι' αυτό. Αν επιλεχθεί να κρατηθεί το κλειδί και να ρυθμιστεί στο παραπάνω configuration αρχείο, το Maven θα "παραπονεθεί" γι' αυτό και θα κάνει throw ενός error. Αν χρησιμοποιούνταν ένα διαφορετικό κλειδί, το SonarQube θα δημιουργούσε ένα διαφορετικό Project και δεν θα υπήρχε όφελος από τις παραπάνω ρυθμίσεις που έγιναν. Γι' αυτό πρέπει να γίνει ενημέρωση του κλειδιού του project στο SonarQube web interface. Στην περίπτωση αυτή ρυθμίζεται σε org.apache.james:james-project.

Επίσης εξαιρούνται όλα τα test αρχεία από την ανάλυση, αλλά και άλλα source directories που δεν ενδιαφέρουν τον tester.

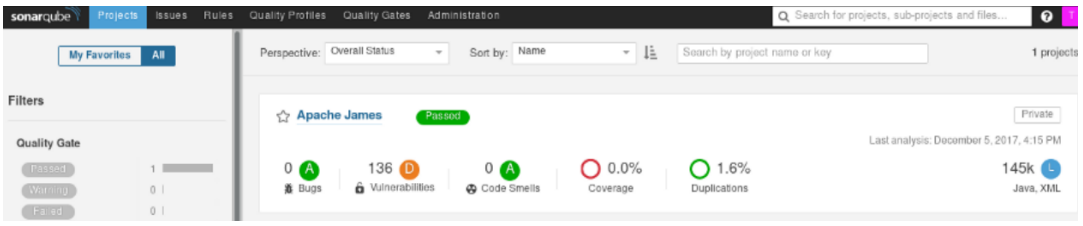
Έπειτα πρέπει να γίνει εκτέλεση της ακόλουθης εντολής για να γίνει το build του James και να δοθεί η άδεια στο Maven να συνεργαστεί κατευθείαν με τον SonarQube server για την εκτέλεση του configured quality profile.

mvn clean install sonar:sonar -DskipTests

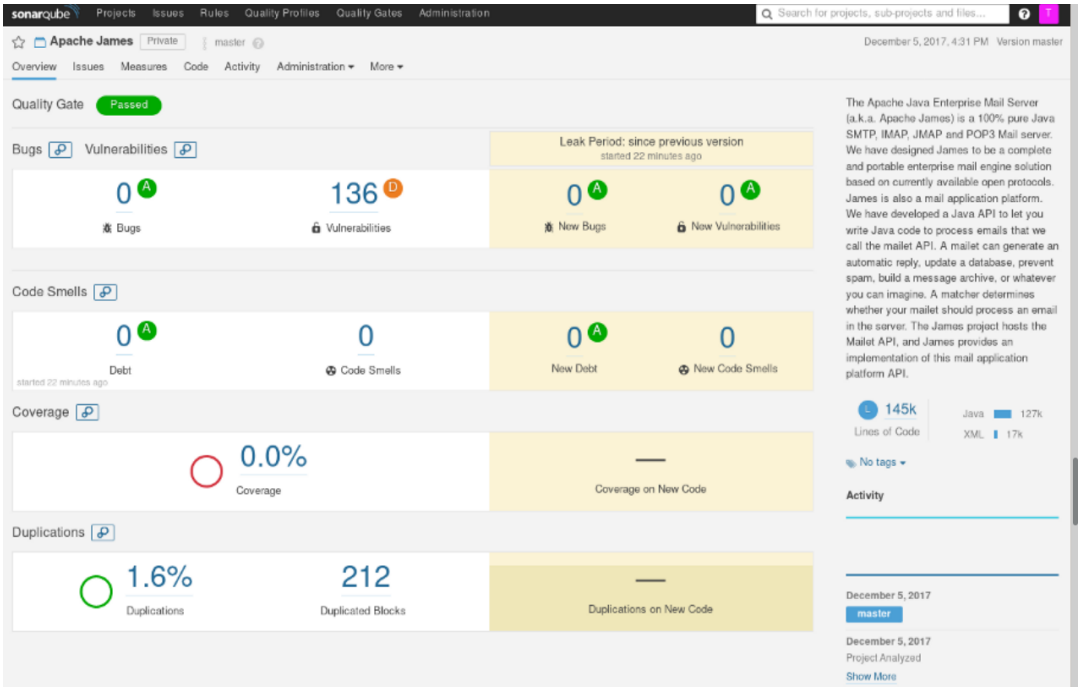
Επίσης διαλέγεται η επιλογή αποφυγής των tests για να μειωθεί ο χρόνος εκτέλεσης αλλά πρέπει να σημειωθεί ότι σαν συνέπεια το SonarQube δεν θα παράξει σχετικά test δεδομένα, όπως code coverage και test success statistics. Γι' αυτό αν ο χρήστης χρειάζεται αυτά τα δεδομένα πρέπει η παραπάνω εντολή να εκτελεστεί χωρίς το flag -DskipTests.

Κάτι που επίσης πρέπει να σημειωθεί είναι ότι όλες οι ιδιότητες που προστίθενται στο αρχείο XML για το configuration παραπάνω μπορεί να προσφερθεί και ως επιλογές command-line. Αυτό μπορεί να είναι χρήσιμο, για παράδειγμα όταν ένα τέτοιο αρχείο είναι μέσα στο Git repository και δεν θέλει ο χρήστης να διαρρεύσει το login security token.

Όταν το build και η ανάλυση ολοκληρωθούν, θα πρέπει να εμφανιστεί ένα παράθυρο όπως το παρακάτω στο web interface του SonarQube:

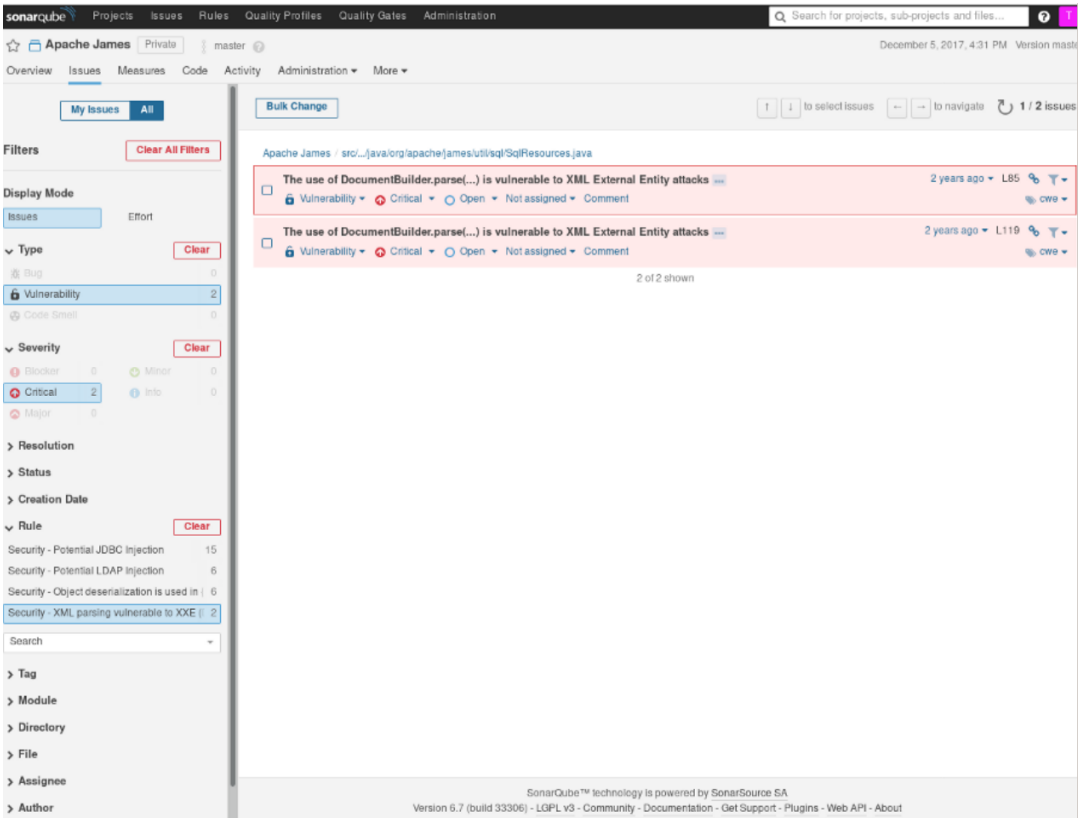


Επίσης στο overview page του project θα πρέπει να εμφανίζεται το παρακάτω παράθυρο:



Παραπάνω φαίνεται ο αριθμός των bugs που βρέθηκαν, ευπάθειες και code smells. Επίσης φαίνεται το εύρος κώδικα, όμως είναι 0 στην περίπτωση αυτή επειδή δεν τρέξαμε τους ελέγχους. Επίσης φαίνονται στατιστικά σχετικά με διπλότυπο κώδικα. Τέλος εμφανίζεται η εξέλιξη που υπήρξε από την τελευταία ανάλυση, ώστε να μπορεί ο χρήστης να δει αν υπήρξε βελτιωση ή όχι.

Έπειτα ο χρήστης μπορεί να πάει στο Issues tab. Είναι εφικτή η εφαρμογή φίλτρων με πολλά κριτήρια όπως το severity, αρχεία ή κανόνες:



Μπορεί να επιλεγεί ένα issue ώστε να γίνει μεταφορά στο σχετικό κομμάτι πηγαίου κώδικα. Μετά ο χρήστης μπορεί για παράδειγμα να αποφασίσει να αλλάξει την κατηγορία του ζητήματος, να προσαρμόσει το severity του, να το επιβεβαιώσει ή να το μαρκάρει ως λυμένο, να το αναθέσει σε κάποιον, να γράψει ένα σχόλιο κλπ.

Προφανώς, θα εμφανιστούν πολλά false positives. Αν ο χρήστης δει ότι κάποια quality profiles που περιλαμβάνουν κανόνες bugs και code smells, δεν τον ενδιαφέρουν μπορεί να τα απενεργοποιήσει. Επίσης υπάρχει δυνατότητα “μιξαρίσματος” κανόνων δημιουργώντας έτσι διάφορα custom quality profiles.

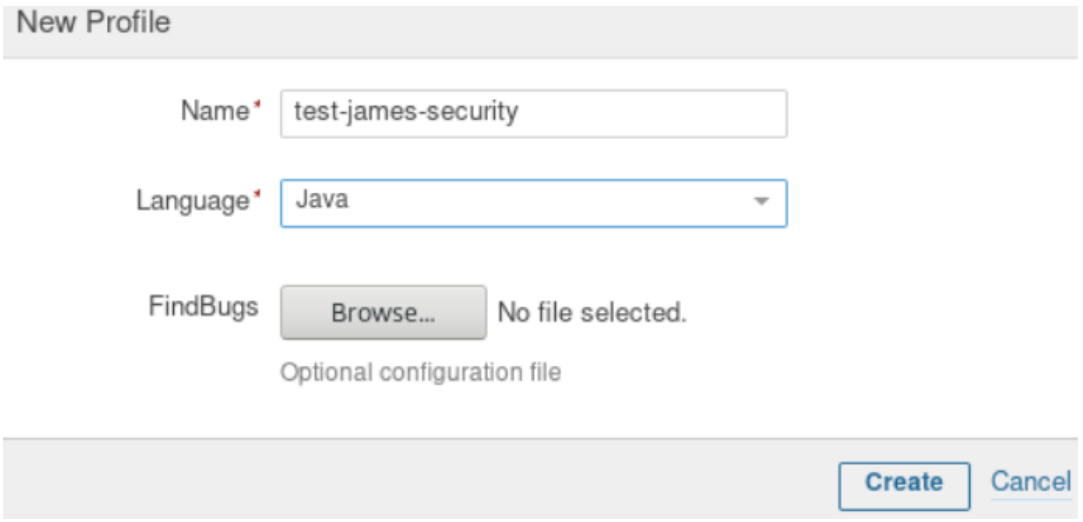
4.1.3.1.4 Δημιουργία Custom Quality Profile

Η δημιουργία custom quality profiles είναι ενδιαφέρουσα επειδη:

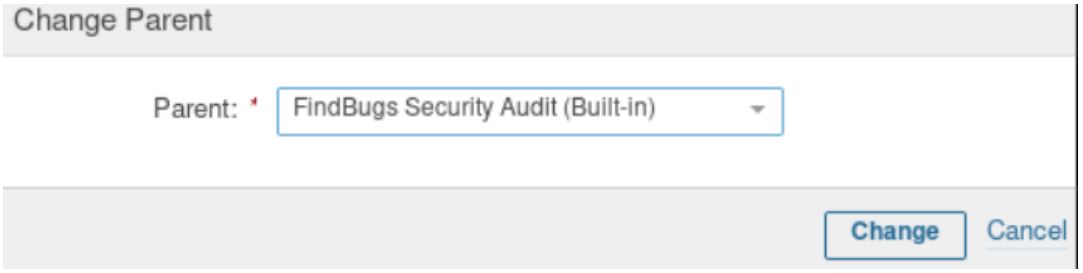
- ❏ Τα default profiles δεν μπορούν να γίνουν edit, όπότε ο χρήστης δεν θα μπορεί να τα παραμετροποιήσει στις ανάγκες του.
- ❏ Αυτό επιτρέπει στο χρήστη να αντιμετωπίζει τα default profiles σαν βάση και να παρακολουθεί το profile του καθώς κάνει αλλαγές σε αυτό.
- ❏ Τα default profiles ενημερώνονται με κάθε νέα έκδοση του σχετικού plugin για την προσθήκη κανόνων και κάποιες φορές τον μετασχηματισμό του severity του κάθε κανόνα. Αυτές οι αλλαγές εφαρμόζονται αυτόματα σε υπάρχουσες αντιγραφές των προφίλ.

Να σημειωθεί ότι τα ακόλουθα απαιτούν Administer Quality profiles global άδεια ή ο χρήστης να είναι μέλος του group sonar-administrators, που ισχύει ήδη σε αυτή την περίπτωση από το αρχικό bash script.

Στο Quality Profiles section, ο χρήστης δημιουργεί ένα νέο quality profile:

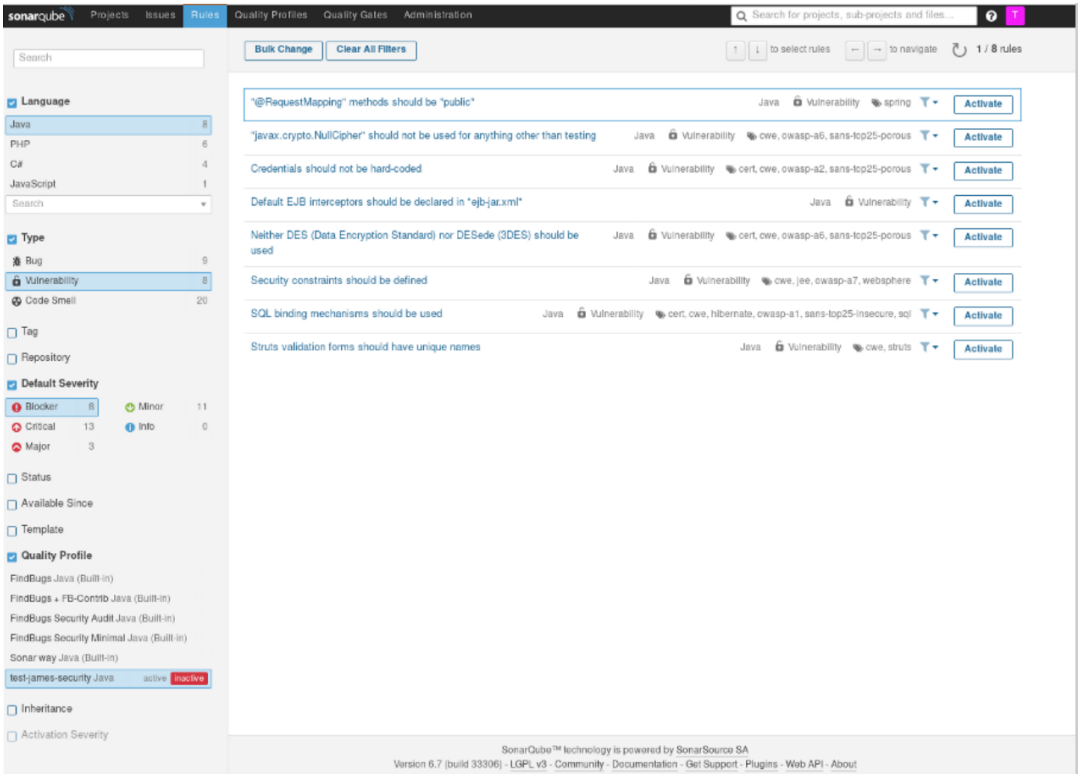


Έπειτα ο χρήστης πρέπει να εκμεταλλευτεί την κληρονομικότητα μεταξύ των quality profiles αλλάζοντας τον parent του δημιουργημένου profile:



Δεν πρέπει να ξεχαστεί η ανάθεση του νέου quality profile στο project για να χρησιμοποιηθεί στην επόμενη ανάλυση.

Τώρα ο χρήστης μπορεί να ενεργοποιήσει ή να απενεργοποιήσει κανόνες. Για παράδειγμα, μπορεί ο χρήστης να δει για μη ενεργούς Java κανόνες στο FindBugs Security Audit quality profile και επίσης ότι είναι κατηγοριοποιημένοι σαν ευπάθειες με Blocker severity:



Επίσης είναι δυνατό να γίνει backup των custom quality profiles, κατεβάζοντας το σαν αρχείο, επιτρέποντας στο χρήστη να το κάνει Import αργότερα στην ίδια ή σε μία άλλη Instance του SonarQube. Ακόμα μπορεί να γίνει σύγκριση διαφορετικών quality profiles, για παράδειγμα αν θέλει ο χρήστης κάποια στιγμή να πάρει μία περίληψη του πώς τα custom quality profiles εξελίχθηκαν από τους original parents.

Τέλος, να σημειωθεί ότι οι κανόνες μπορούν να επισημανθούν ως deprecated. Αυτό είναι σαν μία προειδοποίηση και δεν απενεργοποιεί αυτόματα ενεργούς κανόνες. Τα deprecations θα εμφανιστούν ως plugin updates και θα μεταδοθούν στα custom quality profiles, αφού στην περίπτωση αυτή χρησιμοποιήθηκε κληρονομικότητα.

4.1.3.1.5 Συμπέρασμα

Το SonarQube είναι ένα σχετικά πρακτικό εργαλείο και έχει ένα καλό user interface. Μπορεί να γίνει ένα πολύ καλό εργαλείο, απλά χρειάζεται λίγο χρόνο ώστε ο χρήστης να το καταλάβει και να το εκμεταλλευτεί πλήρως. Το documentation μπορεί να βοηθήσει έναν χρήστη να ανακαλύψει περισσότερα για το SonarQube.[87]

4.2 Dynamic Application Security Testing (DAST)

4.2.1 Εισαγωγή

Το Dynamic application security testing (DAST) δοκιμάζει την ασφάλεια από την εξωτερική πλευρά της εφαρμογής web. Μία ορθή παρομοίωση θα ήταν η εξέταση ασφαλείας ενός χρηματοκιβωτίου τράπεζας κάνοντας επίθεση σε αυτό. Το DAST προϋποθέτει ότι ο δοκιμαστής ασφαλείας δεν έχει λάβει καμία γνώση για την εσωτερική δομή της εφαρμογής. Αυτό λέγεται “black box” μέθοδος δοκιμής, επειδή ο δοκιμαστής δεν μπορεί να δει μεταφορικά “το κουτί”. Ο στόχος του είναι να προσομοιάσει μία αληθινή επίθεση.

[87] "Using SonarQube to Analyze a Java Project | by ... - Medium." 15 Δεκ. 2017, https://medium.com/linagora-engineering/using-sonarqube-to-analyze-a-java-project-abeee15e3779. Πρόσβαση στις 8 Νοε. 2020.

Επίσης στην ερώτηση, αν το DAST είναι μία αυτοματοποιημένη μέθοδος ή όχι, η απάντηση είναι στη μέση, αφού το DAST είναι αρκετά ευρύ για να περιλαμβάνει και τις δύο μεθόδους. Μεγάλο μέρος των δοκιμασιών μπορούν να αυτοματοποιηθούν, αλλά χρειάζεται ακόμα η κρίση του ανθρώπου για την ανακάλυψη ευπαθειών.

4.2.2 Χρήση DAST

4.2.2.1 Αυτοματοποιημένο DAST

Όπως γνωρίζουμε, η ιδέα πίσω από το DAST είναι ότι μιμείται μια πραγματική επίθεση. Και όπως ένας ληστής τραπεζών, το πρώτο πράγμα που θα ασχοληθεί ένας πραγματικός εισβολέας στον κυβερνοχώρο είναι οι εγκαταστάσεις.

Ένας crawler είναι ένας τύπος bot που μπορεί αυτόματα να επισκέπτεται και να καταγράφει κάθε σελίδα μιας εφαρμογής ιστού. Οπλισμένος με αυτήν τη γνώση, μπορεί στη συνέχεια να δημιουργήσει έναν χάρτη. Η κατασκευή ενός crawler είναι στην πραγματικότητα πολύ πιο περίπλοκη από ό, τι ακούγεται, δεδομένης της δυναμικής και ευμετάβλητης φύσης πολλών σύγχρονων εφαρμογών ιστού.

Στη συνέχεια, στην περίπτωση του Owasp Zap, το λογισμικό θα ελέγξει την εφαρμογή για ευπάθειες. Αυτό θα μπορούσε να περιλαμβάνει οτιδήποτε, από τη χρήση τεχνικών έγχυσης με brute force όπως το "fuzzing", έως την αναζήτηση περιπτώσεων όπου οι λεπτομέρειες σύνδεσης χρήστη αντιμετωπίζονται με μη ασφαλή τρόπο.

Ο αυτοματοποιημένος σαρωτής του OWASP Zap είναι σε θέση να εντοπίσει μια μεγάλη λίστα ευπαθειών ασφαλείας - πολλές περιπτώσεις για τις οποίες δεν θα αναφέρονταν από ένα συμβατικό DAST. Αυτές οι αυξημένες δυνατότητες έρχονται χάρη στην είσοδο από τις τεχνικές IAST (interactive application security testing) και OAST (out-of-band application security testing) τεχνικές.

4.2.2.2 Μη Αυτοματοποιημένο DAST

Κανένας αυτοματοποιημένος σαρωτής ευπαθειών δεν βρίσκει κάθε σφάλμα. Ενώ το αυτοματοποιημένο λογισμικό εξοικονομεί χρόνο σε penetration testers και bug hunters, υπάρχουν ορισμένες καταστάσεις όπου η ανθρώπινη δημιουργικότητα και σκέψη είναι αναντικατάστατες.

Συχνά, ένας penetration tester θα χρησιμοποιεί πρώτα μια αυτοματοποιημένη λύση DAST, για τη συγκομιδή «low hanging fruits». Αυτή η προσέγγιση ελευθερώνει επιπλέον χρόνο για να εργαστούν σε πιο ενδιαφέροντα τρωτά σημεία. Αυτός είναι ο λόγος για τον οποίο, εκτός από το Zap Scanner, ο OWASP προσφέρει επίσης έναν ισχυρό διακομιστή μεσολάβησης προσαρμοσμένο στις ανάγκες των μη αυτόματων ελεγκτών ασφάλειας ιστού.

Ένας intercepting proxy είναι μια αρκετά απλή έννοια. Στην περίπτωση του OWASP Zap, περιλαμβάνει ένα λογισμικό που παρακολουθεί όλη την κυκλοφορία HTTP μεταξύ του προγράμματος περιήγησης του tester και της στοχευμένης εφαρμογής ιστού. Το Zap θα το κάνει ακόμη και για HTTPS (κρυπτογραφημένη) κυκλοφορία. Η δυνατότητα ανάγνωσης όλων των επικοινωνιών που αποστέλλονται μεταξύ μιας εφαρμογής ιστού και του προγράμματος περιήγησής σας είναι ανεκτίμητη στο περιβάλλον DAST.

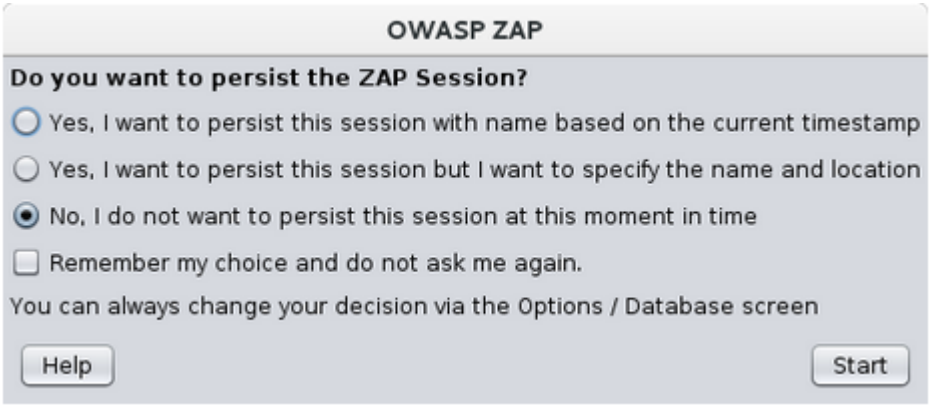
Χρησιμοποιώντας την intercepting proxy προσέγγιση, ένας tester μπορεί να αλλάξει το αίτημα που αποστέλλεται σε έναν διακομιστή από το πρόγραμμα περιήγησής του - ανοίγοντας μια πληθώρα ευκαιριών για να διερευνηθούν ευπάθειες. [88]

4.2.2 Χρήση OWASP Zap

Το Zap είναι προεγκατεστημένο σε Kali Linux. Αν επιθυμεί ο χρήστης να το εγκαταστήσει σε άλλο λειτουργικό μπορεί να το κατεβάσει από εδώ.

Όταν ξεκινήσει για πρώτη φορά το ZAP, θα ερωτηθεί ο χρήστης εάν θέλει να διατηρήσει τη συνεδρία ZAP. Από προεπιλογή, οι περίοδοι σύνδεσης ZAP καταγράφονται πάντα σε δίσκο σε μια βάση δεδομένων HSQLDB με προεπιλεγμένο όνομα και τοποθεσία. Εάν δεν διατηρηθεί η συνεδρία, αυτά τα αρχεία διαγράφονται όταν γίνει έξοδος από το ZAP.

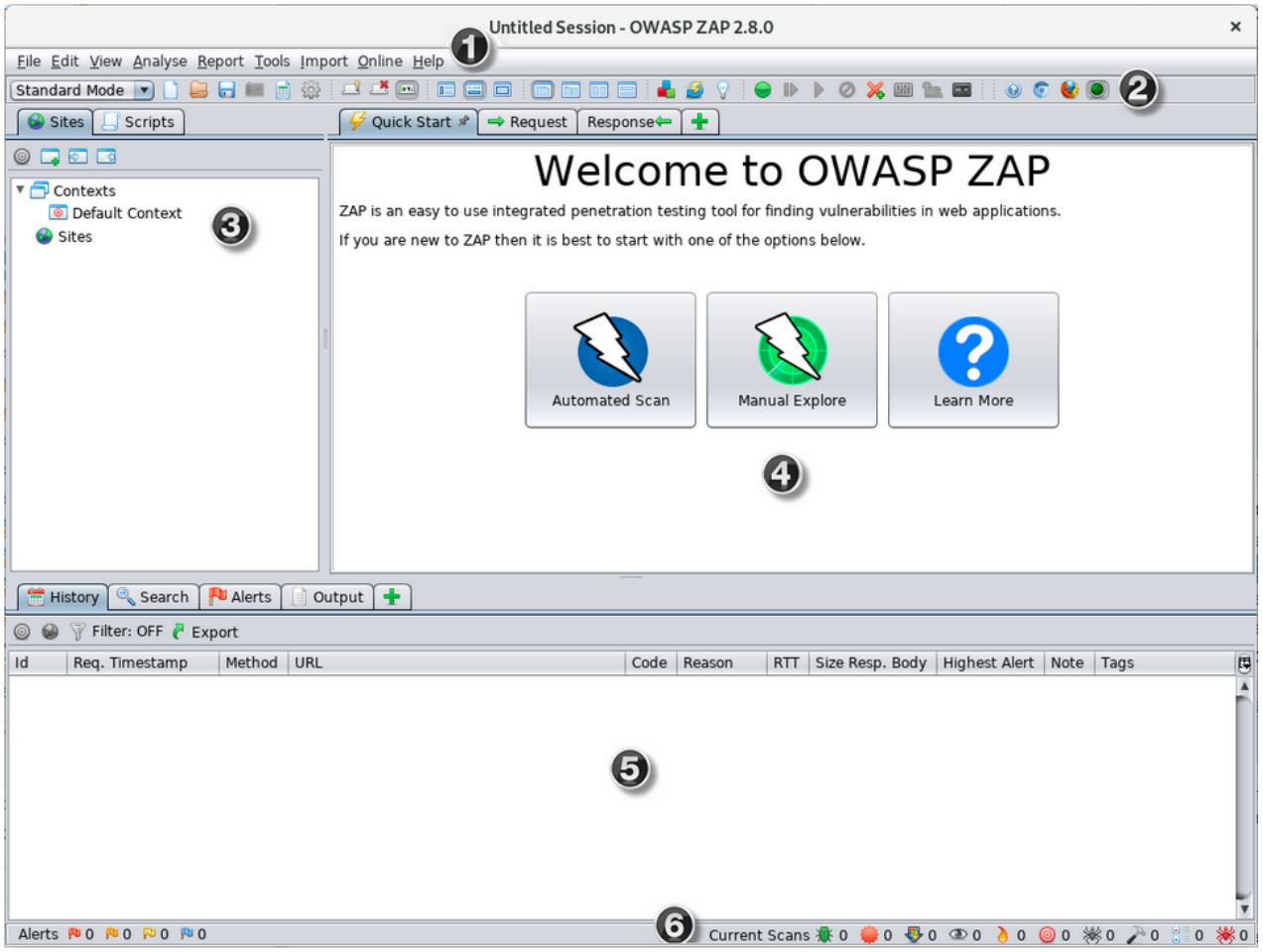
Εάν γίνει επιλογή να διατηρηθεί μια περίοδος σύνδεσης, οι πληροφορίες περιόδου λειτουργίας θα αποθηκευτούν στην τοπική βάση δεδομένων, ώστε να μπορεί ο χρήστης να έχει πρόσβαση σε αυτήν αργότερα.



Το περιβάλλον εργασίας χρήστη ZAP Desktop αποτελείται από τα ακόλουθα στοιχεία:

1. **Γραμμή μενού -** Παρέχει πρόσβαση σε πολλά από τα αυτοματοποιημένα και χειροκίνητα εργαλεία.
2. **Γραμμή εργαλείων -** Περιλαμβάνει κουμπιά που παρέχουν εύκολη πρόσβαση σε λειτουργίες που χρησιμοποιούνται πιο συχνά.
3. **Παράθυρο Δέντρου -** Εμφανίζει το δέντρο Sites και το δέντρο Scripts.
4. **Παράθυρο χώρου εργασίας -** Εμφανίζει αιτήματα, απαντήσεις και σενάρια και επιτρέπει την επεξεργασία τους.
5. **Παράθυρο πληροφοριών -** Εμφανίζει λεπτομέρειες των αυτοματοποιημένων και μη αυτόματων εργαλείων.
6. **Υποσέλιδο -** Εμφανίζει μια περίληψη των ειδοποιήσεων που βρέθηκαν και την κατάσταση των κύριων αυτοματοποιημένων εργαλείων.

[88] "Dynamic Application Security Testing (DAST) - PortSwigger." https://portswigger.net/burp/application-security-testing/dast. Πρόσβαση στις 4 Ιαν. 2021.

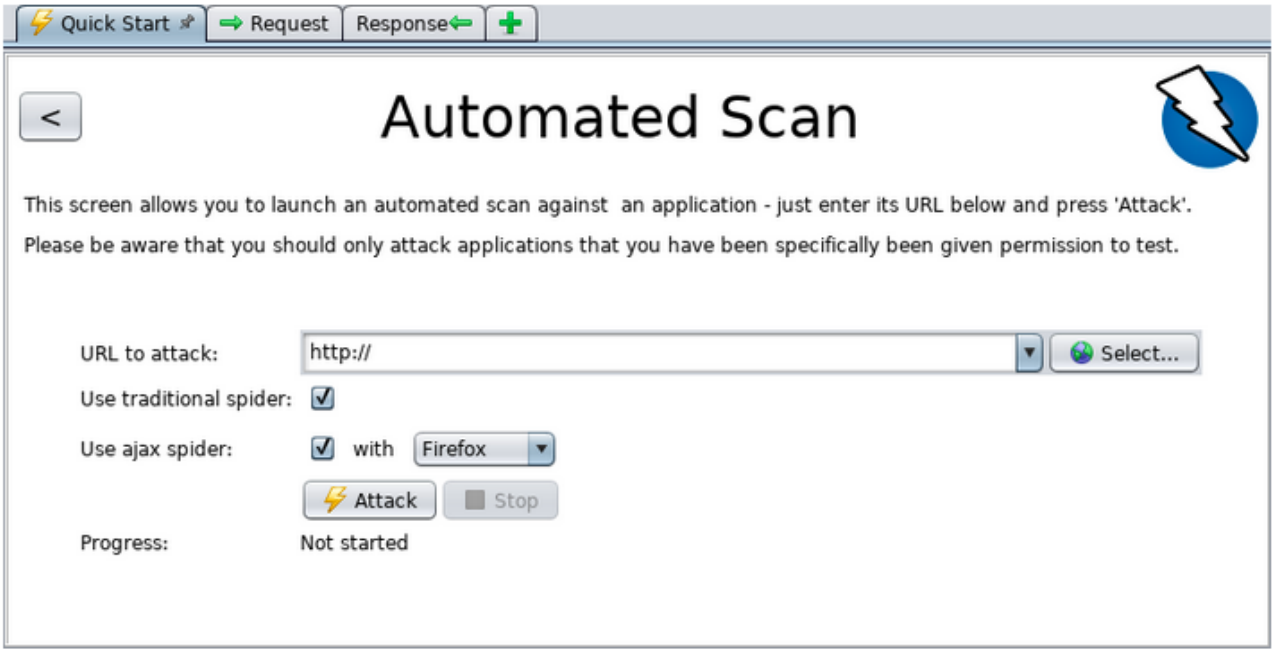


4.2.2.1 Αυτοματοποιημένη Σάρωση

Ο ευκολότερος τρόπος για να να χρησιμοποιήσει κανείς το ZAP είναι μέσω της καρτέλας Γρήγορη εκκίνηση. Το Quick Start είναι ένα πρόσθετο ZAP που περιλαμβάνεται αυτόματα κατά την εγκατάσταση του ZAP.

Για να εκτελεστεί μια αυτόματη σάρωση γρήγορης εκκίνησης πρέπει ο χρήστης να:

1. Ξεκινήσει το ZAP και να κάνει κλικ στην καρτέλα Γρήγορη εκκίνηση του παραθύρου χώρου εργασίας.
2. Κάνει κλικ στο μεγάλο κουμπί αυτόματης σάρωσης.
3. Στο πλαίσιο κειμένου URL για επίθεση, να εισάγει το πλήρες URL της εφαρμογής ιστού που θέλει να επιτεθεί.
4. Κλικ στο Attack.



Το ZAP θα προχωρήσει στο crawl της εφαρμογής ιστού με την αράχνη της και θα σαρώσει παθητικά κάθε σελίδα που βρίσκει. Στη συνέχεια, το ZAP θα χρησιμοποιήσει τον ενεργό σαρωτή για να επιτεθεί σε όλες τις αποκαλυφθείσες σελίδες, τη λειτουργικότητα και τις παραμέτρους.

Το ZAP παρέχει 2 αράχνες για crawl εφαρμογών ιστού, μπορεί να χρησιμοποιηθεί το ένα ή και τα δύο.

Η παραδοσιακή αράχνη ZAP ανακαλύπτει συνδέσμους εξετάζοντας το HTML σε responses από την εφαρμογή ιστού. Αυτή η αράχνη είναι γρήγορη, αλλά δεν είναι πάντα αποτελεσματική κατά την εξερεύνηση μιας διαδικτυακής εφαρμογής AJAX που δημιουργεί συνδέσμους χρησιμοποιώντας JavaScript.

Για εφαρμογές AJAX, η αράχνη AJAX του ZAP είναι πιθανό να είναι πιο αποτελεσματική. Αυτή η αράχνη εξερευνά την διαδικτυακή εφαρμογή κάνοντας χρήση προγραμμάτων περιήγησης που ακολουθούν τους συνδέσμους που έχουν δημιουργηθεί. Η αράχνη AJAX είναι πιο αργή από την παραδοσιακή αράχνη και απαιτεί επιπρόσθετη διαμόρφωση για χρήση σε περιβάλλον «headless».

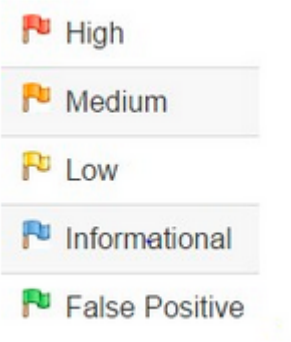
Το ZAP θα σαρώσει παθητικά όλα τα αιτήματα και τις απαντήσεις που παρέχονται μέσω αυτού. Μέχρι στιγμής το ZAP έχει πραγματοποιήσει μόνο παθητικές σαρώσεις της εφαρμογής ιστού. Η παθητική σάρωση δεν αλλάζει καθόλου τις απαντήσεις και θεωρείται ασφαλής. Η σάρωση πραγματοποιείται επίσης σε thread παρασκηνίου για να μην επιβραδύνει την εξερεύνηση. Η παθητική σάρωση είναι καλή για την εύρεση ορισμένων τρωτών σημείων και ως τρόπου για να αποκτηθεί μια αίσθηση για τη βασική κατάσταση ασφάλειας μιας εφαρμογής ιστού και να εντοπιστεί πού μπορεί να απαιτηθεί περισσότερη έρευνα.

Η ενεργή σάρωση, ωστόσο, επιχειρεί να βρει άλλες ευπάθειες χρησιμοποιώντας γνωστές επιθέσεις εναντίον των επιλεγμένων στόχων. Η ενεργή σάρωση είναι μια πραγματική επίθεση σε αυτούς τους στόχους και μπορεί να θέσει τους στόχους σε κίνδυνο, οπότε δεν πρέπει να χρησιμοποιείται ενεργή σάρωση σε στόχους που δεν υπάρχει άδεια.

Καθώς το ZAP κάνει spider την εφαρμογή ιστού, δημιουργεί έναν χάρτη των σελίδων των εφαρμογών ιστού και των πόρων που χρησιμοποιούνται για το render αυτών των σελίδων. Στη συνέχεια, καταγράφει τα αιτήματα και τις απαντήσεις που αποστέλλονται σε κάθε σελίδα και δημιουργεί ειδοποιήσεις εάν υπάρχει κάτι πιθανόν λάθος με ένα αίτημα ή απάντηση.

Για να εξεταστεί μια ιεραρχημένη προβολή των εξερευνημένων σελίδων, πρέπει να γίνει κλικ στην καρτέλα Ιστότοποι στο Παράθυρο δέντρου. Μπορεί να γίνει επέκταση των κόμβων για να γίνουν ορατές οι μεμονωμένες διευθύνσεις URL.

Η αριστερή πλευρά του υποσέλιδου περιέχει μια μέτρηση των Ειδοποιήσεων που βρέθηκαν κατά τη δοκιμή, χωρισμένες σε κατηγορίες κινδύνου. Αυτές οι κατηγορίες κινδύνου είναι:



Για να δει ο χρήστης τις ειδοποιήσεις που δημιουργήθηκαν κατά τη διάρκεια της δοκιμής πρέπει:

1. Να κάνει κλικ στην καρτέλα Ειδοποιήσεις στο παράθυρο πληροφοριών.
2. Να κάνει κλικ σε κάθε ειδοποίηση που εμφανίζεται σε αυτό το παράθυρο για να εμφανιστεί η διεύθυνση URL και η ευπάθεια που εντοπίστηκε στη δεξιά πλευρά του παραθύρου πληροφοριών.
3. Στο Παράθυρο χώρου εργασίας, να γίνει κλικ στην καρτέλα Response για να εμφανιστούν τα περιεχόμενα της κεφαλίδας και του σώματος του Response. Θα επισημανθεί το μέρος του Respnose που δημιούργησε την ειδοποίηση.

4.2.2.2 Μη Αυτοματοποιημένη Σάρωση

Η παθητική σάρωση και η λειτουργία αυτοματοποιημένης επίθεσης είναι ένας πολύ καλός τρόπος για να ξεκινήσει κάποιος μια αξιολόγηση ευπαθειών μιας εφαρμογής ιστού, αλλά έχει ορισμένους περιορισμούς. Μεταξύ αυτών είναι:

- Τυχόν σελίδες που προστατεύονται από μια σελίδα σύνδεσης δεν είναι δυνατό να εντοπιστούν κατά τη διάρκεια μιας παθητικής σάρωσης, επειδή, εκτός εάν έχει ρυθμιστεί η λειτουργία ελέγχου ταυτότητας του ZAP, το ZAP δεν θα χειριστεί τον απαιτούμενο έλεγχο ταυτότητας.

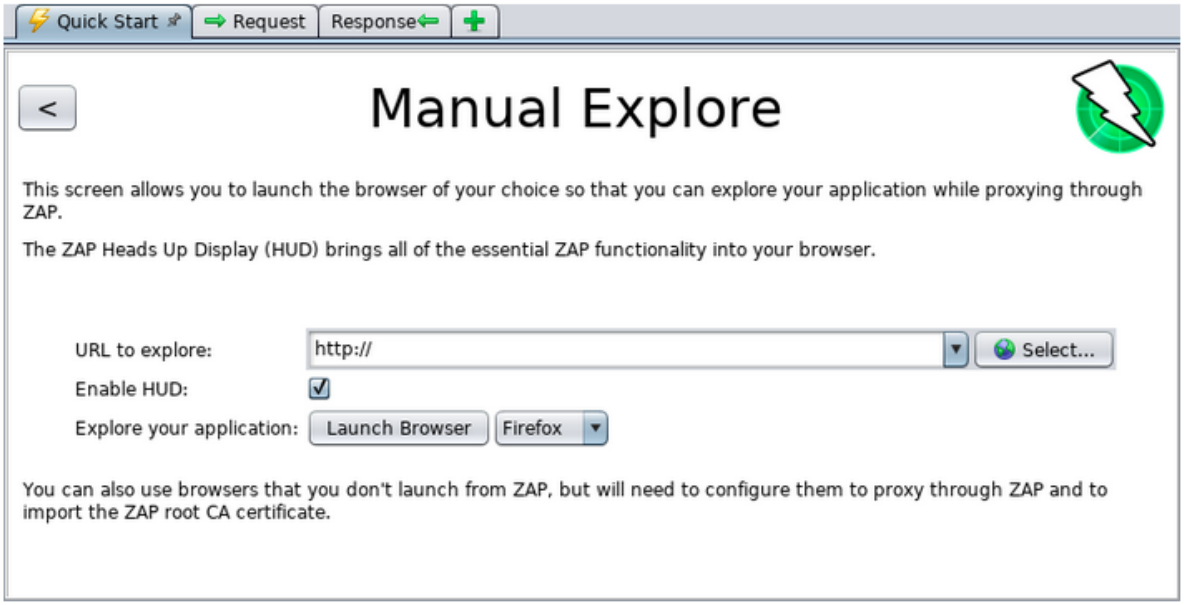
- Δεν υπάρχει μεγάλος έλεγχος της ακολουθίας της εξερεύνησης σε μια παθητική σάρωση ή των τύπων επιθέσεων που πραγματοποιούνται σε μια αυτοματοποιημένη επίθεση. Το ZAP παρέχει πολλές επιπλέον επιλογές για εξερεύνηση και επιθέσεις εκτός της παθητικής σάρωσης.

Οι αράχνες είναι ένας πολύ καλός τρόπος για να εξερευνήσει κάποιος τον βασικό ιστότοπό, αλλά θα πρέπει να συνδυάζονται με τη χειροκίνητη εξερεύνηση για να είναι πιο αποτελεσματικές. Οι αράχνες, για παράδειγμα, θα εισάγουν βασικά προεπιλεγμένα δεδομένα σε φόρμες στην εφαρμογή ιστού, αλλά ένας χρήστης μπορεί να εισαγάγει πιο σχετικές πληροφορίες που μπορούν, με τη σειρά τους, να εκθέσουν περισσότερο την εφαρμογή ιστού στο ZAP. Αυτό ισχύει ιδιαίτερα για πράγματα όπως φόρμες εγγραφής όπου απαιτείται έγκυρη διεύθυνση email. Η αράχνη μπορεί να εισαγάγει μια τυχαία συμβολοσειρά, η οποία θα προκαλέσει σφάλμα. Ένας χρήστης θα είναι σε θέση να αντιδράσει σε αυτό το σφάλμα και να παρέχει μια σωστά μορφοποιημένη συμβολοσειρά, η οποία μπορεί να προκαλέσει την έκθεση περισσότερων από την εφαρμογή όταν υποβληθεί και αποδεχθεί η φόρμα.

Θα πρέπει να γίνει εξερεύνηση όλης την εφαρμογή ιστού με τον διακομιστή μεσολάβησης του ZAP. Καθώς γίνεται αυτό, το ZAP σαρώνει παθητικά όλα τα αιτήματα και τις απαντήσεις που έγιναν κατά τη διάρκεια της εξερευνησης για ευπάθειες, συνεχίζει να δημιουργεί το δέντρο τοποθεσιών και καταγράφει ειδοποιήσεις για πιθανές ευπάθειες που βρέθηκαν κατά τη διάρκεια της εξερεύνησης.

Είναι σημαντικό να ζητηθεί από το ZAP να εξερευνήσει κάθε σελίδα της εφαρμογής ιστού, είτε συνδέεται με άλλη σελίδα είτε όχι, για ευπάθειες. Η απόκρυψη δεν είναι ασφάλεια και οι κρυμμένες σελίδες μερικές φορές μεταδίδονται ζωντανά χωρίς προειδοποίηση ή ειδοποίηση. Γι 'αυτό πρέπει ο χρήστης να είναι όσο πιο προσεκτικός μπορεί κατά την εξερεύνηση του ιστότοπού.

Ο χρήστης μπορεί επίσης να ξεκινήσει γρήγορα και εύκολα προγράμματα περιήγησης που είναι προεπιλεγμένα για διακομιστή μεσολάβησης μέσω ZAP μέσω της καρτέλας Γρήγορη εκκίνηση. Τα προγράμματα περιήγησης που ξεκίνησαν με αυτόν τον τρόπο θα αγνοήσουν επίσης τυχόν προειδοποιήσεις επικύρωσης πιστοποιητικών που διαφορετικά θα αναφερόταν.



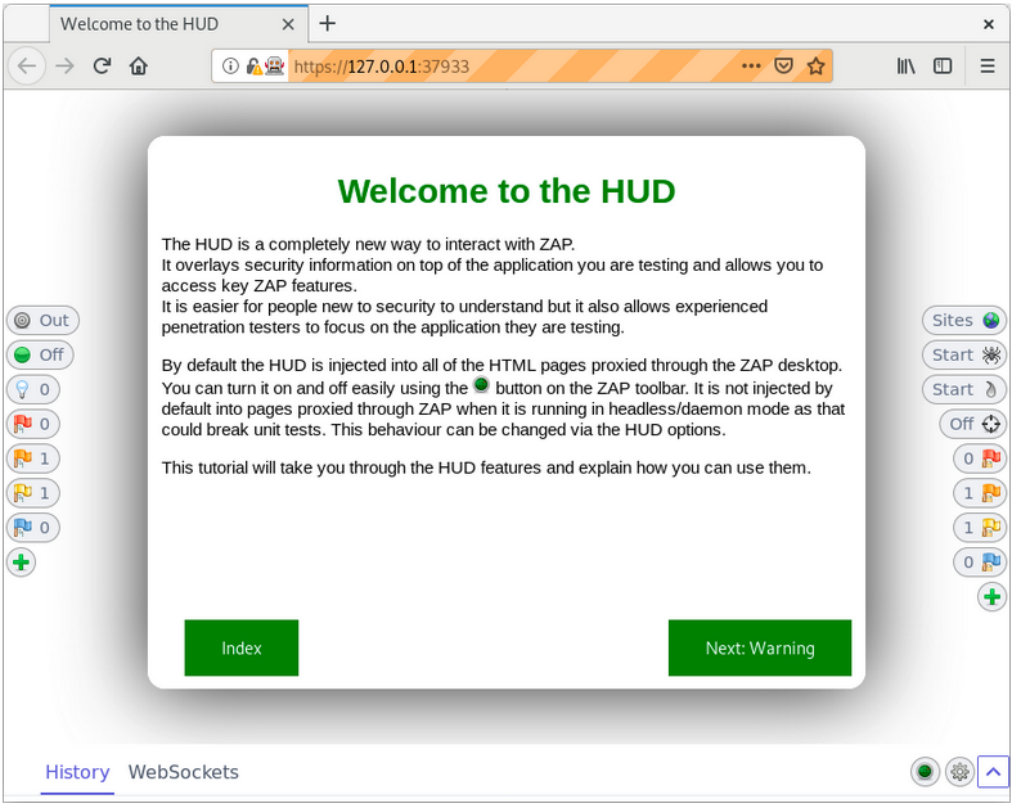
Για να εξερευνήσει ο χρήστης μη αυτόματα την εφαρμογή πρέπει:

- Να ξεκινήσει το ZAP και να κάνει κλικ στην καρτέλα Γρήγορη εκκίνηση του παραθύρου χώρου εργασίας.
- Να κάνει κλικ στο μεγάλο κουμπί Μη αυτόματης Εξερεύνησης.
- Στο πλαίσιο κειμένου URL για εξερεύνηση, να εισάγει το πλήρες URL της εφαρμογής ιστού που θέλει να εξερευνήσει.
- Να επιλέξει το πρόγραμμα περιήγησης που θέλει να χρησιμοποιήσει.
- Να κάνει κλικ στο Εκκίνηση προγράμματος περιήγησης

Αυτή η επιλογή θα ξεκινήσει οποιοδήποτε από τα πιο κοινά προγράμματα περιήγησης που έχει εγκαταστήσει με νέα προφίλ.

Εάν θέλει ο χρήστης να χρησιμοποιήσει οποιοδήποτε από τα προγράμματα περιήγησής σας με υπάρχον προφίλ, για παράδειγμα με εγκατεστημένα άλλα πρόσθετα προγράμματος περιήγησης, τότε θα πρέπει να διαμορφώσει με μη αυτόματο τρόπο το πρόγραμμα περιήγησής με διακομιστή μεσολάβησης μέσω ZAP και να εισάγει και να εμπιστευτεί το πιστοποιητικό ZAP Root CA.

Από προεπιλογή, το ZAP Heads Up Display (HUD) θα ενεργοποιηθεί. Εάν καταργήσει ο χρήστης την επιλογή της σχετικής επιλογής σε αυτήν την οθόνη πριν ξεκινήσει ένα πρόγραμμα περιήγησης, θα απενεργοποιηθεί το HUD.
Το Heads Up Display (HUD) είναι μια νέα, καινοτόμος διεπαφή που παρέχει πρόσβαση στη λειτουργικότητα ZAP απευθείας στο πρόγραμμα περιήγησης. Είναι ιδανικό για άτομα που είναι νέοι στην ασφάλεια του διαδικτύου και επιτρέπει επίσης στους έμπειρους penetration testers να επικεντρώνονται σε μια λειτουργικότητα εφαρμογών παρέχοντας παράλληλα βασικές πληροφορίες ασφάλειας και λειτουργικότητα.



Το HUD επικαλύπτεται στην κορυφή της εφαρμογής στόχου στο πρόγραμμα περιήγησής όταν ενεργοποιείται μέσω της οθόνης "Μη αυτόματη εξερεύνηση" ή της γραμμής εργαλείων. Υποστηρίζονται μόνο σύγχρονα προγράμματα περιήγησης όπως ο Firefox και το Chrome.

Από προεπιλογή, εμφανίζεται μια οθόνη εκκίνησης για το HUD που περιλαμβάνει έναν σύνδεσμο προς ένα tutorial που θα καθοδηγήσει τον χρήστη στις δυνατότητες HUD και θα εξηγήσει πώς μπορεί ο χρήστης να τα χρησιμοποιήσει.

5.0 Συμπεράσματα

Από την παρούσα εργασία μπορεί κάποιος να καταλάβει πόσο περίπλοκη και απαιτητική είναι η ασφάλεια. Οι μεταβλητές που πρέπει να ληφθούν υπόψη είναι πάρα πολλές και είναι αδύνατο να προληφθούν όλοι οι κίνδυνοι. Αν όμως ακολουθηθούν οι βασικές αρχές και υπάρχει συνεχής επένδυση στην ασφάλεια είναι βέβαιο ότι μια επιχείρηση μπορεί να διασώσει πολύτιμες πληροφορίες από κακόβουλους χρήστες. Είναι ένας αγώνας που δεν έχει τέλος αλλά είναι ζωτικής σημασίας για την ελαχιστοποίηση του κινδύνου και για τους καταναλωτές αλλά και τις επιχειρήσεις.

Βιβλιογραφία

[1] "Security Through Obscurity - SecurityTrails." https://securitytrails.com/blog/security-through-obscurity
[2] "Securing Enterprise Web Applications at the Source - OWASP ...." https://owasp.org/www-pdf-archive/Securing_Enterprise_Web_Applications_at_the_Source.pdf . Σελ 19-20
[3] "Securing Enterprise Web Applications at the Source - OWASP ...." https://owasp.org/www-pdf-archive/Securing_Enterprise_Web_Applications_at_the_Source.pdf . Σελ 21-22
[4] "Securing Enterprise Web Applications at the Source - OWASP ...." https://owasp.org/www-pdf-archive/Securing_Enterprise_Web_Applications_at_the_Source.pdf . Σελ 137-138
[5] "Fail Securely | OWASP." https://owasp.org/www-community/Fail_securely .
[6] "Defence in depth and how it applies to web applications ...." https://www.acunetix.com/websitesecurity/defence-in-depth-and-how-it-applies-to-web-applications/ .
[7] "The Dangers of Trusting User Input - enisa - europa.eu." https://www.enisa.europa.eu/publications/info-notes/the-dangers-of-trusting-user-input .
[8] "Reducing your attack surface | Digital Shadows.", https://www.digitalshadows.com/blog-and-research/reducing-your-attack-surface/ .
[9] "Logging - OWASP Cheat Sheet Series." https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html .
[10] "Defence in depth and how it applies to web applications ...." https://www.acunetix.com/websitesecurity/defence-in-depth-and-how-it-applies-to-web-applications/ . (KISS Principle)
[11] "Separation of duties - OWASP - Linux." https://www.linuxsecrets.com/owasp-wiki/index.php/Separation_of_duties.html .
[12] "Security by Design Principles according to OWASP." https://blog.threatpress.com/security-design-principles-owasp/ .
[13] "Promote Security throughout your Software Design phase ...." https://www.advantio.com/blog/secure-software-development-life-cycle-design-phase .
[14] "OWASP TOP 10: Sensitive Data Exposure | Detectify Blog." https://blog.detectify.com/2016/07/01/owasp-top-10-sensitive-data-exposure-6/ .
[15] "C10: Handle all Errors and Exceptions | OWASP." https://owasp.org/www-project-proactive-controls/v3/en/c10-errors-exceptions .
[16] "Denial of Service - OWASP Cheat Sheet Series." https://cheatsheetseries.owasp.org/cheatsheets/Denial_of_Service_Cheat_Sheet.html .
[17] "1. (Miscel) What is the difference between fault avoidance and ...." https://www.csm.ornl.gov/~sheldon/cs422/solns/Ch1-9StudyQsKey.pdf .
[18] "Why is loose coupling between services so ... - Ben Morris." https://www.ben-morris.com/why-is-loose-coupling-between-services-so-important/ .
[19] "Low Coupling, High Cohesion. The key to creating ... - Medium." https://medium.com/clarityhub/low-coupling-high-cohesion-3610e35ac4a6 .
[20] "The OWASP Foundation OWASP Technology and Business ...." https://owasp.org/www-pdf-archive//Technology_and_Business_Risk_Management_How_Application_Security_Fits_In.pdf .
[21] "Security in Software Development and Infrastructure System ...." https://medium.com/cermati-tech/security-in-software-development-and-infrastructure-system-design-7b675c2323fc .
[22] "Information Systems in Your Life - GitHub Pages." https://saylordotorg.github.io/text_business-information-systems-design-an-app-for-that/s05-information-systems-in-your-li.html .
[23] "What are the Software Development Life Cycle (SDLC) phases?." https://www.linkedin.com/pulse/what-software-development-life-cycle-sdlc-phases-private-limited .
[24] "Input Validation - OWASP Cheat Sheet Series." https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html .
[25] "Data Validation - Teach Computer Science." https://teachcomputerscience.com/validation/ .
[26] "Validation Using a Lookup Table | Importing and ... - Flylib.com." https://flylib.com/books/en/2.305.1/validation_using_a_lookup_table.html .
[27] "Input Mask Format - Formidable Forms." 9 Ιουλ. 2020, https://formidableforms.com/knowledgebase/format/ .
[28] "Open Source Validation Frameworks - Java-Source.net." https://java-source.com/open-source/validation .
[29] "Define and Map Filters - Rogue Wave - Documentation." https://docs.roguewave.com/en/hydraexpress/4.3.0/html/rwsfexpservletug/4-8.html .
[31] "Struts 2 Custom Validation - Workflow Interceptor - javatpoint." https://www.javatpoint.com/struts-2-custom-validation-workflow-interceptor .
[32] "Struts 2 Validation by bundled validators - javatpoint." https://www.javatpoint.com/struts-2-validation-by-bundled-validators .
[33] "Spring MVC Tutorial - javatpoint." https://www.javatpoint.com/spring-mvc-tutorial .
[34] "JSF f:validateBean - javatpoint." https://www.javatpoint.com/jsf-validatebean .
[35] "Single Sign-on Using Kerberos in Java - Oracle Help Center." https://docs.oracle.com/javase/10/security/single-sign-using-kerberos-java1.htm .
[36] "What is Discretionary Access Control (DAC)? - Techopedia." https://www.techopedia.com/definition/229/discretionary-access-control-dac .
[37] "What is mandatory access control (MAC)? - SearchSecurity." https://searchsecurity.techtarget.com/definition/mandatory-access-control-MAC .

[38] "What is role-based access control (RBAC)? - Definition from ...." https://searchsecurity.techtarget.com/definition/role-based-access-control-RBAC .
[39] "Java Authentication and Authorization Service (JAAS ...." https://docs.oracle.com/javase/10/security/java-authentication-and-authorization-service-jaas-reference-guide.htm .
[40] "Spring Security." https://spring.io/projects/spring-security .
[41] "Spring Authentication - Cheng." 31 Ιουλ. 2019, https://chengyu.home.blog/2019/07/31/spring-authentication/ .
[42] "下一步Storing Username and Password - Cheng." https://chengyu.home.blog/2019/07/31/storing-username-and-password /.
[43] "Configuring Anonymous Login - Cheng."https://chengyu.home.blog/2019/07/31/configuring-anonymous-login/.
[44] "JSF Primefaces Spring Security Example - CodeNotFound.com ." https://codenotfound.com/jsf-primefaces-spring-security-example.html .
[45] "Cipher | cryptology | Britannica." https://www.britannica.com/topic/cipher .
[46] "Digital signature - Wikipedia." https://en.wikipedia.org/wiki/Digital_signature .
[47] "confidentiality, integrity, and availability (CIA triad) - WhatIs.com." https://whatis.techtarget.com/definition/Confidentiality-integrity-and-availability-CIA .
[48] "Confidentiality, Integrity and Availability - The CIA Triad ...." https://www.certmike.com/confidentiality-integrity-and-availability-the-cia-triad/ .
[49] "Java Cryptography Architecture (JCA) Reference Guide." https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html .
[50] "Java Cryptography Extension (JCE) - IBM Knowledge Center." https://www.ibm.com/support/knowledgecenter/SSYKE2_7.1.0/com.ibm.java.security.component.71.doc/security-component/JceDocs/jce.html .
[52] "Java Cryptography - Message Digest - Tutorialspoint." https://www.tutorialspoint.com/java_cryptography/java_cryptography_message_digest.htm .
[53] "Java Cryptography - Creating a MAC - Tutorialspoint." https://www.tutorialspoint.com/java_cryptography/java_cryptography_creating_mac.htm .
[54] "Java Cryptography - Creating Signature - Tutorialspoint." https://www.tutorialspoint.com/java_cryptography/java_cryptography_creating_signature.htm
[55] "Java Cryptography - Verifying Signature - Tutorialspoint." https://www.tutorialspoint.com/java_cryptography/java_cryptography_verifying_signature.htm .
[56] "HttpSession with example in Servlet - BeginnersBook.com." https://beginnersbook.com/2013/05/http-session/ .
[57] "Cookies in Servlet - javatpoint." https://www.javatpoint.com/cookies-in-servlet .
[58] "URL Rewriting in Servlet - javatpoint." https://www.javatpoint.com/url-rewriting-in-session-tracking .
[59] "Hidden Form Field in Servlet - javatpoint." https://www.javatpoint.com/hidden-form-field-in-session-tracking .
[60] "Using Http Session With Spring Based Web ... - DZone." 29 Απρ. 2014, https://dzone.com/articles/using-http-session-spring .
[61] "Control the Session with Spring Security | Baeldung." 15 Αυγ. 2020, https://www.baeldung.com/spring-security-session .
[62] "Session hijacking attack Software Attack | OWASP Foundation." https://owasp.org/www-community/attacks/Session_hijacking_attack .
[63] "Session fixation Software Attack | OWASP Foundation." https://owasp.org/www-community/attacks/Session_fixation .
[64] "HTTP Header Injection | GracefulSecurity." 7 Μαρ. 2016, https://gracefulsecurity.com/http-header-injection/ .
[65] "What Is Session Fixation | Acunetix." 12 Δεκ. 2019, https://www.acunetix.com/blog/web-security-zone/what-is-session-fixation/
[66] "(PDF) Session Hijacking: Threat Analysis and Countermeasures." 28 Μαΐ. 2015, https://www.researchgate.net/publication/277307339_Session_Hijacking_Threat_Analysis_and_Countermeasures .
[68] "Checked vs Unchecked Exceptions in Java - GeeksforGeeks." https://www.geeksforgeeks.org/checked-vs-unchecked-exceptions-in-java/ .
[69] "How to print an exception stack trace using Log4J (or ...." 5 Φεβ. 2017, https://alvinalexander.com/blog/post/java/how-print-exception-stack-trace-using-log4j-commons/ .
[70] "Java Exception Handling: How to Specify and ... - Stackify." 17 Ιουλ. 2017, https://stackify.com/specify-handle-exceptions-java/
[71] "ERR00-J. Do not suppress or ignore checked ... - Confluence." https://wiki.sei.cmu.edu/confluence/display/java/ERR00-J.+Do+not+suppress+or+ignore+checked+exceptions .
[72] "ERR01-J. Do not allow exceptions to expose ... - Confluence." https://wiki.sei.cmu.edu/confluence/display/java/ERR01-J.+Do+not+allow+exceptions+to+expose+sensitive+information .
[73] "ERR02-J. Prevent exceptions while logging data - Confluence ...." https://wiki.sei.cmu.edu/confluence/display/java/ERR02-J.+Prevent+exceptions+while+logging+data .
[74] "ERR03-J. Restore prior object state on method ... - Confluence." https://wiki.sei.cmu.edu/confluence/display/java/ERR03-J.+Restore+prior+object+state+on+method+failure .
[75] "ERR04-J. Do not complete abruptly from a finally ... - Confluence." https://wiki.sei.cmu.edu/confluence/display/java/ERR04-J.+Do+not+complete+abruptly+from+a+finally+block .
[76] "ERR05-J. Do not let checked exceptions escape ... - Confluence." https://wiki.sei.cmu.edu/confluence/display/java/ERR05-J.+Do+not+let+checked+exceptions+escape+from+a+finally+block .
[77] "ERR06-J. Do not throw undeclared checked ... - Confluence." https://wiki.sei.cmu.edu/confluence/display/java/ERR06-J.+Do+not+throw+undeclared+checked+exceptions .
[78] "ERR07-J. Do not throw RuntimeException ... - Confluence." https://wiki.sei.cmu.edu/confluence/display/java/ERR07-J.+Do+not+throw+RuntimeException%2C+Exception%2C+or+Throwable
[79] "ERR08-J. Do not catch NullPointerException or any of its ...." https://wiki.sei.cmu.edu/confluence/display/java/ERR08-J.+Do+not+catch+NullPointerException+or+any+of+its+ancestors .

[80] "ERR09-J. Do not allow untrusted code to terminate the JVM." https://wiki.sei.cmu.edu/confluence/display/java/ERR09-J.+Do+not+allow+untrusted+code+to+terminate+the+JVM .
[81] "JSF Custom Error Pages - JavaBeat." https://javabeat.net/jsf-custom-error-pages/ .
[82] "JSF 2 Ajax Error Handling Example - JavaBeat." 2 Απρ. 2014, https://javabeat.net/jsf-2-ajax-error-handling/ .
[83] "Definition of Static Application Security Testing (SAST) - Gartner." https://www.gartner.com/en/information-technology/glossary/static-application-security-testing-sast .
[84] "What Is SAST and How Does Static Code Analysis ... - Synopsys." https://www.synopsys.com/glossary/what-is-sast.html .
[85] "What Is SAST and How Does Static Code Analysis ... - Synopsys." https://www.synopsys.com/glossary/what-is-sast.html .
[86] "What is source code analysis? - Definition from WhatIs.com." https://searchsoftwarequality.techtarget.com/definition/source-code-analysis .
[87] "Using SonarQube to Analyze a Java Project | by ... - Medium." https://medium.com/linagora-engineering/using-sonarqube-to-analyze-a-java-project-abeee15e3779 .
[88] "Dynamic Application Security Testing (DAST) - PortSwigger." https://portswigger.net/burp/application-security-testing/dast .