UNIVERSITY OF PIRAEUS

M. Sc. «Digital Systems Security»

Project:"Web Application Security (HTTP security)"

Όνομα :Γιώργος

Επίθετο: Βαλλάτος

Αριθμός Μητρώου: MTE 14003

Mailto: george.vallatos@gmail

Επιβλέπων Καθηγητής: Κωσταντίνος Λαμπρινουδάκης

Mailto: clam@unipi.gr

Έτος:2016

Περιεχόμενα

Εισαγωγή

Η Ασφάλεια των ηλεκτρονικών υπολογιστών και των εφαρμογών του Διαδικτύου είναι ένα θέμα που ακούμε όλο και περισσότερο τον τελευταίο καιρό. Η παραπάνω συζήτηση φαντάζει όλο και πιο λογική άμα αναλογιστούμε ότι το ηλεκτρονικό έγκλημα ή όπως συνηθίζεται να λέγεται κυβερνο-έγκλημα έχει σημειώσει τρομακτική έξαρση σε παγκόσμια κλίμακα. Οι εκτιμήσεις που έχουμε από την Norton είναι ότι παγκόσμια πάνω από δεκαοχτώ ενήλικες πέφτουν θύματα ηλεκτρονικού εγκλήματος κάθε δευτερόλεπτο ενώ παγκόσμια το 2012 υπήρχαν απώλειες περίπου $110.000.000.000.Αυτές οι εκτιμήσεις και τα νούμερα άμα αναλογιστεί κανένας την όλο και μεγαλύτερη διείσδυση στον παγκόσμιο ιστό(Internet) αλλά και την τάση του εμπορίου και των συναλλαγών να παίρνουν ηλεκτρονική μορφή , παρουσιάζουν μια συνεχόμενη άνοδο.

Μια λανθασμένη άποψη που επικρατεί είναι ότι ο τομέας της Ασφάλειας είναι ότι αφενός αφορά μόνο λίγους ειδικούς αλλά και αφετέρου ότι είναι κάτι που κάποιος χρειάζεται να ασχοληθεί μία φορά μόνο. Η αλήθεια όμως δεν είναι αυτή, διότι ούτε λίγες μετατροπές στον κώδικα ενός προγράμματος δεν εξασφαλίζει για πάντα την ασφάλεια καθώς μέρα με την μέρα οι επιτιθέμενοι βρίσκουν όλο και πιο έξυπνους και στοχευόμενους τρόπους επίθεσης και επίσης το ζήτημα της ασφάλειας είναι κάτι που αφορά από τους προγραμματιστές μέχρι τους χρήστες και τους οργανισμούς και πρέπει ο καθένας από την μεριά του να ξέρει τι μπορεί να κάνει για να προφυλαχτεί.

Αυτή λοιπόν η εργασία έχει ως στόχο την παρουσίαση των συνηθέστερων και πιο επικίνδυνων επιθέσεων που στοχεύουν τις διαδικτυακές εφαρμογές(web applications) από κακόβουλους χρήστες αλλά και τις αντίστοιχες καλές πρακτικές(αντίμετρα) που μπορούμε να λάβουμε με σκοπό την αποτελεσματικότερη αντιμετώπιση τους. Αρχικά θα αποσαφηνίσουμε βασικές έννοιες που σχετίζονται με τις διαδικτυακές εφαρμογές όπως και τα πρωτόκολλα που τις περικλείουν και έπειτα θα αναλύσουμε ποιες είναι οι κυριότερες ευπάθειες που συναντάμε στις διαδικτυακές εφαρμογές, πως ένας κακόβουλος χρήστης μπορεί να τις εκμεταλλευτεί και ποιο είναι το χρέος των σχεδιαστών αυτών των εφαρμογών αλλά και των απλών χρηστών για την αποφυγή ζημιάς είτε από την μεριά αυτού που παρέχει την εφαρμογή είτε από την μεριά του ατόμου που την χρησιμοποιεί. Οι επιθέσεις που θα αναλύσουμε

στηρίζονται στην λίστα του OWASPμε τις κύριες ευπάθειες των διαδικτυακών εφαρμογών. Ο OWASP(Open Web Application Security Project) είναι μία πρωτοβουλία που έχει σαν στόχο την αντιμετώπιση τρωτών σημείων στα λογισμικά των εφαρμογών. Είναι ένας μη κερδοσκοπικός οργανισμός ο οποίος ακολουθεί την ιδεολογία του ανοιχτού/ελεύθερου κώδικα(open source).

Η εργασία αυτή έγινε με κατευθυντήριες σημειώσεις και διαλέξεις του Jim Manico ο οποίος είναι ενεργό μέλος του OWASP από το 2008 , διάφορα επιστημονικά συγγράμματα όπως και άρθρα πάνω στο αντικείμενο της Ασφάλειας.

Κεφάλαιο 1: Προσέγγιση HTTP Security

1.1 Διαδικτυακές Εφαρμογές (Web Applications ή webapp)

Η εργασία αυτή πραγματεύεται την ασφάλεια των διαδικτυακών εφαρμογών, άρα αρχικά πρέπει να προσδιορίσουμε τι ονομάζουμε διαδικτυακές εφαρμογές. Διαδικτυακές εφαρμογές λοιπόν ονομάζονται οι εφαρμογές οι οποίες είναι διαθέσιμες στους χρήστες μέσω του Διαδικτύου(Internet) ή ενός εσωτερικού εταιρικού δικτύου(Intranet).Οι χρήστες αποκτούν πρόσβαση σε αυτές μέσω του προσωπικού τους περιηγητή (browser)(Mozilla Firefox,Google Chrome,Opera,Internet Explorer). Οι εφαρμογές αυτές συνήθως, λόγω υψηλών υπολογιστικών απαιτήσεων εκτελούνται σε ισχυρές υπολογιστικές μηχανές οι οποίες έχουν το ρόλο του σταθμού εξυπηρέτησης και παρέχουν-διαμοιράζουν τις υπηρεσίες της εφαρμογής σε περισσότερους του ενός τελικού χρήστη.

1.2 Εισαγωγή στο TCP/IP πρωτόκολλο

Αφού προσδιορίσαμε ότι οι διαδικτυακές εφαρμογές είναι διαθέσιμες σε εμάς μέσω του διαδικτύου ή ενός εσωτερικού δικτύου το επόμενο θέμα που πρέπει να ασχοληθούμε είναι πως επιτυγχάνεται η επικοινωνία σε αυτό. Η επικοινωνία μεταξύ των υπολογιστών στο διαδίκτυο καθορίζεται μέσα από τα πρωτόκολλα επικοινωνίας.. Ένα πρωτόκολλο επικοινωνίας υπολογιστών (computer communication protocol) είναι ένα σύνολο κανόνων αυστηρά καθορισμένων που περιγράφουν την διαδικασία που πρέπει να ακολουθήσουν οι υπολογιστές για να μπορέσουν να επικοινωνήσουν μεταξύ τους. Ένα από τα πιο σημαντικά πρωτόκολλα επικοινωνίας που χρησιμοποιείται είναι το TCP/IP το οποίο ορίζει την επικοινωνία των υπολογιστών που είναι συνδεδεμένοι στο Διαδίκτυο.

Παρόλο που μιλάμε για μία στοίβα πρωτοκόλλων ονομάζεται TCP/IPαπό τα αρχικά των δύο από τα σημαντικότερα πρωτόκολλα που χρησιμοποιούνται στο

Internet, δηλαδή στο TCP και το IP.Τα αρχικά του TCP/IP σημαίνουν Transmission Control Protocol και Internet Protocol αντίστοιχα , δηλαδή , Πρωτόκολλο Ελέγχου Εκπομπής /Πρωτόκολλο του Internet. Το πρότυπο(standard) αυτό ορίζει το πώς οι ηλεκτρονικές συσκευές, όπως οι υπολογιστές, θα πρέπει να συνδέονται στο Internetκαι πώς θα πρέπει να μεταδίδονται τα δεδομένα(data) ανάμεσά τους. Όπως είπαμε μέσα στο TCP/IPσυμμετέχουν και άλλα πρωτόκολλα για τον χειρισμό της επικοινωνίας των δεδομένων(data communication).Τα πρωτόκολλα που υπάρχουν σε αυτό περιληπτικά είναι:

- Το TCP(Transmission Control Protocol): για επικοινωνία ανάμεσα σε εφαρμογές(application)(εγκατάσταση σύνδεσης πριν την αποστολή).
- Το UDP(User Datagram Protocol): για απλή επικοινωνία ανάμεσα σε εφαρμογές(χωρίς εγκατάσταση σύνδεσης πριν την αποστολή).
- Το IP(Internet Protocol): για επικοινωνία ανάμεσα σε υπολογιστές.
- Το ICMP (Internet Control Message Protocol): για λάθη και στατιστικές.
- Το DHCP (Dynamic Host Configuration Protocol): για δυναμική διευθυσιοδότηση (dynamic addressing).

1.3 Λειτουργίες του πρωτοκόλλου TCP/IP

Το TCP φροντίζει για την επικοινωνία ανάμεσα στο λογισμικό της εφαρμογής, δηλαδή στον φυλλομετρητή (browser) και το λογισμικό του δικτύου μας ενώ το IP πρωτόκολλο φροντίζει για την επικοινωνία με τους άλλους υπολογιστές. Το TCP/IP χρησιμοποιεί 32 bits ή 4 αριθμούς με τιμές ανάμεσα σε 0 και 255 για αν αποδώσει μια διεύθυνση (address) σε έναν υπολογιστή.

1.3.1 Το πρωτόκολλο TCP

Το πρωτόκολλο TCP προορίζεται για επικοινωνία ανάμεσα σε εφαρμογές (applications). Όταν μια εφαρμογή θελήσει να επικοινωνήσει με μια άλλη εφαρμογή μέσω του TCP, στέλνει μια αίτηση επικοινωνίας (communication request). Αυτή η αίτηση θα πρέπει να σταλεί σε μια συγκεκριμένη διεύθυνση. Αφού καθιερωθεί μια χειραψία (handshake) ανάμεσα στις δύο εφαρμογές, το TCP θα καθιερώσει μια ταυτόχρονη αμφίπλευρη (full-duplex) επικοινωνία ανάμεσα στις δύο εφαρμογές. Η ταυτόχρονη αμφίπλευρη (full-duplex) επικοινωνία θα καταλάβει τη γραμμή επικοινωνίας ανάμεσα στους δύο υπολογιστές μέχρι αυτή να κλείσει από μια από τις δύο εφαρμογές. Επίσης το TCP είναι υπεύθυνο για τη διάσπαση των δεδομένων σε IP πακέτα πριν αυτά αποσταλούν καθώς και για την αντίστοιχη συναρμολόγησή τους όταν αυτά φτάσουν στον προορισμό τους Το πρωτόκολλο UDP είναι πολύ παρόμοιο με το TCP με τη διαφορά ότι είναι πιο απλό και λιγότερο αξιόπιστο (reliable) καθώς δεν εγκαθιστά κάποια σύνδεση πριν την αποστολή αλλά τα πακέτα πηγαίνουν αυτοδύναμα στο δίκτυο και όχι σειρά πακέτων όπως στο TCP.

1.3.2 Το πρωτόκολλο IP

Το IP είναι υπεύθυνο για την αποστολή των πακέτων στον παραλήπτη (receiver). Είναι ένα connection-less (χωρίς σύνδεση) πρωτόκολλο επικοινωνίας (communication protocol). Αυτό σημαίνει ότι δεν καταλαμβάνει τη γραμμή επικοινωνίας ανάμεσα σε δύο επικοινωνούντες υπολογιστές. Μ’ αυτόν τον τρόπο το IP ελαττώνει την ανάγκη για γραμμές δικτύωσης. Έτσι, η κάθε γραμμή θα μπορεί να χρησιμοποιηθεί για επικοινωνία ανάμεσα σε πολλούς διαφορετικούς υπολογιστές την ίδια στιγμή.

1.4 Τα πρωτόκολλα HTTP , HTTPS, SSL

Το TCP/IP πρωτόκολλο όπως προείπαμε αποτελεί μια συλλογή από πολλά διαφορετικά πρωτόκολλα επικοινωνίας, τα οποία βασίζονται στα δύο σημαντικότερα, το TCP και το IP. Μερικά πρωτόκολλα από αυτά (που περιλαμβάνονται στο πρωτόκολλο TCP/IP) είναι το HTTP, HTTPS , SSL(TLS η εξέλιξή του):

- HTTP(Hyper Text Transfer Protocol): Το πρωτόκολλο HTTP φροντίζει για την επικοινωνία ανάμεσα σ' έναν Web server και έναν Web browser (φυλλομετρητή). Χρησιμοποιείται για την αποστολή των αιτήσεων (requests) από έναν Web client (browser) σ' έναν Web server καθώς και για την επιστροφή του περιεχομένου (Web content), δηλαδή των ιστοσελίδων (Web pages), από τον server πίσω στον χρήστη (client).
- HTTPS(Secure HTTP):Το πρωτόκολλο HTTPS φροντίζει για την ασφαλή επικοινωνία ανάμεσα σ' έναν Web server και έναν Web browser. Χειρίζεται τυπικά τις συναλλαγές που γίνονται με πιστωτικές κάρτες (credit card transactions) καθώς και μ' άλλα ευαίσθητα δεδομένα.
- SSL(Secure Sockets Layer):Το πρωτόκολλο SSL χρησιμοποιείται για την κωδικοποίηση (κρυπτογράφηση – encryption) των δεδομένων για να είναι έτσι ασφαλής η μεταφορά τους. Το HTTPS στην ουσία είναι το πρωτόκολλο HTTP το οποίο χρησιμοποιεί κρυπτογράφηση SSL.

1.5Χαρακτηριστικά μιας Διαδικτυακής Εφαρμογής (Web Application)

Όπως αναφέρθηκε παραπάνω οι διαδικτυακές εφαρμογές εκτελούνται σε ισχυρές υπολογιστικές μηχανές οι οποίες ονομάζονται web servers και διαμοιράζονται στους χρήστες αυτής μέσω του προσωπικού τους φυλλομετρητή (browser). Οι επικοινωνία λοιπόν μεταξύ του browser των χρηστών και των αντίστοιχων web servers πραγματοποιείτε με διάφορες αιτήσεις (HTTP request) από τον browser προς τον web server όπου ο τελευταίους απαντάει πίσω σε αυτόν. Τα HTTP request τις

περισσότερες φορές υλοποιούνται με τις μεθόδους GET και POST. Τα scripting(σενάρια για τον έλεγχο μίας ή περισσότερων εφαρμογών) μπορεί να είναι είτε από την μεριά του server(server-side)(πχ perl,php,asp,jsp) είτε από την μεριά του πελάτη(client-side)(πχ,javascript, flash,applets).Τα δεδομένα στην συνέχεια δημοσιεύονται-τοποθετούνται(posted) στην δικτυακή εφαρμογή μέσω των HTTPμεθόδων(GET,POST), τα δεδομένα αυτά επεξεργάζονται από το σχετικό script και το αποτέλεσμα "γυρνάει" στον browser του χρήστη.

1.5.1 HTTPPOST και HTTPGET

Οι μέθοδοι get και post χρησιμοποιούνται συχνότερα όταν ένας χρήστης θέλει να πραγματοποιήσει ένα request στον Web Server της εφαρμογής. Η μέθοδος "GET" παραθέτει τις ευαίσθητες πληροφορίες αυθεντικοποίησης στο URL,στον Web Server και στο Proxy Server logs όπως και στην σχετική HTTP κεφαλίδα(header).Η μέθοδος POST τοποθετεί τις πληροφορίες αυτές στον κορμό(body) του HTTP request και όχι στο URL.Ήδη λοιπόν φαίνεται ότι ευαίσθητα στοιχεία όπως κωδικούς αυθεντικοποίησης δεν θα πρέπει ποτέ να μεταφέρονται με μέθοδο GETκαθώς τα δεδομένα θα εκτεθούν πάνω στο URL.

1.6Cookies (μπισκότα)

Ένα άλλο σημαντικό στοιχείο του πρωτοκόλλου HTTPείναι τα λεγόμενα cookies που στηρίζονται οι περισσότερες web εφαρμογές. Η αναφορά στα cookiesγίνεται επειδή συχνά μπορούν να χρησιμοποιηθούν ως βοήθεια για την αξιοποίηση τρωτών σημείων. Στην ουσία τα cookieseίναι text(αρχεία κειμένου) τα οποία αποθηκεύονται στον προσωπικό μας φυλλομετρητή κατά την περιήγησή μας στο διαδίκτυο. Τα αρχεία αυτά περιέχουν πληροφορίες σχετικά με τις ιστοσελίδες τις οποίες επισκεπτόμαστε. Πληροφορίες τέτοιες είναι συχνά το username μας και ο κωδικός

μας(password) με σκοπό να την επόμενη φορά που θα θέλουμε να περιηγηθούμε ξανά στην ιστοσελίδα να έχει τα στοιχεία μας και να μην χρειαστεί να ξανακάνουμε login.

1.7Ανατομία ενός URL

Προηγουμένως μιλήσαμε για το πρωτόκολλο HTTPκαι τις μεθόδους GETκαι POSTκαι πως η μία μέθοδος εμφανίζει το ερώτημα μας πάνω στο URLενώ ή άλλη όχι. Συνεπώς είναι απαραίτητο να κατανοήσουμε καλύτερα το URL.Σε αυτή την ενότητα θα παρουσιάσουμε και θα εξηγήσουμε τα μέρη ενόςURL. Έστω λοιπόν το URL:

http://www.example.nlm.nih.gov/VecScreen/VecScreen.txt

Το παραπάνω URL αποτελείται από:

- Το πρωτόκολλο http
- www.example.nlm.gov: host
- www: πέμπτο επίπεδο domain (fifth level domain)
- example: τέταρτο επίπεδο domain (fourth level domain)
- nlm: τρίτο επίπεδο domain (third level domain)
- nih.gov: Domain (πρώτο και δεύτερο επίπεδο)
- VecScreen: Directory name
- VecScreen.txt: file name

1.8 Penetrationtest – Pentesting

Μια διαδικτυακή εφαρμογή λοιπόν θα πρέπει να ελεγχθεί κατά πόσο είναι ευπαθείς(vulnerable) στις διάφορες επιθέσεις από κακόβουλους χρήστες-επιτιθέμενους, η διαδικασία ή τεστ που υπόκειται η εφαρμογή λέγεται penetration test. Έκτος από τις διαδικτυακές εφαρμογές το penetration test το χρησιμοποιούμε για να δούμε και τα κενά ασφάλειας(security bridge) ενός δικτύου ή ακόμα και ενός

υπολογιστή. Το penetration test λοιπόν είναι μια μέθοδος η οποία στοχεύει στο να αξιολογηθεί ένας υπολογιστής, μια εφαρμογή ή ένα δίκτυο ως προς την ασφάλεια τους, αναπαριστώντας μια επίθεση στον υπολογιστή την εφαρμογή ή στο δίκτυο προς εξέταση από εσωτερικές(internal) ή εξωτερικές(external) απειλές(σαν απειλή ορίζεται οτιδήποτε μπορεί να περιορίσει την ασφάλεια ενός πληροφοριακό συστήματος(στην περίπτωση μας εφαρμογή) Το penetration test είναι ένα σκέλος μιας ολοκληρωμένης εκτίμησης ασφάλειας(security assessment).

Οι παραδοσιακές μέθοδοι που θέλουν τα pentesting να γίνονται μία φορά τον χρόνο (ετήσια) παρέχουν μικρή ασφάλεια, ειδικά στις μέρες μας οι οποίες χαρακτηρίζονται από συνεχείς εξελίξεις σε αυτόν τον τομέα. Υπάρχουν πάρα πολλοί παράμετροι και πολύ λίγος χρόνος για να εξασφαλιστεί η ασφάλεια στα πλαίσια ετησίων pentesting κάτι το οποίο μπορεί κανένας να αναλογιστεί σκεφτόμενος ότι ένας επιτιθέμενος έχει 24 ώρες την ημέρα , 7 μέρες την εβδομάδα και 52 εβδομάδες τον χρόνο ενώ ο χρόνος που έχει ο αμυνόμενος (ο χρόνος ο οποίος διεξάγονται τα pentesting) είναι κατά μέσο όρο 20 ημέρες όπου η κάθε μέρα υπολογίζεται σε εργασιακές ώρες. Άρα είναι εύκολο να αντιληφθούμε ότι η μεριά του επιτιθέμενου έχει πολλαπλάσιο χρόνο να ρίξει την άμυνα από τον χρόνο που «χτίζεται» αυτή.

Κεφάλαιο 2 : Application Security

2.1 Εισαγωγή στις Διαδικτυακές Εφαρμογές (Web Application)

Στα πρώτα στάδια της εξέλιξης του Διαδικτύου , αυτό αποτελούταν μόνο από websites. Αυτά ήταν ουσιαστικά αποθήκες πληροφοριών που περιείχαν ένα σύνολο στατικών εγγράφων. Τα προγράμματα περιήγησης στο Web εφευρέθηκαν ως μέσο για την ανάκτηση και την εμφάνιση των εγγράφων αυτών. Η ροή από πληροφορίες ήταν μονόδρομος, από το server στον browser.Οι περισσότεροι ιστότοποι δεν παρείχαν πιστοποίηση χρηστών. Συνεπώς κάθε χρήστης υποβαλλόταν σε επεξεργασία με τον ίδιο τρόπο και ο ιστότοπος παρουσιαζόταν με τις ίδιες πληροφορίες.

Σήμερα το Διαδίκτυο είναι σχεδόν αγνώριστο από την προηγούμενη μορφή του. Η πλειοψηφία των sites στο διαδίκτυο είναι στην πραγματικότητα εφαρμογές που είναι ιδιαίτερα λειτουργικές και βασίζονται στην αμφίδρομη ροή πληροφοριών μεταξύ του server και του browser.Υποστηρίζουν εγγραφή και σύνδεση, χρηματοπιστωτικές συναλλαγές όπως και αναζήτηση και συγγραφή του περιεχομένου τους από τους χρήστες. Το περιεχόμενο που εμφανίζεται στους χρήστες δημιουργείται δυναμικά και συχνά προσαρμόζεται σε κάθε χρήστη. Μεγάλο μέρος της επεξεργασίας των πληροφοριών που είναι ιδιωτικές είναι εξαιρετικά ευαίσθητες, επομένως η ασφάλειά του είναι ένα μεγάλο ζήτημα**.**

2.2 Θέματα Ασφάλειας στις Διαδικτυακές Εφαρμογές

Αν και η ευαισθητοποίηση του web σε θέματα ασφάλειας των εφαρμογών έχει αυξηθεί τα τελευταία χρόνια, παραμένει λιγότερο αναπτυγμένη από ότι σε άλλες περιοχές όπως δίκτυα και λειτουργικά συστήματα. Παρότι οι περισσότεροι που εργάζονται στον τομέα της ασφάλειας έχουν μια λογική κατανόηση από τα βασικά σημεία της διασφάλισης των δικτύων από την άλλη επικρατεί μια σύγχυση για στην

ασφάλεια των web εφαρμογών. Ένας προγραμματιστής web εφαρμογής έχει σαν έργο την επεξεργασία δεκάδων ή ακόμη εκατοντάδων πακέτων τρίτου κατασκευαστή, όλα σχεδιασμένα για να είναι αφηρημένα(abstract) από τις βασικές τεχνολογίες.

Με τον όρο web application penetration testing αναφερόμαστε στην αξιολόγηση της ασφάλειας μιας διαδικτυακής εφαρμογής .Η διαδικασία περιλαμβάνει μια ενεργή ανάλυση της εφαρμογής για τις όποιες αδυναμίες , τεχνικές ατέλειες ή τρωτά σημεία έχει. Τυχόν θέματα ασφαλείας που βρεθούν, θα παρουσιαστούν στον ιδιοκτήτη του συστήματος μαζί με μία αξιολόγηση του αντίκτυπου τους σε περίπτωση επιτυχημένης επίθεσης και συχνά με μια πρόταση για τον μετριασμό ή μια τεχνική λύση. Η μεθοδολογία του OWASPείναι αυτή που χρησιμοποιείται για να διεξάγουμε μια διαδικασία web application penetration testing στην οποία θα αναφερθούμε στην επόμενη ενότητα.

2.3 Μεθοδολογία OWASP

Όταν γίνεται μια αξιολόγηση για την ασφάλεια ενός δικτύου, ενός υπολογιστή ή όπως στην εργασία μας σε μία εφαρμογή υπάρχουν τρείς τρόποι προσέγγισης. Έχουμε την προσέγγιση white box(λευκό κουτί) όπου η ομάδα ή το άτομο που διεξάγει το penetration testέχει πλήρη γνώση για την αρχιτεκτονική και την τεχνολογία της εφαρμογής και στην ουσία το τεστάρει σαν να ήταν ο προγραμματιστής, επίσης υπάρχει η προσέγγιση grey box όπου στην ουσία έχει λιγότερες πληροφορίες και την εξετάζει σαν χρήστης με πρόσβαση στις εσωτερικές πληροφορίες και τέλος έχουμε την προσέγγιση black box όπου δεν ξέρουμε τίποτα για την εφαρμογή και την εξετάζουνε σαν χρήστες. Η μεθοδολογία OWASP βασίζεται στην προσέγγιση του μαύρου κουτιού(black box).

Η μεθοδολογία OWASP έχει δημιουργηθεί για τον έλεγχο των διαδικτυακών εφαρμογών. Δεν δομείται σε συγκεκριμένες φάσης (συλλογή πληροφοριών, ανάλυση τρωτοτήτων και εκμετάλλευση) αλλά σε δύο κυρίως τμήματα παθητικής και ενεργητικής λειτουργίας που περιέχουν ανεξάρτητους ελέγχους. Οι έλεγχοι εστιάζουν σε συγκεκριμένα τμήματα των διαδικτυακών εφαρμογών που πρέπει να ελεγχθούν. Πιο συγκεκριμένα έλεγχοι όπως η εισαγωγή δεδομένων εξετάζουν κάθε είδους

γνωστής επίθεσης που μπορεί να χρησιμοποιηθεί για να αποκτήσει κάποιος πρόσβαση είτε αφορά τη βάση δεδομένων είτε αφορά τα πρωτόκολλα http, την αυθεντικοποίηση, την εξουσιοδότηση κ.α.

2.4 Παθητική Φάση (Passive)

Όπως προείπαμε η μεθοδολογία OWASPστηρίζεται σε δύο κυρίως τμήματα, την παθητική και την ενεργητική φάση. Στην παθητική φάση ο αναλυτής χρησιμοποιεί την εφαρμογή για να καταλάβει τον τρόπο λειτουργίας της. Συλλέγει πληροφορίες (information gathering) για την εφαρμογή μέσω εργαλείων.

Το information gathering γίνεται με τους εξής τρόπους:

- Αξιολόγηση Μεταδεδομένων του διακομιστή(server) για την αποκόμιση πληροφοριών.(Review Web server Metafiles for Information Leakage).

- Μηχανές αναζήτησης για διαρροή πληροφοριών (Conduct search engine discovery/reconnaissance for information leakage).

- Αναγνώριση τρόπων πρόσβασης στην εφαρμογή (Identify application entry Points).

- Αποτυπώματα διακομιστών (Fingerprint Web Servers).

- Απαρίθμηση εφαρμογών στον διακομιστή (Enumerate Applications on Web server).

- Αξιολόγηση των σχόλιων και των μεταδεδομένων της εφαρμογής για διαρροή πληροφοριών.(Review web page comments and metadata for information leakage).

- Καταγραφή μονοπατιών εκτέλεσης μέσω εφαρμογής (Map execution paths through application).

- Χαρτογράφηση αρχιτεκτονικής δικτύου και εφαρμογής(Map Network and Application Architecture).

2.5 Ενεργητική Φάση (active)

Στην ενεργητική φάση ο αναλυτής κάνει τους εξής ελέγχους:

Το Configuration and deploy management testing αναλύει την τοπολογία, την αρχιτεκτονική και την υποδομή της διαδικτυακής εφαρμογής, το πηγαίο κώδικα. Αναλύεται στις εξής υποφάσεις:

-Έλεγχος Παραμετροποίησης υποδομής/εφαρμογής(Test Network/Infrastructure Configuration).

2.6 Αποτίμηση Κινδύνων κατά OWASP

Η μεθοδολογία OWASPόπως αναφέραμε έχει ως στόχο την αξιολόγηση και την αποτίμηση των κινδύνων στις διαδικτυακές εφαρμογές. Με βάση τον OWASP οι μεγαλύτεροι κίνδυνοι είναι:

A1.Injection(ένεση)

A2.Διαχείριση δεδομένων αυθεντικοποίησης και συνόδου

A3.XSS(Cross-Site-Scripting)

A4.Επισφαλείς Αναφορές σε Αντικείμενα

A5.Επισφαλείς Ρυθμίσεις Ασφάλειας

A6.Αποκάλυψη Ευαίσθητων Δεδομένων

A7.Λανθασμένη Λειτουργία για τον Έλεγχο Πρόσβασης

A8.CSRF(Cross-Site-Request-Forgery)

A9.Χρησιμοποίηση εξαρτημάτων με γνωστές αδυναμίες

Κεφάλαιο 3: Επιθέσεις στις Διαδικτυακές εφαρμογές

3.1 SQL Injection

Μια από τις πιο γνωστές και πολυσυζητημένες ευπάθειες στα λογισμικά των εφαρμογών και από τις πρώτες στην λίστα του OWASPείναι η SQL Injection. Ένας σωστός pentester που αξιολογεί την ασφάλεια μίας εφαρμογής είναι από τις πρώτες ευπάθειες(vulnerabilities) που ψάχνει. Όπως με τις περισσότερες κατανεμημένες εφαρμογές, οι web εφαρμογές αντιμετωπίζουν ένα θεμελιώδες πρόβλημα το οποίο πρέπει να αντιμετωπιστεί για να θεωρούνται ασφαλής. Επειδή ο πελάτης είναι έξω από τον έλεγχο της εφαρμογής, οι χρήστες μπορούν να υποβάλουν είσοδο στην εφαρμογή. Στην αίτηση(request) πρέπει να υποθέσουμε ότι όλες οι είσοδοι είναι δυνητικά κακόβουλες. Ως εκ τούτου θα πρέπει να ληφθούν μέτρα για να εξασφαλιστεί ότι οι επιτιθέμενοι δεν μπορούν να χρησιμοποιήσουν δημιουργημένη είσοδο η οποία θέτει σε κίνδυνο την εφαρμογή παρεμβαίνοντας με την λογική την συμπεριφορά της, ώστε να μην μπορούν να αποκτήσουν μη εξουσιοδοτημένη πρόσβαση σε δεδομένα. Πλέον οι μεγαλύτερες αδυναμίες στο λογισμικό δεν βρίσκονται στο λειτουργικό σύστημα(Linux,Windows,MacOS,BSD) ούτε στους web servers(Apache,IISκ.α) αλλά στις εφαρμογές που προορίζονται για τον τελικό χρήστη(client server).Τέτοιες εφαρμογές αποτελούν τα γνωστά Συστήματα Διαχείρισης Περιεχομένου(CMS, από το Content Management System) αλλά και πολλές άλλες που εκτελούνται στους διάφορους servers. Συχνά οι προγραμματιστές είτε για να προλάβουν τις διορίες(deadlines) είτε από άγνοια των βασικών αρχών ασφάλειας δεν προσέχουν ιδιαίτερα τον κώδικα που γράφουν. Αυτό έχει σαν αποτέλεσμα ο κώδικας ο οποίος γράφουν να μην είναι ασφαλής και να περιέχουν κενά ή αλλιώς όπως λέμε τρύπες(holes).Βασικά πρόκειται για αδυναμίες που θα μπορούσε να εκμεταλλευτεί κάποιος για να αποκτήσει πρόσβαση σε δεδομένα που κανονικά δεν θα έπρεπε να έχει. Πολλές φορές εκμεταλλευόμενος αυτές τις αδυναμίες ο επιτιθέμενος μπορεί να αποκτήσει πρόσβαση στον server με ότι αυτό συνεπάγεται. Συγκεκριμένα σε αυτό το κεφάλαιο θα εστιάσουμε στις αδυναμίες οι οποίες εμφανίζονται πιο συχνά και πως αυτές τις εκμεταλλεύονται οι επιτιθέμενοι.

3.2 SQL Injection και PHP-My SQL

Μία γλώσσα που χρησιμοποιείται ευρέως στο back-endτων εφαρμογών είναι η PHP (server side).Κατά την διάρκεια ενός ελέγχου ασφάλειας είναι πάρα πολύ συχνό φαινόμενο να ερχόμαστε αντιμέτωποι με ένα κομμάτι κώδικα PHP σαν το ακόλουθο:

```
// Σύνδεση με την βάση δεδομένων
    $link= mysql_connect ("Host", "user", "pass") OR die
("Database Error");
    $db_selected = mysql_select_db ('db1' , $link);
// Πάρε Χρήστη και pass από URL
$user=$_GET["user"];
    $password=$_GET["pass"];
//Ελεγχος αν ο χρήστης είναι στην βάση
$query="SELECT * FROM ajv_users WHERE user_nick="$user"
          AND password="$password"
    $nrows=mysql_num_rows($result);
       If ( $nrows==0) {
die ("Λανθασμένα Στοιχεία!");
```

Ο παραπάνω κώδικας είναι ευάλωτος σε SQL Injections διότι δεν ελέγχει τα περιεχόμενα των μεταβλητών $user και $password πριν τα προσθέσει στην μεταβλητή $query. Επίσης όπως παρατηρούμε οι μεταβλητές περιέχονται στοURL καθώς χρησιμοποιείτε η μέθοδος GETγια το request κάτι το οποίο όπως είδαμε παραπάνω πρέπει να αποφεύγεται σε αποστολή κωδικών και ονομάτων. Όσο αναφορά των έλεγχο των τιμών που εισάγει ισχύει ο χρυσός κανόνας ασφάλειας:

“Οτιδήποτε μπορεί να προέρχεται από τον τελικό χρήστη είναι κατά πάσα πιθανότητα κακόβουλο”

3.3 GET και POST Method

Υπάρχουν δύο κύριες μέθοδοι έτσι ώστε ο χρήστης να δώσει τα στοιχεία του (username,password), η GET και η POST.

- GET: η μέθοδος GET παρουσιάζει-εκθέτει(expose) τα στοιχεία του χρήστη στο URL(username,password)

- POST:η μέθοδος POST τοποθετεί (places) τα στοιχεία του χρήστη στον κορμό(body) του request και όχι στο URL.

Παράδειγμα GET request:

```
Get / search.jsp?name=blah&type=1 HTTP/1.0
User-Agent: Mozilla /4.0
Host: www.mywebsite.com
Cookie:
SESSIONID=2KDSU72H9GSA978
<CRLF>
```

Παράδειγμα POST request:

```
Post /search.jsp HTTP/1.0
User-Agent: Mozilla/4.0
Host: www.mywebsite.com
Content-Lenght:16
Cookie:
```

```
SESSIONID:2KDSU72H9GSA289
<CRLF>
name=blah$type=1
<CRLF>
```

3.4 Ανατομία του SQL Injection

Έστω λοιπόν ότι η σελίδα μας βρίσκεται στο www.mywebsite.com/safe.php.Για να λειτουργήσει σωστά πρέπει ο χρήστης να δώσει στο URL το εξής:

www.mywebsite.com/Safe.php?user=admin&password=mypass

Οι μεταβλητές $user και $password θα έχουν αντίστοιχα τις τιμές admin και mypass.Αυτές οι μεταβλητές θα εισαχθούν στην μεταβλητή $query και στην συνέχεια θα εκτελεστεί η αντίστοιχη εντολή sql στην βάση δεδομένων . Αν δεν βρεθεί ο χρήστης θα διακοπεί το πρόγραμμα με το μήνυμα “Wrong Credentials”.Έστω ότι ένας κακόβουλος χρήστης δώσει το παρακάτω:

www.mywebsite.com/Safe.php?user=admin&password='OR' 1=1

Αν κάνει κάτι τέτοιο ο κακόβουλος χρήστης η εντολή sql που θα εκτελεστεί στη βάση δεδομένων είναι η εξής:

```
SELECT * FROM ajv_users WHERE usr_nick=’admin’ AND user_pass= ‟ OR ‘ 1=1’
```

Αυτή η συνθήκη θα επιστρέφει πάντα μία ή περισσότερες εγγραφές από τον πίνακα, καθώς η συνθήκη ‘ 1=1 ‘ είναι πάντα αληθής. Ως αποτέλεσμα η μεταβλητή

$nrows δεν θα είναι ποτέ μηδενική και το πρόγραμμα θα ¨νομίζει¨ ότι εφόσον βρήκε εγγραφές στον πίνακα χρηστών, ο χρήστης είναι νόμιμος

Η Βασική φόρμα ενός sql query είναι της μορφής:

SELECT<στήλες>

FROM<έναν ή περισσότερους πίνακες>

Οι στόχοι της επίθεσης SQL Injection είναι οι εξής:

-Authentication Bypass

-Αποκάλυψη Δεδομένων (Information Disclosure)

- Ακεραιότητα Δεδομένων(Compromised data Integrity)

- Διαθεσιμότητα Δεδομένων(Compromised availability of data)

- Απομακρυσμένη εκτέλεση εντολών(Remote Command Execution)

3.5 Τύποι SQL Injection

Οι επιθέσεις SQL Injection έχουν διάφορους τύπους ανάλογα τον τρόπο που εκτελούνται:

Αρχικά έχουμε τις κανονικές επιθέσεις ή βασισμένες στο λάθος(error based) sql injections όπου η εφαρμογή γυρνάει χρήσιμες πληροφορίες από τα queries που εκτελεί ο επιτιθέμενος

Επίσης έχουμε και τις τυφλές επιθέσεις (blind attacks) : Η εφαρμογή δεν “δίνει” κάποια έξοδο από τα queries που εκτελεί ο επιτιθέμενος, σε αυτές τις περιπτώσεις χρησιμοποιούνται sql injections τα οποία:

- Είτε είναι μια σειρά ερωτήσεων του τύπου αληθής(TRUE) ψευδής(FALSE) μέσω sql δηλώσεων(statements)

- Είτε επιθέσεις χρόνου(timing attacks): εισάγει καθυστέρηση σαν κομμάτι της sql δήλωσης.

3.6 Cross-Site Scripting (XSS)

Μία ακόμα από τις κυριότερες επιθέσεις που εντοπίζεται από τον OWASP είναι οι επιθέσεις τύπου Cross-Site Scripting.

Εκτός από τις επιθέσεις sql injections υπάρχουν και άλλες πολλές που έχουν να κάνουν με την εισαγωγή κακόβουλου κώδικα, οι οποίες δεν απαιτούν την ύπαρξη κάποιας βάσης δεδομένων (sql injection). Η διασημότερη από αυτές είναι οι επιθέσεις Cross-Site Scripting(XSS). Το ακρώνυμο της επίθεσης είναι XSS και όχι CSS για να αποφύγουμε την παρεξήγηση με τα Cascading Style Sheets που εν συντομία ονομάζονται CSS. Ας δούμε λοιπόν ένα κομμάτι κώδικα PHP για να αρχίσουμε να αναλύουμε αυτή την επίθεση:

```
<?php
 $name=$GET_name[“name”];
 echo “Helloname” ;
   ?>
```

Ο παραπάνω κώδικας διαβάζει το όνομα που δηλώνει ο χρήστης μέσα από το URL και τυπώνει έναν σύντομο χαιρετισμό. Ο κώδικας αυτός αν και απλός επιτρέπει επιθέσεις του τύπου XSS.Αυτό συμβαίνει διότι η μεταβλητή $name λαμβάνει την τιμή της από τον χρήστη και χρησιμοποιείται άμεσα, χωρίς να έχει ελεγχθεί ή να έχει γίνει sanitized. Όταν ελέγχουμε κώδικα για τρύπες ασφαλείας (security bridge) έχουμε στο μυαλό μας έναν πολύτιμο κανόνα

Οποιαδήποτε μεταβλητή παίρνει την τιμή της από τον χρήστη είναι ύποπτη και πρέπει να φιλτραριστεί, ασχέτως αν χρησιμοποιήθηκε η μέθοδος POST ή GET ή κάποιο άλλο HEADER.

3.6.1 XSS και SQL Injection

Τόσο το XSS(Cross Site Scripting) όσο και το SQL Injection είναι επιθέσεις εισαγωγής κώδικα.Το XSS υλοποιείται κατά βάση με εισαγωγή κώδικα JavaScript. Στην επίθεση XSS ο browser εκτελεί τα δεδομένα που έχει εισάγει ο χρήστης σαν κώδικα. Στην επίθεση SQL Injection η βάση(database) ή ο πηγαίος κώδικας καλεί την βάση και "μπερδεύει" το περιεχόμενο δεδομένων(data context) και ANSISQL (περιεχόμενο για εκτέλεση[execution context])

Ας υποθέσουμε ότι ο παραπάνω κώδικας PHP που είδαμε προηγουμένως εκτελείται σε έναν web server, τοπικά. Για να επιτεθούμε (εκπαιδευτικά) στον web server θα δώσουμε το ακόλουθο URL:

-Localhost/tests/SafeXSS.php?name=<script>alert(“Hello XSS”) </script>

Σε αυτό το παράδειγμα χρησιμοποιούμε την μεταβλητή name για να φυτέψουμε παράνομο κώδικα σε Javascript.Ο κώδικας αυτός θα μεταφερθεί κανονικά στον server και από εκεί θα επιστρέψει στον browser μας να εκτελεστεί. Κατά την διαδικασία του rendering ο browser εμφάνισε τα περιεχόμενα της μεταβλητής name και γενικότερα όσα του έστειλε ο server. Τα περιεχόμενα όμως της μεταβλητής είναι κώδικας Javascript(<script>alert(“HelloXSS”)</script>).Ο browser δεν χειρίστηκε το περιεχόμενο της μεταβλητής σαν μια σειρά απλών χαρακτήρων αλλά ως κώδικας Javascript, τον οποίο και εκτέλεσε.

3.7 CROSS-SITE-REQUEST-FORGERY (CSRF)

Μία ακόμα επίθεση που βρίσκεται ψηλά στην λίστα του OWASP είναι οι επιθέσεις τύπου CSRF.

Το CSRF είναι μια επίθεση κατά την οποία ο φυλλομετρητής του θύματος εξαπατείται να εκτελέσει μια εντολή σε μια ευπαθή εφαρμογή .Προκαλείται από το γεγονός ότι οι φυλλομετρητές εμπεριέχουν αυτόματα δεδομένα αυθεντικοποίησης (session id,διεύθυνση IP,διαπιστευτήρια Windows Domain κ.α) σε κάθε αίτηση .Οι συνήθεις επιπτώσεις μιας τέτοιας επιθέσεις είναι:

- Διενέργεια συναλλαγών (μεταφορά χρημάτων, αποσύνδεση χρήστη, κλείσιμο λογαριασμού).
- Πρόσβαση σε ευαίσθητα δεδομένα.
- Αλλαγή στοιχείων χρήστη.

3.7.1Εισαγωγή στο CSRF

Έστω ότι είμαστε υπεύθυνοι (administrators) σε ένα forum που έχει να κάνει με ασφάλεια υπολογιστών. Κάθε φορά που επισκεπτόμαστε το forum με τον αγαπημένο μας web Browser δίνουμε κωδικό και password για να αποκτήσουμε πρόσβαση. Η διαδικασία αυτή ονομάζεται αυθεντικοποίηση (authentication). Όση ώρα είμαστε συνδεδεμένοι στο site μπορούμε να εκτελέσουμε διάφορες ενέργειες όπως να γράψουμε ένα νέο post, ν' απαντήσουμε σε κάποιο άλλο κ.λπ. Δηλαδή ότι μπορούν και τα υπόλοιπα μέλη του forum. Όμως εμείς (ως administrator) μπορούμε να εκτελέσουμε κάποιες εργασίες που τα άλλα μέλη δεν μπορούν, όπως για παράδειγμα την διαγραφή κάποιου post ενός άλλου μέλους, ή την αποβολή του από το forum (ban) και πολλά άλλα. Πώς γνωρίζει ο server ποιοι είμαστε για να μας αφήσει να εκτελέσουμε αυτές τις ενέργειες;

Το ότι κάναμε authentication μια δεδομένη χρονική στιγμή δεν σημαίνει απολύτως τίποτε διότι το Internet είναι stateless και κάθε φορά που πατάμε μια ενέργεια στον web browser μας, ο server που επισκεπτόμαστε είναι σαν να μας “βλέπει” για πρώτη φορά. Αυτή η έλλειψη “μνήμης” του server πρέπει με κάποιο τρόπο να αντιμετωπιστεί, έτσι ώστε να μας θυμάται για να μην απαιτεί authentication κάθε φορά που του ζητάμε κάτι. Για την αντιμετώπιση αυτού του προβλήματος έχουμε τα μπισκοτάκια(cookies).

Cookies ονομάζονται κάποιες μικρού μεγέθους πληροφορίες που βάζει ο server στη μνήμη του υπολογιστή μας μέσω του browser, έτσι ώστε κάθε φορά που τον καλούμε να θυμάται ποιοι είμαστε. Οπότε αυτό που χρειάζεται είναι να "κλαπεί" το cookie του administrator το οποίο θα χρησιμοποιηθεί για αυθεντικοποίηση ως administrator.

3.7.2 CSRF και XSS

Το CSRF και το XSS ενώ είναι επιθέσεις που μοιάζουν ίδιες έχουν μια σημαντική διαφορά. Ενώ λοιπόν στο XSS ο επιτιθέμενος εκμεταλλεύεται την εμπιστοσύνη που έχει ο web server στον χρήστη(ανεπιθύμητη είσοδο) στο CSRF ο επιτιθέμενος εκμεταλλεύεται την εμπιστοσύνη που έχει ο web server στον εκάστοτε web browser.Συνεπώς ο επιτιθέμενος βασίζεται και εκμεταλλεύεται διαφορετικές αδυναμίες του server για να εξαπολύσει την επίθεση του.

3.7.3 Παράδειγμα CSRF

Έστω κάποιος χρήστης έχει κάνει εισαγωγή σε μια διαδικτυακή εφαρμογή μιας τράπεζας για να κάνουμε τις πληρωμές μας κι αυτή μας έχει δώσει ένα session cookie. Αν καταφέρει κάποιος επιτιθέμενος να κλέψει το id του cookie μας ενώ

είμαστε συνδεδεμένοι. και το cookie δεν περιλαμβάνει κι άλλα αναγνωριστικά στοιχεία θα είναι πολύ εύκολο στον υποκλοπέα να δημιουργήσει ένα νέο cookie, χρησιμοποιώντας το ίδιο id με αυτό του χρήστη-θύματος. Σ' αυτή την περίπτωση ο υποκλοπέας θα έχει τα ίδια δικαιώματα με τον χρήστη στην τράπεζα κι αυτό, με άλλα λόγια, σημαίνει ότι θα έχει πρόσβαση στο λογαριασμό του.

Αν όμως τα cookies της τράπεζας εκτός από το id διατηρούν και πληροφορίες όπως ο web browser και η διεύθυνση IP από την οποία συνδεθήκαμε, ο υποκλοπέας θα είναι αδύνατο να εξαπατήσει τον server της τράπεζας. Όποτε ακόμα κι αν το πλαστό cookie ήταν πανομοιότυπο με το cookie του χρήστη η διεύθυνσή του υποκλοπέα θα είναι σίγουρα διαφορετική από του θύματος κι αυτό θα "χτυπήσει" στον server της τράπεζας, ο οποίος με τη σειρά του θα αρνηθεί να εξυπηρετήσει τα σχετικά requests.

Οι προγραμματιστές, λοιπόν, θα πρέπει να προσέχουν πολύ τον τρόπο με τον οποίο κατασκευάζονται τα session cookies από τα προγράμματά τους. Αυτά δεν θα πρέπει σε καμιά περίπτωση να λειτουργούν σε οποιοδήποτε μηχάνημα κι οποτεδήποτε. Θα πρέπει με κάποιον τρόπο να δένονται με τον συγκεκριμένο client και για μια προκαθορισμένη, σύντομη χρονική περίοδο. Έτσι, ακόμα κι αν καταφέρει κάποιος να τα υποκλέψει δεν θα λειτουργούν στο δικό του μηχάνημα. Για παράδειγμα, θα ήταν φρόνιμο τα cookies να περιέχουν πληροφορίες από τον client, οι οποίες σε κάθε request θα επαληθεύονται για την ορθότητα τους. Πιο συγκεκριμένα, θα μπορούσαν να διατηρούν το όνομα και την έκδοση του web browser, την IP του χρήστη, τον http referrer, έναν μοναδικό, τυχαίο και πολύ μεγάλο αριθμό που θα προέρχεται από το server και πάει λέγοντας. Όλες αυτές οι πληροφορίες θα πρέπει επίσης να είναι κρυπτογραφημένες, ώστε να μην είναι εύκολη η τροποποίησή τους (tampering).

Έστω ο παρακάτω κώδικας :

```
  <form action="http://site.com/Transfer.asp"
method="POST" id="form1">
  <p>Account Num: <input type="text" name="acct
  value="2345"/></p>
```

```
  <p>Transfer Amt: <input type="text" name="amount" value="10000"/></p>
  </form>
  <script>document.getElementById("form1").submit();</script>
```

Μια τέτοια φόρμα θα δώσει το ακόλουθο request:

-GET http://www.example.com/Transfer.asp?acct=##amount=##

Άμα λοιπόν ο επιτιθέμενος θέλει να ξεγελάσει κάποιο θύμα θα έδινε κάτι τέτοιο:

-<imgsrc= "http://example.com/Transfer.asp?acct=attacker&amount=100000" />

Ας δούμε τα αποτελέσματα από κάτι σαν τον παραπάνω κώδικα του επιτιθέμενου:

-1.Όταν θα "φορτωθεί" το <img>tag ο επιτιθέμενος θα στείλει ένα request στην εφαρμογή της τράπεζα με την οποία κάνει συναλλαγές ο χρήστης-θύμα.

-2.Ο browser του χρήστη από την άλλη θα στείλει το κατάλληλο cookie το οποίο τον αυθεντικοποιεί στο request του επιτιθέμενο

-3.Η εφαρμογή της τράπεζας θα επαληθεύσει ότι το cookie είναι έγκυρο και θα επεξεργαστεί το request του επιτιθέμενου.

Ο επιτιθέμενος δεν μπορεί να δει το αποτέλεσμα του request.

3.8 Έλεγχος Πρόσβασης (Access Control)

Μια ακόμα επίθεση που στοχεύει στην υποτίμηση της ασφάλειας μια διαδικτυακής εφαρμογής είναι επιθέσεις στον έλεγχο πρόσβασης μιας εφαρμογής.

3.8.1 Τι είναι ο Έλεγχος Πρόσβασης

Στην επιστήμη τον ηλεκτρονικών υπολογιστών ο Έλεγχος Πρόσβασης ή Access Control List (ACL) είναι μία λίστα με δικαιώματα που χαρακτηρίζουν κάποιο συγκεκριμένο αντικείμενο. Η λίστα αυτή καθορίζει με ακρίβεια ποιος μπορεί να εκτελέσει μία εργασία πάνω σε ένα αντικείμενο και τι είδους εργασία θα είναι αυτή (στην περίπτωσή μας το αντικείμενο είναι η διαδικτυακή εφαρμογή και οι λειτουργίες αυτής).

Για την καλύτερη κατανόηση αυτού του κεφαλαίου αρχικά θα δώσουμε κάποιους ορισμούς που θα μας βοηθήσουν στην ανάγνωσή του.

Εξουσιοδότηση: Εξουσιοδότηση ονομάζεται η διεργασία ενός συστήματος που αποφασίζει αν ένας συγκεκριμένος χρήστης έχει πρόσβαση στους πόρους μιας εφαρμογής.

Άδεια(:Permission):Καθορίζει την συμπεριφορά της εφαρμογής προς το χρήστη.

Δικαίωμα (Entitlement):Τι επιτρέπεται ο χρήστης να "κάνει" σε μία εφαρμογή.

Αξίωμα/χρήστη(Principle/User):Ποίος και ποιά είναι τα δικαιώματά του

Συγκεκριμένος ρόλος: ονομασία άδειας, συσχέτιση με άλλους χρήστες

-if(user.isRole("Manager"));

Γενικός Ρόλος: ονομασία άδειας, συσχέτιση με τους πόρους της εφαρμογής

-if(user.isAuthorized("report:view:1234"));

3.8.2 Τύποι Επιθέσεων στον Έλεγχο Πρόσβασης

Έχουμε δύο τύπους επιθέσεων στον έλεγχο πρόσβασης

1.Ένας κακόβουλος χρήστης καταφέρνει να "μπει" στην εφαρμογή μεν δικαιώματα διαχειριστή (admin)

2.Παρόμοια με την προηγούμενη απλά καταφέρνεις να έχεις πρόσβαση σαν κάποιος άλλος χρήστης κερδίζοντας πρόσβαση στα ευαίσθητα δεδομένα του χρήστη/θύματος.

3.8.3 Συνέπειες την επίθεσης στον Έλεγχο Πρόσβασης

1.Απώλεια του accountability.

-Η δυνατότητα του επιτιθέμενου να εκτελεί ενέργειες σαν ένας άλλος χρήστης

2.Αποκάλυψη ευαίσθητων δεδομένων .

-Όταν ο επιτιθέμενος "μπαίνει" από λογαριασμό διαχειριστή (admin) , συνήθως έχει πρόσβαση και στα ευαίσθητα δεδομένα των χρηστών

3.Αλλοίωση των δεδομένων .

-Τα επίπεδα των δικαιωμάτων δεν διαχωρίζουν ποιος χρήστης έχει δικαίωμα μόνο προβολής και ποιος της μετατροπής (modify) των δεδομένων

3.8.4 Κακές Πολιτικές Έλεγχου Πρόσβασης

Παρακάτω θα ονομάσουμε κάποιες πολιτικές ελέγχου πρόσβασης οι οποίες είναι "κακές" πρακτικές(anti-patterns):

1.Οι ρόλοι να είναι "καρφωμένοι"(hard-coded) στον κώδικα της εφαρμογής

2.Έλλειψη λογικής κεντρικού ελέγχου πρόσβασης

3.Το να χρησιμοποιούνται μη-αξιόπιστα δεδομένα για την απόφαση ελέγχου πρόσβασης

4.Ο έλεγχος πρόσβασης όπου είναι ελεύθερος(open by default)

5.Έλεγχος πρόσβασης που απαιτεί πολιτική ανά χρήστη

3.8.4.1 Hard-Coded Roles

Ας δούμε ένα παράδειγμα κατά το οποίο οι ρόλοι είναι ¨καρφωμένοι¨ στον κώδικα:

```
void editProfile ( User u , EditUsereu) {
if ( u.isManage()) {
editUser(eu)
  }
}
if ( ( user.isManager () ||
user.isAdministrator () ||
user.isEditor () ) &&
          user.id () != 1132))
{
```

```
    // εκτέλεση ενεργειών
}
```

3.8.4.2 Εξάρτηση από Μη- Έμπιστα Δεδομένα

Οι αποφάσεις ελέγχου πρόσβαση (access contol) δεν πρέπει να εξαρτώνται απόrequest δεδομένα Επίσης οι αποφάσεις ελέγχου πρόσβασης δεν πρέπει να γίνονται σε Javascript. Τέλος οι αποφάσεις εξουσιοδότησης δεν πρέπει να βασίζονται σε :

-κρυφά πεδία (hidden fields)

-cookies values

-παραμέτρους φόρμας (form parameters)

-URL παραμέτρους (URL parameters)

-οτιδήποτε από δεδομένα request

3.9 Άλλες Γνωστές Επιθέσεις

Όπως αναφέρθηκε στην αρχή της εργασίας αυτής έχουμε και άλλες επιθέσεις κατά OWASPπου στοχεύουν στην παραβίαση της ασφάλειας των διαδικτυακών εφαρμογών (web application).

3.9.1 Broken Authentication and Session Management

Οι προγραμματιστές συχνά γράφουν δικά τους συστήματα για την πιστοποίηση και τη διαχείριση συνόδου των χρηστών της εφαρμογής τους, αλλά το να κατασκευάζεις σωστά τέτοια συστήματα είναι δύσκολο. Ως αποτέλεσμα, πολλά από αυτά τα custom συστήματα έχουν προβλήματα ασφαλείας σε διάφορες λειτουργίες τους, όπως logout, session timeouts, “remember me”, κλπ. Το να βρεις αυτά τα προβλήματα αν μιλάμε για κάποια custom υλοποίηση είναι δύσκολο αλλά όχι ακατόρθωτο, ενώ ο αντίκτυπος μπορεί να είναι μεγάλος.

3.9.2 Παράδειγμα 1:

Μια custom εφαρμογή για κράτηση εισιτηρίων τοποθετεί για κάποιο λόγο το session ID στο URL της:

http://example.com/sale/tickets;*jsessionid=2P0OC2JDPXM0OQSNDLPSKHCJUN2JV*?dest=Peuki

Ο χρήστης θέλει να ενημερώσει τους φίλους του για το ταξίδι και στέλνει με email το παραπάνω link σε γνωστούς του.

Όταν αυτοί χρησιμοποιήσουν το link αυτό, θα χρησιμοποιήσουν επίσης το δικό του session (με τα δικά του στοιχεία, πχ πιστωτική κάρτα)

3.9.3 Παράδειγμα 2:

Ένας χρήστης χρησιμοποιεί ένα κοινόχρηστο υπολογιστή (internet café) για να συνδεθεί.

Αντί να επιλέξει logout ο χρήστης απλά κλείνει τον browser και φεύγει. Κάποιος άλλος χρήστης (κακόβουλος) χρησιμοποιεί τον ίδιο (ακόμα συνδεδμένο) browser μισή ώρα αργότερα.

3.10 Insecure Cryptographic Storage

Το σύνηθες σφάλμα σε αυτήν την περιοχή είναι απλά η μη κρυπτογράφηση δεδομένων που θα έπρεπε να κρυπτογραφηθούν. Όταν χρησιμοποιείται κρυπτογράφηση, μπορούν να προκύψουν προβλήματα από χρήση αδύναμων αλγορίθμων hashing χωρίς salting (μια ποσότητα πληροφορίας που προστίθεται στην πληροφορία που έχει γίνει hashing) κλπ. Οι επιτιθέμενοι συνήθως δεν σπάνε την ίδια την κρυπτογράφηση, αλλά συνήθως ανακαλύπτουν κάποιο άλλο σφάλμα όπως το να βρουν κάποια έτοιμα κλειδιά, clear text κωδικούς κλπ. Ας δούμε παρακάτω μερικά παραδείγματα

3.10.1 Παράδειγμα 1:

Μια σελίδα καταχωρεί τους κωδικούς των χρηστών της στη Βάση Δεδομένων χωρίς hashing, αλλά σε μορφή clear text. Κάποιος κακόβουλος χρήστης ανακαλύπτει ένα SQL injection flaw στη σελίδα και γίνεται κάτοχος όλης της λίστας χρηστών με τους κωδικούς τους.

3.10.2 Παράδειγμα 2:

Η Βάση Δεδομένων για την αποθήκευση κωδικών των χρηστών μιας διαδικτυακής εφαρμογής χρησιμοποιεί unsalted hashes για την αποθήκευση των κωδικών. Κάποιος κακόβουλος ανακαλύπτει ένα σφάλμα στην εφαρμογή και κατορθώνει να γίνει

κάτοχος του αρχείου κωδικών. Μπορεί να ανακτήσει όλους τους κωδικούς σε4 εβδομάδες, ενώ τα αντίστοιχα salted hashes θα απαιτούσαν3000 χρόνια.

Κεφάλαιο 4: Αντίμετρα

Στο προηγούμενο κεφάλαιο είδαμε τις πιο συχνές και επικίνδυνες επιθέσεις που στοχεύουν τις διαδικτυακές εφαρμογές. Σε αυτό λοιπόν το κεφάλαιο θα δείξουμε κάποιες καλές πρακτικές οι οποίες έχουν ως στόχο την προστασία των διαδικτυακών εφαρμογών από τις επιθέσεις αυτές

4.1Αντίμετρα για SQL Injection

Όπως είδαμε παραπάνω οι επιθέσεις SQL Injection είναι και αρκετά συχνής αλλά και πολύ επικίνδυνες. Σε αυτήν την ενότητα θα παρουσιάσουμε κάποιες τεχνικές που έχουν ως στόχο την αντιμετώπιση τέτοιου είδους επιθέσεων.

Parameterized Query :

Ένα αντίμετρο είναι η παραμετροποίηση (parameterized) των queries η οποία διασφαλίζει ότι ο επιτιθέμενος δεν μπορεί να αλλάξει τον σκοπό ενός query ακόμα και αν οι sql εντολές έχουνε εισαχθεί από αυτόν. Παρακάτω θα δούμε μερικές πρακτικές ασφαλούς κώδικα σε βασικές γλώσσες προγραμματισμού με παραδείγματα:

JavaEE- χρησιμοποιούμε την συνάρτηση PreparedStatement():

```
  String query= "SELECT account_balance FROM user_data WHERE user_name=?";
  PreparedStatementpstmt=connection.prepareStatement(query);
  pstmt.setString(1,custname);
  ResultSet results=pstmt.executeQuery();
```

C# .NET Prepared Statement παράδειγμα:

```
  String query = "SELECT account_balance FROM user_data
WHERE user_name=?";
        try {
  OleDbCommand command=new
OleDbCommand(query,connection);
  command.Parameters.Add(new
OleDbParameter("customerName,CustomerNameName.text));
  OleDbDataReader reader=command.ExecuteReader();//..
  }
  catch (OleDbException se) {
  //error handling
  }
```

.NET Dynamic sql (όχι τόσο καλή ασφάλεια):

```
  string sql="SELECT * FROM User WHERE Name='" +
NameTextBox.Text +'" AND Password ='" +
PasswordTextBox.Text +'"";
```

Παραμετροποιημένο query(parameterized)(παρέχει καλή ασφάλεια):

```
  SqlConnectionobjConnection= new
SqlConnection(_Connection);
  objConnection.Open();
  SqlCommandobjCommand= new SqlCommand( "SELECT * FROM
Users WHERE Name=@Name AND Password
=@Password",objConnection);
```

```
  obgCommand.Parameters.Add("@Name", NameTextBox.Text);
  objCommand.Parameters.Add("@Password",
PasswordTextBox.Text);
  SqlDataReaderobjReader=objCommand.ExecuteReader();
  if (obgReader.Read()) {...
```

Java Prepared Statement

```
  Dynamic sql (ευπαθής σε sql
injection)(injectable):


  -String sqlQuery="UPDATE EMPLOYEES SET
SALARY='+
  request.getParameter("newSalary") +
'WHERE ID='+
```

PreparedStatement (not injectable):

```
  -double newSalary=request.getParameter("newSalary");
  int id=request.getParameter("id");
  PreparedStatementpstmt=con.preparedStatement("UPDATE
EMPLOYEES SET SALARY = ? WHERE ID= ?");
  pstmt.setDouble(1,newSalary);
pstmt.setInt(2,id);
```

Μη ασφαλής HQL statement query(Hybernate):

```
  -unsafeHQLQuery=session.createQuery("from
Inventory where
productID='"+userSuppliedParameter+'"");
```

Ασφαλής έκδοση του προηγούμενου query χρησιμοποιώντας named parameters

```
  - Query safeHQLQuery=session.createQuery("from
Invetory where productID=:productid");
safeHQLQuery.set.Parameter("productid",userSupplied
Parameter);
```

Τέλος ας δούμε και στην PHP

```
  $stm=$dbh→prepare("update users set email =:new_email
where id=:user_id");
  $stmt→bindParam(' :new_email',$email)
     $stmt→bindParam(' :user_id', $id)
```

Τα διάφορα αποσπάσματα κώδικα από που είδαμε παραπάνω έχουν σαν στόχο την παραμετροποίηση (parameterized) των queries για να αποφευχθεί όποια ενέργεια έχει ως στόχο την αλλαγή της συμπεριφοράς αυτών

4.2 Αντιμετώπιση των επιθέσεων XSS

Για την αντιμετώπιση αυτής της επίθεσης πρέπει να εξασφαλίσουμε ότι ο browser δεν θα χειριστεί το περιεχόμενο της μεταβλητής $name σαν κώδικα, αλλά σαν μία απλή συμβολοσειρά.

Πρώτη σκέψη είναι τα qoutes εντός των οποίων περικλείεται το μήνυμα “HelloXSS”

Με βάση αυτά που είχαμε δει στο προηγούμενο κεφάλαιο:

```
<?php
    function Sanitized($Input)
    {
return (get_magic_quotes_gpc()) ?
mysql_real_escape_string(stripslashes($Input)) :
mysql_real_escape_string($Input);
  }
  $link  =  mysql_connect('localhost','myname','mypass')
OR die (mysql_error());
  $name= Sanitized($_GET[“name”]);
  echo “Hello $name”;
?>
```

Με αυτές τις προσθήκες ο κώδικας ψάχνει να βρει μονά quote και όπου τα συναντήσει βάζει μπροστά του ένα backslash (\) και τα απενεργοποιεί (κάνει το λεγόμενο escaping). Με αυτή την τροποποίηση το παράδειγμα επίθεσης που είδαμε

παραπάνω δεν θα πετύχαινε όμως δεν κατάφερε να ξεφύγει από XSS επιθέσεις αλλά από την συγκεκριμένη

Πάμε να δούμε ένα παρόμοιο παράδειγμα χωρίς μονά quotes αλλά με παρόμοιο αποτέλεσμα:

<script>alert(String.fromCharCode(88, 83, 83 , 33));</script>

Το “88,83,83,33” αποτελεί μια κωδικοποιημένη σειρά χαρακτήρων και η συνάρτηση String.fromCharCode() χρησιμοποιείται για την αποκωδικοποίησή της. Οι αριθμοί 88,83,83,33 αποτελούν τους κώδικες ASCII των χαρακτήρων του “XSS!”.Είναι ξεκάθαρο ότι η απενεργοποίηση των quotes δεν μας προστατεύει. Αυτό τελικά που κάνει τον web server να θεωρεί ότι ο server στέλνει κώδικα είναι τα εξής δύο tags: το <script> και </script> .Οπότε πρέπει να απομονώσουμε τους χαρακτήρες “<”,”>”.Με αυτόν τον τρόπο θα απενεργοποιήσουμε τον κώδικα Javascript αλλά και εκείνον σε HTML, που ενδέχεται κάποιος να φυτέψει σε μία μεταβλητή.

Ο απλούστερος τρόπος είναι να μετατρέψουμε το “<” στο αντίστοιχο HTML entity "<" καθώς και το ">" στο αντίστοιχο ">".Αυτή η προσέγγιση έχει το καλό ότι κατά το rendering οι χαρακτήρες θα εμφανίζονται κανονικά(δεν θα αλλοιωθούν ούτε θα χαθούν),αλλά σαν απλό κείμενο και όχι ως tags. Ας δούμε την συνάρτηση που κάνει τα παραπάνω μαζί με τις αλλαγές που πρέπει να κάνουμε στον αρχικό μας κώδικα:

```
<?php
   function SanitizedXSS($Input)
   {
      return
str_replace('<',''$lt;'',srtr_replace('>',''&gt;'',
$Input));
   }
```

```
 $name = SanitizedXSS($_GET[''name'']);
 echo ''<p> Hello $name'';
?>
```

Μετά τις προηγούμενες προσθήκες/τροποποιήσεις ο κώδικας είναι προστατευμένος από κάθε επίθεση σαν αυτές που είδαμε στο παράδειγμα. Ο κώδικας που "φυτέψαμε" στην μεταβλητή $name τυπώνεται ως απλό κείμενο και δεν εκτελείται.

Με αυτά που είδαμε καταφέραμε να ξύσουμε την επιφάνεια του XSS. Αυτή η τεχνική είναι μπορεί να είναι αποτελεσματική όταν ο παράνομος κώδικας έρχεται μέσω μιας μεταβλητής, αλλά ενδέχεται να έρχεται μέσω μιας εικόνας ή εντός του κώδικα HTML. Σε τέτοιες περιπτώσεις χρησιμοποιούμε ειδικά φίλτρα αλλά και βιβλιοθήκες anti-XSS.

Γνωστή anti-XSS βιβλιοθήκη είναι η SafeHTMLAnti-XSSfilter (http://bit.ly/SafeHTMLantiXSS)

-Πρόκειται για έναν parser που χρησιμοποιείτε για το φιλτράρισμα ολόκληρων σελίδων HTML, όταν αυτές προέρχονται από κάποιον client.

4.2.1 Αντιμετώπιση XSS επιθέσεων κατά OWASP

Το Reform Project του OWASP(http://bit.ly/OWASPencodingProject) αποτελεί μια πάρα πολύ καλή και δωρεάν λύση. Πρόκειται για μια έτοιμη κλάση που παρέχεται σε διάφορες εκδοχές, για τις περισσότερες γλώσσες(PHP,ASP.NET,HTML,Javascriptκ.α).Για την PHP το μόνο που χρειάζεται είναι να κατεβάσουμε ένα αρχείο. Ας δούμε το πρόγραμμα μας τροποποιημένο ώστε να χρησιμοποιεί την Reform.

```
<?php
```

```
mb_internal_encoding(''UTF-8'');
  include ('Reform.inc.php');
  $reform=new Reform;
  $name=$reform→HtmlEncode($_GET[''name'']);
  echo ''<p> Hello $name'';
?>
```

Η συγκεκριμένη κλάση προβλέπει τη μετατροπή όλων των χαρακτήρων Unicode σε UTF-8.Κατ' αυτόν τον τρόπο είναι πλέον εφικτός ο έλεγχος καθενός χαρακτήρα χωριστά και ο εντοπισμός των επικίνδυνων. Χωρίς αυτή την μετατροπή θα ήταν σχεδόν αδύνατον να προβλεφτούν όλοι οι δυνατοί χαρακτήρες, για όλα τα δυνατά encodings όλων των γλωσσών.

Η βιβλιοθήκη php-AntiXSS (http://code.google.com/p/php-antixss) αποτελεί επίσης μια καλή λύση που είναι επίσης δωρεάν. Η λειτουργία της βασίζεται στην συνάρτηση preg_match και σε δύο λίστες με επιτρεπτούς (white-listed) και απαγορευμένους (black-listed) χαρακτήρες ή συμβολοσειρές. Έτσι καταφέρνει να απομονώσει και να αντιμετωπίσει κατάλληλα οποιονδήποτε επικίνδυνο χαρακτήρα ή string.

4.2.3 Κωδικοποίηση Εξόδου (EncodingOutput)

Παρακάτω παραθέτεται ένας πίνακας με ασφαλής τρόπους αναπαράστασης "επικίνδυνων" χαρακτήρων σε μία web σελίδα:

Χαρακτήρες (Characters)	Δεκαδικό (Demical)	Δεκαεξαδικό (Hexdemical)	HTML σετ χαρακτήρων	Unicode
" (Διπλά qoutes)	&#34;	&#x22;	&qout;	\u0022
' (Μονά qoutes)	&#39;	&#x27;	&apos;	\uoo27
&	&#38;	&#x26;	&amp;	\u0026
<	&#60;	&#x3C;	&lt;	\u003c
>	&#62;	&#x3F;	&gt;	\u003e

4.3 Προστασία – Άμυνα από επιθέσεις τύπου CSRF

Ας αριθμήσουμε κάποιες διαδικασίες οι οποίες πρέπει να είναι μέρος της αρχιτεκτονικής της εφαρμογής κατά την ανάπτυξη της για να δυσκολέψει αρκετά τον επιτιθέμενο να υλοποιήσει μια τέτοιου είδους επίθεσης:

1.Καταρχήν θα πρέπει να request να γίνονται με POST μέθοδο (εφόσον αφορούν ευαίσθητα δεδομένα)

2.Να απαιτείται να επανα-αυθεντιοποιείται ο χρήστης(re-authentication)

3.Χρήση κρυπτογραφικών Token

4.Το Challenge-Response είναι μια καλή άμυνα για ο CSRF

5.Synchronizer Token Pattern

4.3.1 Challenge-Response

Στην ουσία το challenge-response είναι ένας διαδεδομένος τρόπος αυθεντικοποίησης όπου ο χρήστης υπόκειται σε μια σειρά από ερωτήσεις ή ενέργειες(challenges) τις οποίες καλείτε να απαντήσει(response) έτσι ώστε να αυθεντικοποίηθει.

Παρακάτω είναι μερικές καλές επιλογές για challenge-response.

- CAPTCHA(Completely Automated Public Turing test to tell Computers and Human Apart) : Είναι ένας τύπος challenge-response όπου έχει ο στόχο να καταλάβει να ο χρήστης είναι άνθρωπος ή μηχανή. Το έχουμε συναντήσει όταν κάνουμε εγγραφή σε ένα site που μας ζητάει να αναπαραστήσουμε το περιεχόμενο ενός ταμπλό το οποίο περιέχει γράμματα και αριθμούς.(κάποιες φορές και εμείς οι ίδιοι δυσκολευόμαστε να διακρίνουμε)

- Re-Authentication (password) :Όταν ζητάμε από τον χρήστη ενώ περιηγείται στο site να ξανά-αυθεντικοποιηθεί όταν εκτελεί κάποιες ενέργειες.

- One-time Token : Όπου ο κωδικός ισχύει για μία συνεδρία(session) ή χρηματική μεταφορά(transaction) οπότε σε περίπτωση που κλαπεί δεν μπορεί να χρησιμοποιηθεί ξανά.

Παρόλο που το challenge-response είναι μια πολύ δυνατή άμυνα για CSRF(υποθέτουμε κατάλληλη υλοποίηση), δεν υπάρχει αρκετή εμπειρία του χρήστη. Για εφαρμογές που απαιτούν υψηλό επίπεδο ασφαλείας, τα tokens και τα challenge-response πρέπει να χρησιμοποιούν συναρτήσεις υψηλού κινδύνου.

4.3.2 Synchronizer Token Pattern

Τη τιμή του Token την καθορίζει ο web server και την δίνει την στιγμή "μπαίνουμε" στην σελίδα. Η τιμή αυτή αποθηκεύεται στο session(για java εφαρμογές συνιστάται η java.security.Secure Randomclass)

Μετά την αίτηση token στέλνεται με την φόρμα(form), η τιμή του token πρέπει να ταιριάζει με την τιμή στο session .Ο επιτιθέμενος δεν έχει στην διάθεσή του την

τιμή του token Με επιθέσεις XSS μπορεί να κλαπεί αυτό το token αν η σελίδα είναι ευπαθείς σε XSS επιθέσεις .Θα έχουμε ένα τέτοιο αποτέλεσμα με synchronizer token:

```
  <formaction="/transfer.do"method="post"><input
type="hidden"
name="CSRFToken"value="OWY4NmQwODE4ODRjN2Q2NTlhM
mZlYWEwYzU1YWQwMTVhM2JmNGYxYjJiMGI4MjJjZDE1ZDZjM
TViMGYwMGEwOA==">  ...  </form>
```

4.4 Καλές Πρακτικές Ελέγχου Πρόσβασης

Μια ακόμα συχνή και επικίνδυνη επίθεση που χρήζει αντιμετώπιση είναι επιθέσεις που στοχεύουν στο να προσπελάσεις τον έλεγχο πρόσβασης μιας εφαρμογής. Εδώ λοιπόν θα δούμε πώς πρέπει να δομήσουμε τον έλεγχο πρόσβασης έτσι ώστε να αποφευχθούν τέτοιου είδους επιθέσεις.

Σε αυτή τν περίπτωση μία πολύ καλή πρακτική πάνω σε αυτό είναι η δήλωση ενός κεντρικοποιημένου access controller:

-ACLServise.isAuthorized(PERMISSION_CONSTAN)

-ALCServise.assertAuthorized(PERMISSION_CONSTANT)

Οι αποφάσεις ελέγχου πρόσβασης(access control) γίνονται μέσα από αυτό το απλό API's . Ας δούμε λοιπόν παρακάτω πρακτικές καλής συγγραφής κώδικα:

```
If (AC.hasAccess (''article:edit:12''))
{
// εκτέλεση ενεργειών
}
```

4.4.1 Χρήση κεντρικοποιημένου Controler

Στο επίπεδο παρουσίασης(Presentation layer) :

```
if (isAuthorized(Permission.VIEW_LOG_PANEL))
 {
<h2> Here are the logs</h2>
<%=getLogs ();%/>
 }
```

Στον controler :

```
try (assertAuthorized(Permission.DELETE_USER))
 {
deleteUser();
 } catch (Exception e)
{
 // ειδοποίηση alarm
}
```

4.4.2 Apache SHIRO

Πλέον όταν γράφουμε κώδικα σε μία εταιρία ή και σπίτι μας θα χρησιμοποιήσουμε κάποιο framework το οποίο στην ουσία μας διευκολύνει στην συγγραφή. Πολλά framework στρέφονται και ως προς την ασφάλεια της εφαρμογής προνοώντας για την αντιμετώπιση διάφορων επιθέσεων όπως XSS χρησιμοποιώντας από μόνα τους token κ.α. Ο Apache Shiro είναι ένα δυνατό και εύκολο στην χρήση Java security framework:

Πρόβλημα 1 : Οι δικτυακές εφαρμογές χρειάζονται ένα ασφαλή μηχανισμό ελέγχου πρόσβασης(access control)

Λύση:

```
If ( currentUser.isPermitted(“lightsaber:wield”)) {
  log.info(“You may use a lightsaberring.Use it
wisely.”);
}
else {
 log.info(“Sorry, lightsaber rings are for George
only.”);
}
```

Πρόβλημα 2: Οι δικτυακές εφαρμογές χρειάζονται ένα ασφαλή μηχανισμό για την πρόσβαση σε ένα συγκεκριμένο αντικείμενο(object)

Λύση:

```
  if    (currentUser.isPermitted    (''winnebago:drive:''+
win_id )) {
  log.info(''You are permitted to 'drive' the winnebago
```

```
'with license plate (id) 'eagle5'.Here are the keys -
have fun!'');
  }else{
  log.info(''Sorry, you aren't allowed to drive 'eagle5'
winnebago!'');
  }
```

Εκτός του Apache Shiro υπάρχουν και άλλα security framework όπως της PHP το Laravel που παίρνει διάφορα μέτρα σε σχέση με τις επιθέσεις και τις αδυναμίες την γλώσσας.

4.5 Αντίμετρα κατά του Broken Authentication and Session Management

Προσπαθούμε τα δικά μας συστήματα πιστοποίησης και διαχείρισης sessions (αν είναι απαραίτητο φτιάχνουμε custom απ' την αρχή) να ακολουθούν κάποιες συγκεκριμένες προδιαγραφές ασφαλείας και απαιτήσεις. Ας παραθέσουμε κάποιες λύσεις για την αποφυγή της παραπάνω απειλής

-Ακολουθούμε τα industry standards για την αποθήκευση κωδικών και ευαίσθητων προσωπικών δεδομένων (salted hashing κλπ).

-Χρησιμοποιούμε έτοιμα frameworks και APIs που έχουν ήδη υλοποιημένες τις παραπάνω διαδικασίες για εμάς και βρίσκονται υπό ανάπτυξη και έλεγχο από ειδικούς στον τομέα.

4.6 Αντίμετρα για Insecure Cryptographic Storage

Ο τρόπος με των οποίον τα δεδομένα των χρηστών(κωδικοί,username κ.α) αποθηκεύονται στην βάση δεδομένων τις εφαρμογής είναι ένα μείζων θέμα από μεριάς ασφάλειας απέναντι σε επιτιθέμενους που στοχεύουν στην ανάκτησή της. Για όλα τα ευαίσθητα δεδομένα για τα οποία χρειάζεται κρυπτογράφηση πρέπει να εφαρμόζονται τουλάχιστον τα παρακάτω:

- Όλα τα δεδομένα πρέπει να κρυπτογραφούνται με σωστό τρόπο και σε στάδια που να αποτρέπουν τόσο εξωτερικές αλλά και εσωτερικές επιθέσεις.
- Τα αντίγραφα των δεδομένων μας πρέπει να είναι επίσης σωστά κρυπτογραφημένα, αλλά τα κλειδιά κρυπτογράφησης τους και τα αντίγραφα τους να διαχειρίζονται ξεχωριστά από αυτά.
- Χρήση ισχυρών αλγορίθμων κρυπτογράφησης και ισχυρών κλειδιών.
- Χρήση hashing με σωστό τρόπο για την αποθήκευση των κωδικών πρόσβασης χρηστών.
- Έλεγχος πρόσβασης για όλα τα κλειδιά και τους κωδικούς.

4.7 Κανόνες Ασφαλείας

Φτάνοντας στο τέλος της εργασίας θα δούμε συγκεντρωμένους μερικούς κανόνες που θα πρέπει να ακολουθεί κάθε προγραμματιστής, ώστε να παράγει ασφαλή κώδικα:

- Οι μεταβλητές μας πρέπει να γίνονται πάντα sanitized. Ακόμα και τα cookies, τα HTTP Headers και οτιδήποτε μπορεί να προέλθει από τον χρήστη, πρέπει επίσης να φιλτράρονται από μία συνάρτηση sanitization.
- Η ορθότητα των δεδομένων δεν πρέπει να ελέγχεται μόνο από την πλευρά του χρήστη (για παράδειγμα με Javascript). Σ’ αυτή την περίπτωση, ο κώδικας που

κάνει τον έλεγχο μπορεί να τροποποιηθεί από έναν κακόβουλο χρήστη και να σταλούν δεδομένα με ανορθόδοξο κι ενδεχομένως επιβλαβές περιεχόμενο. Τα δεδομένα του χρήστη, λοιπόν, θα πρέπει να ελέγχονται πάντα και στο server.

- Όταν δημιουργούνται εντολές SQL δυναμικά, θα πρέπει να χρησιμοποιούνται type safe παράμετροι κι όχι παραμέτρους που δέχονται οποιοδήποτε είδος δεδομένων (π.χ., μεταβλητές που δέχονται και ακέραιους και δεκαδικούς και χαρακτήρες).

- Να ελέγχεται πάντα ο κώδικας για προβλήματα ασφάλειας, πριν δοθεί. Θα πρέπει να χρησιμοποιούνται ειδικά προγράμματα επίθεσης (δωρεάν και μη) ώστε να ελέγχεται η γραμμή άμυνας των προγραμμάτων. Τέτοια προγράμματα είναι το Nessus, το Nikto, το Metaspploit, το sqlMap, το sqlNinja και γενικά πολλά προγράμματα που περιέχονται στο KaliLinux

- Δεν πρέπει να καταχωρούνται ευαίσθητα δεδομένα (αριθμούς πιστωτικών καρτών, passwords κ.λπ.) σε μια βάση δεδομένων και χωρίς κρυπτογράφηση.

- Τόσο ο χρήστης της βάσης δεδομένων όσο και ο χρήστης που έχει πρόσβαση στον web server (ο λεγόμενος web user), πρέπει να έχουν τα ελάχιστα δυνατά δικαιώματα. Στην πράξη, πρέπει να έχουν τα απολύτως απαραίτητα δικαιώματα ώστε να λειτουργεί η εφαρμογή μας

- Ακόμα πρέπει να μην εμφανίζετε στον τελικό χρήστη τα πραγματικά λάθη της εφαρμογής τα οποία προέρχονται από κάποιο exception! Επίσης, είναι επικίνδυνο να δίνονται πληροφορίες για τη δομή των καταλόγων και των υποκαταλόγων του δίσκου.

- Πρέπει να «δένονται» όσο πιο καλά μπορούν τα session cookies με τον client, για την αποφυγή φαινομένων impersonation.

- Τα προγράμματα θα πρέπει να αρνούνται τα λεγόμενα αδύναμα passwords. Ένα δυνατό password πρέπει να έχει τουλάχιστον 12 χαρακτήρες και να αποτελείται από κεφαλαία, πεζά, αριθμούς, ειδικούς χαρακτήρες.

Συμπεράσματα

Το λογισμικό πάντα θα έχει κάποια bugsκαι ως αποτέλεσμα αυτών κάποια προβλήματα ασφάλειας. Στόχος θα πρέπει να είναι να μειώσουμε όσο μπορούμε τον αριθμό τους και την σοβαρότητα αυτών που απομένουν. Επίσης η εκμετάλλευση μιας και μόνο αδυναμίας της διαδικτυακής εφαρμογής είναι ικανή να θέσει εκτός λειτουργίας όλη τη διαδικτυακή παρουσία μας , να προκαλέσει απώλεια ή διαρροή δεδομένων κα. Όσο πιο σύντομα ανακαλύπτουμε και διορθώνουμε τα κενά ασφαλείας τόσο μικρότερο είναι το χρονικό «παράθυρο» κατά το οποίο είμαστε ευάλωτοι σε μία επίθεση. Ακόμα η ύπαρξη των αδυναμιών δεν είναι συνώνυμη με την εκμετάλλευση τους. Κάποιος ή κάτι θα πρέπει ενεργά να ασχοληθεί με το πώς θα μπορέσει να προκαλέσει τεχνικά ή οικονομικά προβλήματα, αξιοποιώντας κάποια τεχνική που εκμεταλλεύεται την αδυναμία. Κάθε κακόβουλος χρήστης έχει διαφορετικό σκοπό και κίνητρο. Τέλος κάθε στόχος μπορεί να χωριστεί σε στόχο «ευκαιρίας» και στόχο «επιλογής»:

-Ένας στόχος ευκαιρίας έχει χαμηλότερα πρότυπα ασφαλείας από το μέσο όρο και έτσι η επίθεση εναντίον του αποτελεί «ευκαιρία».

-Ένας στόχος επιλογής είναι κάποιος που διαθέτει δεδομένα ή πληροφορία με μεγάλη αξία για τον κακόβουλο χρήστη, ο οποίος μεθοδικά θα προχωρήσει σε ανάλυση των αδυναμιών που θα μπορέσει να ανακαλύψει ώστε να τις εκμεταλλευτεί κατάλληλα.

Βιβλιογραφία

[1] Σημειώσεις Jim Manico Διάλεξη 28/11/2015

[2] Matteo Meucci and Andrew Muller (2008),*OWASP Testing Guide v 4.0*

[3] OWASP Foundation (2013), *OWASP Top 10* διαθέσιμο στο https://www.owasp.org/index.php/ Top_10_2013-Top_10 (πρόσβαση: 29-6-2015).

[4] Chris Shiflett (2005), *Essential PHP Security*

[5] Παναγιώτης Βαρελάς (2012),*Deltahacker*,Parabig Creations

[6] R. H. Williams (2007), «Introduction to Information Security Concepts»

[7] S. Bosworth και M.E. Kabay(2002), *Computer Security Handbook*, Wiley

[8] «About The Open Web Application Security Project», διαθέσιμοστο

https://www.owasp.org/index.php/ About_OWASP (πρόσβαση: 29-6-2015).

[9] «XPath Injection», διαθέσιμο στο

http://www.webappsec.org/projects/threat/classes/xpath_injection.shtml (πρόσβαση: 29-6-2015).