Στατική Ανάλυση Ύποπτου Λογισμικού με το Ghidra - Δημιουργία κανόνων YARA

Τεχνικές και Εργαλεία Στατικής Ανάλυσης Κακόβουλου Λογισμικού- YARA

Ασφάλεια Υπολογιστών & Δικτύων

ΚΟΣΜΑΣ ΑΡΧΟΝΤΗΣ, 1084020

Πανεπιστήμιο Πατρών

15 Ιανουαρίου 2025

Περιεχόμενα

1 Εισαγωγή

Σκοπός αυτής της αναφοράς είναι η παρουσίαση της στατικής ανάλυσης ενός δοκιμαστικού εκτελέσιμου αρχείου, με σκοπό την αναγνώριση του ως κακόβουλου ή καλοήθους, χρησιμοποιώντας το λογισμικό Ghidra για reverse engineering. Ύστερα, εφόσον κριθεί κακόβουλο, κατασκευάζονται κανόνες YARA για αργότερη αναγνώριση του.

Θα γίνει σύντομη παρουσίαση του Ghidra, βήμα-βήμα παρουσίαση της ανάλυσης επίμαχων/ύποπτων σημείων του προγράμματος, επεξήγησή του και προτάσεις για τους κανόνες.

2 Λογισμικό Ghidra

2.1 Παρουσίαση

Το Ghidra είναι ένα ισχυρό reverse engineering εργαλείο που αναπτύχθηκε από την Υπηρεσία Εθνικής Ασφάλειας των ΗΠΑ (National Security Agency, NSA). Κυκλοφόρησε τον Μάρτιο του 2019, και ο κύριος σκοπός του είναι η ανάλυση compiled λογισμικού για καλύτερη κατανόηση του κώδικά του και της συμπεριφοράς του.

Είναι γραμμένο με τη γλώσσα προγραμματισμού Java, και περιλαμβάνει γραφική διεπαφή που διευκολύνει πολλές λειτουργίες στατικής ανάλυσης, αλλά μπορεί να τρέξει και από το τερματικό.

Για πολλά χρόνια χρησιμοποιούνταν ως εσωτερικό εργαλείο της NSA, πρωτού η υπηρεσία να ελευθερώσει την κυκλοφορία του, και πλέον είναι ευρέως διαδεδομένο στην κοινότητα κυβερνοασφάλειας ως ένα χρήσιμο εργαλείο.

2.2 Λειτουργίες

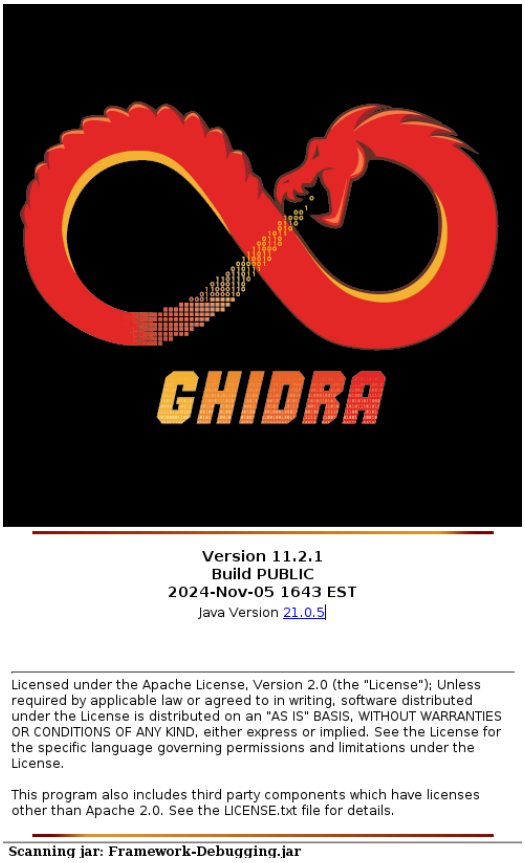
Το Ghidra είναι γνωστό για την εκτεταμένη συλλογή εργαλείων και λειτουργιών του, καθιστώντας το μια ευέλικτη πλατφόρμα για σκοπούς reverse engineering. Παρακάτω παρατίθενται ορισμένες από τις βασικές του λειτουργίες:

- **Disassembly**: Το Ghidra μετατρέπει τις δυαδικές εντολές της γλώσσας μηχανης σε κώδικα assembly, η θεμελιώδης ενέργεια για ανάλυση αρχείων και εκτελέσιμων, κατανόηση της αρχιτεκτονικής τους καθώς και τη ροή των εντολών.

- **Decompilation**: Επιπλέον από disassembler, το λογισμικό επίσης παρέχει λειτουργίες απομεταγλώττισης/decompilation, μετατρέποντας τη γλώσσα μηχανής σε υψηλού επιπέδου γλώσσας προγραμματισμού, παρόμοιας σύνταξης με τη C/C++ - κάνοντας το ακόμα πιο φιλικό στον χρήστη, και διευκολύνοντας της κατανόησης της δομής και συμπεριφοράς του προγράμματος, εν όψει απώλειας του πηγαίου κώδικα.

- **Εύρεση, Αναζήτηση και Ανάλυση Συμβολοσειρών**: Το Ghidra περιλαμβάνει δυνανότητες αναγνώρισης και αναζήτησης string, επιτρέποντας τον εντοπισμό hardcoded πληροφορίας εντός του δυαδικού κώδικα. Αυτή η λειτουργία είναι χρήσιμη για εύρεση file paths, μηνύματα σφάλματος, διαδικτυακά domains και άλλες πιθανές ενδείξεις λειτουργίας κακόβουλου λογισμικού.
- **Ανάλυση και Ροή Συναρτήσεων**: Το Ghidra παρέχει εργαλεία για ανάλυση και οπτικοποίηση της ροής των συναρτήσεων. Οι χρήστες μπορούν να εντοπίσουν πώς καλούνται οι συναρτήσεις, τις παραμέτρους τους και τις τιμές επιστροφής. Αυτή η λειτουργία είναι ιδιαίτερα χρήσιμη για την κατανόηση περίπλοκου λογισμικού με σύνθετα γραφήματα κλήσεων (function call graphs).

Αλλά και πολλές άλλες, όπως ανάλυση πιο κοντά στη δυναμική, καθώς και debugger.



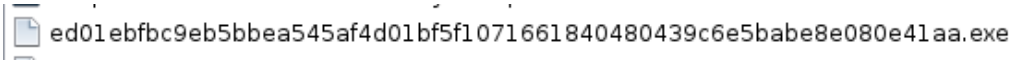
Σχήμα 2.1: Φόρτωση και η έκδοση σε χρήση του Ghidra.

3 Ανάλυση του Εκτελέσιμου

3.1 Περιβάλλον

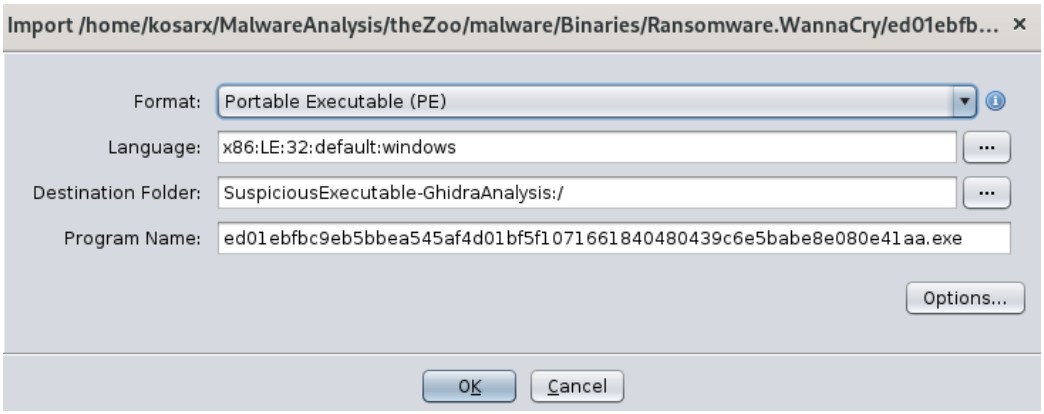
Εφόσον διατρέχεται ο κίνδυνος για μόλυνση του συστήματος από κακόβουλο λογισμικό, επιθυμούμε να εκτελέσουμε την ανάλυσή μας σε ένα απομονωμένο και ελεγχόμενο περιβάλλον.

Κατασκευάζουμε μια Debian Εικονική Μηχανή (Virtual Machine) στο Virtual Box, όπου εγκαθιστούμε το Ghidra και τοποθετούμε το ύποπτο πρόγραμμα σε αυτή.



Σχήμα 3.1: Το εκτελέσιμο υπό εξέταση στο file system.

Δημιουργούμε καινούργιο project για την ανάλυσή μας στο Ghidra, και κάνουμε import το εκτελέσιμο.

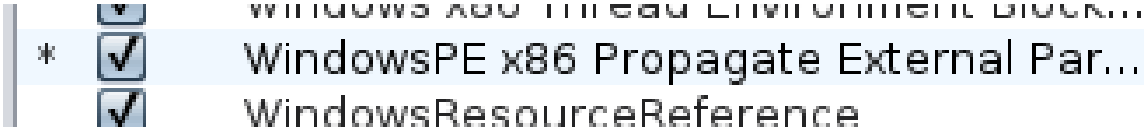


Σχήμα 3.2: Η εισαγωγή του ύποπτου εκτελέσιμου στο Ghidra.

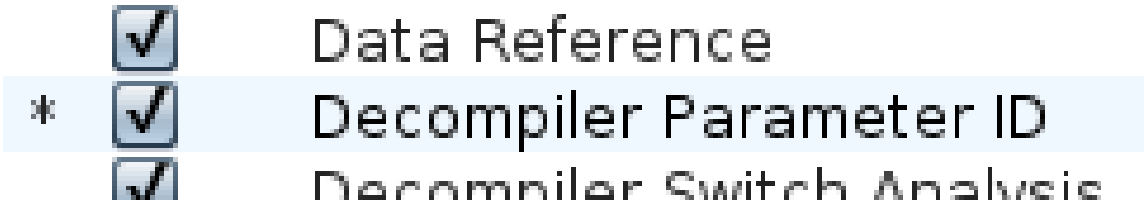
Παρατηρούμε ότι το λογισμικό αναγνωρίζει το αρχείο ως προοριζόμενο για

συστήματα Windows, κάτι που πιθανόν να δείχνει ότι η εκτέλεση του αρχείου στο Debian VM μας δεν είναι δυνατή.

Εφόσον θα πραγματοποιήσουμε στατική ανάλυση σε ένα εκτελέσιμο αρχείο, προσθέτουμε μερικές επιπλέον ρυθμίσεις/επιλογές στην αυτόματη ανάλυση που πραγματοποιεί το Ghidra:

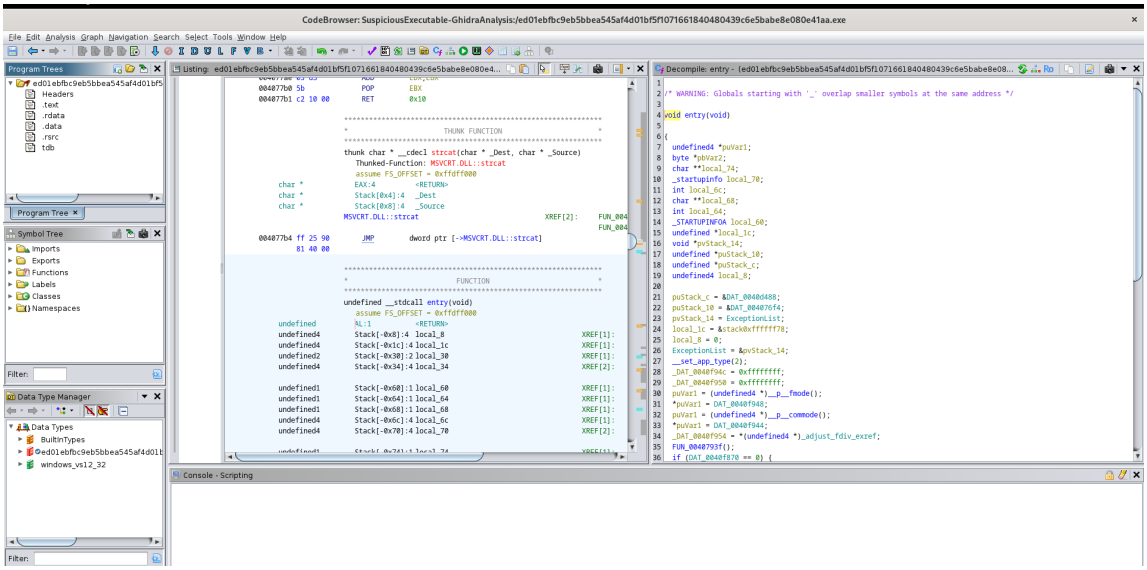


Σχήμα 3.3: Προσθήκη της επιλογής "Windows PE x86 Propagate External Parameters" στην ανάλυση.



Σχήμα 3.4: Προσθήκη της επιλογής "Decompiler Parameter ID" στην ανάλυση.

Και καταλήγουμε στην παρακάτω εικόνα του λογισμικού:

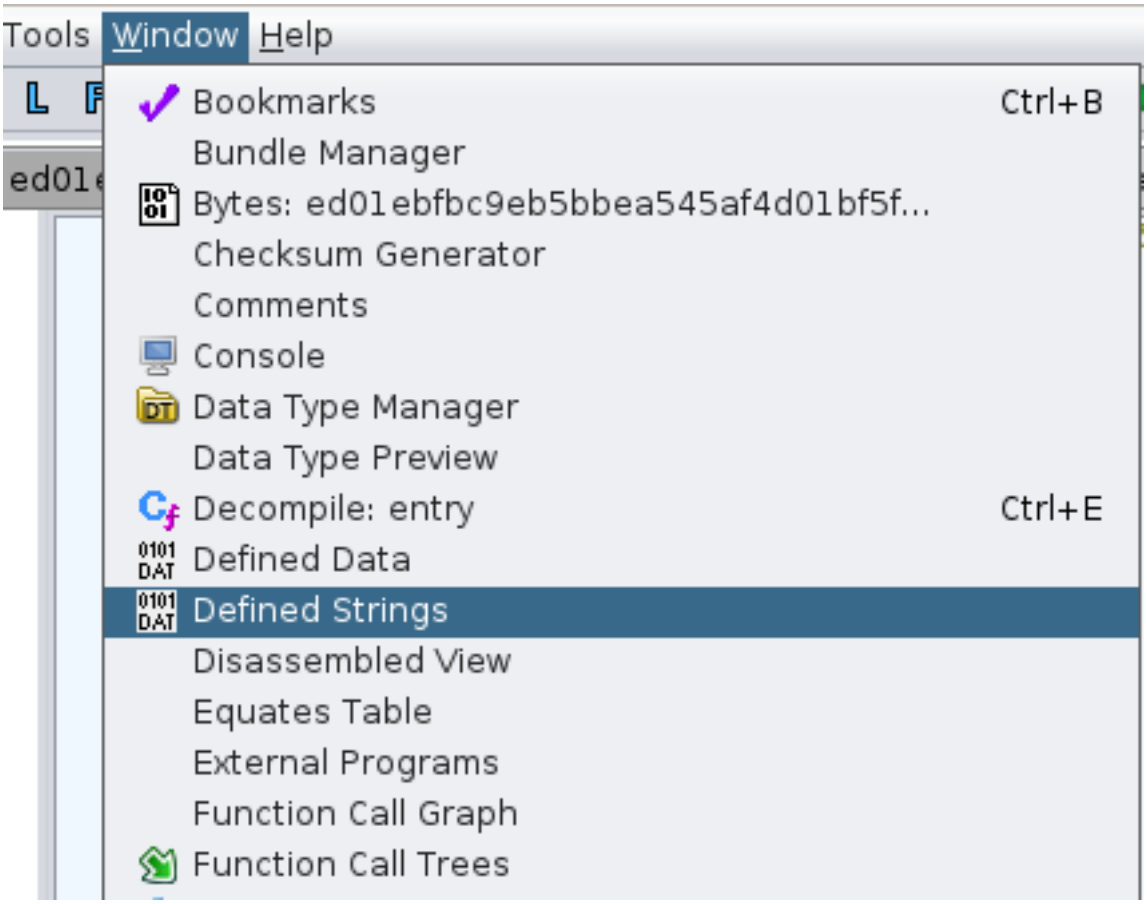


Σχήμα 3.5: Η αρχική μας οθόνη του Ghidra μετά τη πραγματοποιήση της ανάλυσης, αριστερά ο disassembled κώδικας και δεξιά ο decompiled.

3.2 Κύρια Ανάλυση

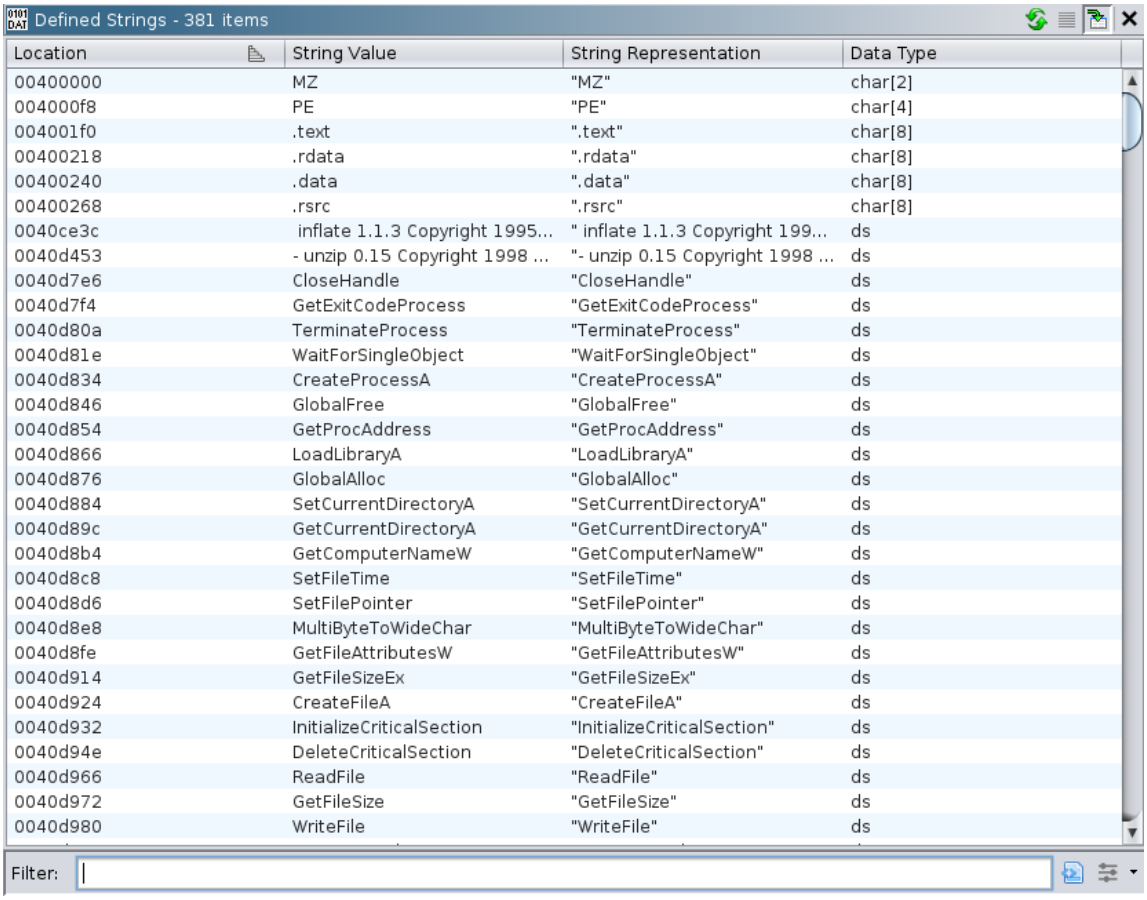
3.2.1 Αναζήτηση και Ανάλυση Συμβολοσειρών

Το πρώτο βήμα προς την ανάλυση του εκτελέσιμου είναι η ανεύρεση και εξερεύνηση των hardcoded strings που περιέχει. Για να το κάνουμε αυτό, στην επιλογή του μενού "Window", διαλέγουμε "Define Strings":



Σχήμα 3.6: Η επιλογή "Defined Strings" στο μενού "Window" του Ghidra.

Και μπορούμε να δούμε πλέον το μενού με τις συμβολοσειρές που έχει εντοπίσει το Ghidra.

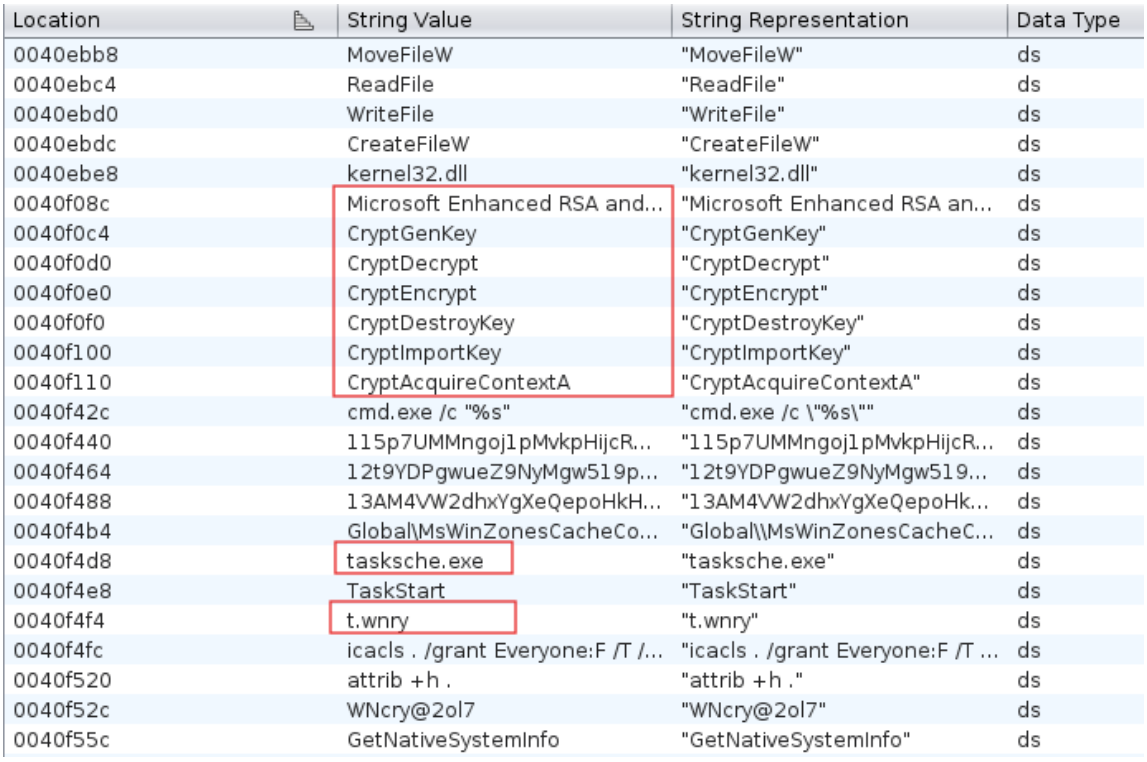


Σχήμα 3.7: Η λίστα με τα strings του ύποπτου προγράμματος.

Αρχικά, παρατηρούμε συναρτήσεις που αντιστοιχούν στο Windows API, και μπορούμε να υποθέσουμε ότι το πρόγραμμα κάνει χρήση αυτών για διαχείριση αρχείων και φακέλων, δημιουργία και τερματισμό διεργασιών και άλλες λειτουργίες.

Εξερευνώντας την πληθώρα των συμβολοσειρών που υπάρχουν, αναζητούμε για τυχόν αναφορές σε ύποπτα file extensions, όπως “`.exe`”, διαδικτυακές ενέργειες μέσω των API ή hardcoded domains, “http”, αλλά και άλλα, απλά strings που πιθανώς να είναι ενδεικτικά κακοηθών προθέσεων.

Η αναζήτησή μας οδηγεί στα παρακάτω ενδιαφέροντα strings που αξίζουν ανάλυση:



Location	String Value	String Representation	Data Type
0040ebb8	MoveFileW	"MoveFileW"	ds
0040ebc4	ReadFile	"ReadFile"	ds
0040ebd0	WriteFile	"WriteFile"	ds
0040ebdc	CreateFileW	"CreateFileW"	ds
0040ebe8	kernel32.dll	"kernel32.dll"	ds
0040f08c	Microsoft Enhanced RSA and...	"Microsoft Enhanced RSA an...	ds
0040f0c4	CryptGenKey	"CryptGenKey"	ds
0040f0d0	CryptDecrypt	"CryptDecrypt"	ds
0040f0e0	CryptEncrypt	"CryptEncrypt"	ds
0040f0f0	CryptDestroyKey	"CryptDestroyKey"	ds
0040f100	CryptImportKey	"CryptImportKey"	ds
0040f110	CryptAcquireContextA	"CryptAcquireContextA"	ds
0040f42c	cmd.exe /c "%s"	"cmd.exe /c \"%s\""	ds
0040f440	115p7UMMngoj1pMvkpHijcR...	"115p7UMMngoj1pMvkpHijcR...	ds
0040f464	12t9YDPgwueZ9NyMgw519p...	"12t9YDPgwueZ9NyMgw519...	ds
0040f488	13AM4VW2dhxYgXeQepoHkH...	"13AM4VW2dhxYgXeQepoHk...	ds
0040f4b4	Global\MsWinZonesCacheCo...	"Global\\MsWinZonesCacheC...	ds
0040f4d8	tasksche.exe	"tasksche.exe"	ds
0040f4e8	TaskStart	"TaskStart"	ds
0040f4f4	t.wnry	"t.wnry"	ds
0040f4fc	icacls . /grant Everyone:F /T /...	"icacls . /grant Everyone:F /T ...	ds
0040f520	attrib +h .	"attrib +h ."	ds
0040f52c	WNcry@2ol7	"WNcry@2ol7"	ds
0040f55c	GetNativeSystemInfo	"GetNativeSystemInfo"	ds

Σχήμα 3.8: Οι συμβολοσειρές υπό εξέταση, σημειωμένες με κόκκινο πλαίσιο.

Παρατηρούμε ότι το πρόγραμμα κάνει χρήση κρυπτογραφικών δυνατοτήτων μέσω της βιβλιοθήκης των Windows, που θα αναλύσουμε περισσότερο αργότερα. Επίσης διακρίνουμε τα ονόματα αρχείου με τα οποία, πιθανώς, αλληλεπιδρά με κάποιο τρόπο το ύποπτο πρόγραμμά μας, “tasksche.exe” και “t.wnry”.

Το πρώτο είναι εκτελέσιμο αρχείο, το δεύτερο είναι άγνωστος τύπου αρχείου και πιθανώς εσωτερικός στη λειτουργία του προγράμματος.

Επιπλέον, εντοπίζουμε τις παρακάτω συγκεκριμένες συμβολοσειρές:

Location	String Value	String Representation	Data Type
0040ebb8	MoveFileW	"MoveFileW"	ds
0040ebc4	ReadFile	"ReadFile"	ds
0040ebd0	WriteFile	"WriteFile"	ds
0040ebdc	CreateFileW	"CreateFileW"	ds
0040ebe8	kernel32.dll	"kernel32.dll"	ds
0040f08c	Microsoft Enhanced RSA and...	"Microsoft Enhanced RSA an...	ds
0040f0c4	CryptGenKey	"CryptGenKey"	ds
0040f0d0	CryptDecrypt	"CryptDecrypt"	ds
0040f0e0	CryptEncrypt	"CryptEncrypt"	ds
0040f0f0	CryptDestroyKey	"CryptDestroyKey"	ds
0040f100	CryptImportKey	"CryptImportKey"	ds
0040f110	CryptAcquireContextA	"CryptAcquireContextA"	ds
0040f42c	cmd.exe /c "%s"	"cmd.exe /c \"%s\""	ds
0040f440	115p7UMMngoj1pMvkpHijcR...	"115p7UMMngoj1pMvkpHijcR...	ds
0040f464	12t9YDPgwueZ9NyMgw519p...	"12t9YDPgwueZ9NyMgw519...	ds
0040f488	13AM4VW2dhxYgXeQepoHkH...	"13AM4VW2dhxYgXeQepoHk...	ds
0040f4b4	Global\MsWinZonesCacheCo...	"Global\\MsWinZonesCacheC...	ds
0040f4d8	tasksche.exe	"tasksche.exe"	ds
0040f4e8	TaskStart	"TaskStart"	ds
0040f4f4	t.wnry	"t.wnry"	ds
0040f4fc	icacls . /grant Everyone:F /T /...	"icacls . /grant Everyone:F /T ...	ds
0040f520	attrib +h .	"attrib +h ."	ds
0040f52c	WNcry@2ol7	"WNcry@2ol7"	ds
0040f55c	GetNativeSystemInfo	"GetNativeSystemInfo"	ds

Σχήμα 3.9: Η συμβολοσειρά “WNcry@2oI7” στη λίστα των Defined Strings.

0040eb64	.docx	u".docx"	unicode
0040eb70	.doc	u".doc"	unicode
0040eb7c	WANACRY!	"WANACRY!"	ds
0040eb88	%s\%s	u"%s\\%s"	unicode
0040eb94	CloseHandle	"CloseHandle"	ds
0040eba0	DeleteFileW	"DeleteFileW"	ds

Σχήμα 3.10: Η συμβολοσειρά “WANACRY!” στη λίστα των Defined Strings.

Που φαίνεται να συσχετίζονται με το προηγούμενο file extension “.wnry”, ενώ ο σκοπός τους δεν γίνεται ξεκάθαρος. Αν ακολουθήσουμε τη χρήση του string “WANACRY!”, σύμφωνα με το cross-reference feature του Ghidra:

Σχήμα 3.11: Η συμβολοσειρά “WANACRY!” αναφέρεται σε κάποιο σημείο στην εκτέλεση.

Μας οδηγεί να εξετάσουμε τη συνάρτηση `FUN_004014a6`, στο σημείο μνήμης *00401566*, και πως χρησιμοποιεί το δηλωμένο string `s_WANACRY!_0040eb7c` (το οποίο αντιστοιχεί στο “WANACRY!”).

```
if ((local_28.s.HighPart < 1) && ((local_28.s.HighPart < 0 || (local_28.s.LowPart < 0x6400001)))
   ) {
  iVar2 = (*DAT_0040f880)(hFile,&local_240,8,local_20,0);
  if (iVar2 != 0) {
    iVar2 = memcmp(&local_240,s_WANACRY!_0040eb7c,8);
    if (iVar2 == 0) {
      iVar2 = (*DAT_0040f880)(hFile,local_248,4,local_20,0);
      if ((iVar2 != 0) && (local_248[0] == 0x100)) {
        iVar2 = (*DAT_0040f880)(hFile,*(undefined4 *)((int)this + 0x4c8),0x100,local_20,0);
```

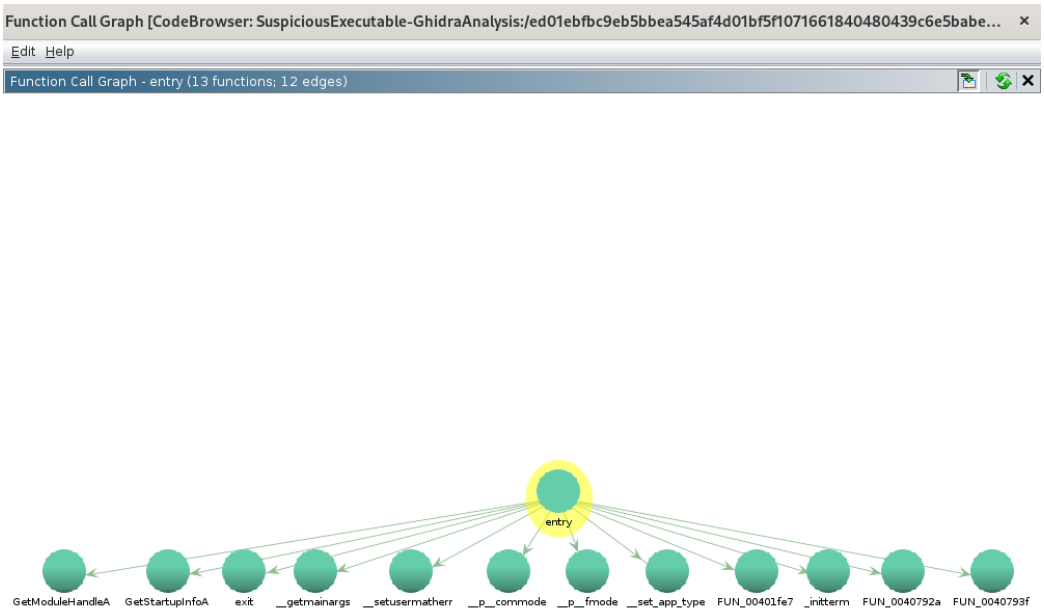
Σχήμα 3.12: Η συμβολοσειρά “WANACRY!” σε χρήση στην συνάρτηση `FUN_004014a6`.

Η τοπική μεταβλητή “`local_240`” αφορά το αρχείο “`t.wnry`” μέσω των παραμέτρων της συνάρτησης που καλείται.

Μπορούμε να συνειδητοποιήσουμε ότι συγκρίνει τα πρώτα 8 bytes του αρχείου αυτού με τη συμβολοσειρά “WANACRY!” και πράττει αναλόγως το αποτέλεσμα - πιθανόν για επιβεβαίωση και έγκυρη συνέχεια ενεργειών.

3.2.2 Απόκρυψη και Απελευθέρωση Δικαιωμάτων Διαχείρισης και Πρόσβασης Αρχείων

Αν επιστρέψουμε στην ανάλυση κώδικα, η είσοδος του προγράμματος βρίσκεται στη συνάρτησή του “`entry`”, η οποία πρόκειται για τη κλασική διαχείριση προγραμμάτων από το λειτουργικό σύστημα. Κοιτώντας το function call graph της “`entry`”:



Σχήμα 3.13: Οι συναρτήσεις που καλέι η είσοδος του προγράμματος.

Οι 2 τελευταίες συναρτήσεις, "FUN_0040792a" και "FUN_0040793f", αποτελούν cleanup και τελικής διαχείρισης, ενώ η καρδιά του προγράμματος βρίσκεται στη "FUN_00401fe7". Αν εξετάσουμε και τις δικές της εντολές, μπορούμε να εντοπίσουμε τις παρακάτω συναρτήσεις:

```
48  SetCurrentDirectoryA(&local_210);
49  FUN_004010fd(1);
50  FUN_00401dab((HMODULE)0x0);
51  FUN_00401e9e();
52  FUN_00401064(s_attrib_+h_._0040f520,0,(LPDWORD)0x0);
53  FUN_00401064(s_icacls_._/grant_Everyone:F_/T_/C_0040f4fc,0,(LPDWORD)0x0);
54  iVar7 = FUN_0040170a();
55  if (iVar7 != 0) {
```

Σχήμα 3.14: Η συνάρτηση "FUN_00401064" καλείται με 2 διαφορετικές παραμέτρους.

Τα συγκεκριμένα strings "s_attrib_+h_._0040f520", "s_icacls_._/grant_Everyone:F_/T_/C_0040f4fc" επίσης εντοπίστηκαν στην προηγούμενη ανάλυση των συμβολοσειρών:

Location	String Value	String Representation	Data Type
0040ebb8	MoveFileW	"MoveFileW"	ds
0040ebc4	ReadFile	"ReadFile"	ds
0040ebd0	WriteFile	"WriteFile"	ds
0040ebdc	CreateFileW	"CreateFileW"	ds
0040ebe8	kernel32.dll	"kernel32.dll"	ds
0040f08c	Microsoft Enhanced RSA and...	"Microsoft Enhanced RSA an...	ds
0040f0c4	CryptGenKey	"CryptGenKey"	ds
0040f0d0	CryptDecrypt	"CryptDecrypt"	ds
0040f0e0	CryptEncrypt	"CryptEncrypt"	ds
0040f0f0	CryptDestroyKey	"CryptDestroyKey"	ds
0040f100	CryptImportKey	"CryptImportKey"	ds
0040f110	CryptAcquireContextA	"CryptAcquireContextA"	ds
0040f42c	cmd.exe /c "%s"	"cmd.exe /c \"%s\""	ds
0040f440	115p7UMMngoj1pMvkpHijcR...	"115p7UMMngoj1pMvkpHijcR...	ds
0040f464	12t9YDPgwueZ9NyMgw519p...	"12t9YDPgwueZ9NyMgw519...	ds
0040f488	13AM4VW2dhxYgXeQepoHkH...	"13AM4VW2dhxYgXeQepoHk...	ds
0040f4b4	Global\MsWinZonesCacheCo...	"Global\\MsWinZonesCacheC...	ds
0040f4d8	tasksche.exe	"tasksche.exe"	ds
0040f4e8	TaskStart	"TaskStart"	ds
0040f4f4	t.wnry	"t.wnry"	ds
0040f4fc	icacls . /grant Everyone:F /T /...	"icacls . /grant Everyone:F /T ...	ds
0040f520	attrib +h .	"attrib +h ."	ds
0040f52c	WNcry@2ol7	"WNcry@2ol7"	ds
0040f55c	GetNativeSystemInfo	"GetNativeSystemInfo"	ds

Σχήμα 3.15: Τα δύο strings στην λίστα των defined strings.

Και περνάνε ως παράμετροι στη συνάρτηση "`FUN_00401064`", η οποία με απλή ανάλυση φαίνεται να τα εκτελεί. Μπορούμε να συμπεράνουμε ότι σκοπός αυτής της συνάρτησης, μαζί με τις συμβολοσειρές, έχει σκοπό την απόκρυψη των αρχείων του current working directory, μέσω της εντολής `attrib +h`. Ενώ επίσης ξεκλειδώνει όλα τα δικαιώματα πρόσβασης και διαχείρισης των αρχείων στον ίδιο φάκελο, με την αντίστοιχη εντολή. Μπορούμε να σημειώσουμε ότι είναι εξαιρετικά ασυνήθιστες ενέργειες.

3.2.3 Δημιουργία και Εκτέλεση πολλών Service

Ενδιαφέρον παρουσιάζει επίσης να σημειώσουμε ότι διάφορες συναρτήσεις του εκτελέσιμου έχουν να κάνουν με δημιουργία, εκκίνηση και διαχείριση Services, όπως φαίνεται στη παρακάτω συνάρτηση:

```
undefined4 __cdecl FUN_00401ce8(undefined4 param_1)

{
  undefined4 uVar1;
  SC_HANDLE hService;
  char local_410 [1024];
  SC_HANDLE local_10;
  undefined4 local_c;
  SC_HANDLE local_8;

  local_c = 0;
  local_8 = OpenSCManagerA((LPCSTR)0x0,(LPCSTR)0x0,0xf003f);
  if (local_8 == (SC_HANDLE)0x0) {
    uVar1 = 0;
  }
  else {
    local_10 = OpenServiceA(local_8,(LPCSTR)&lpMultiByteStr_0040f8ac,0xf01ff);
    if (local_10 == (SC_HANDLE)0x0) {
      sprintf(local_410,s_cmd.exe_/c_"%s"_0040f42c,param_1);
      hService = CreateServiceA(local_8,(LPCSTR)&lpMultiByteStr_0040f8ac,
                                (LPCSTR)&lpMultiByteStr_0040f8ac,0xf01ff,0x10,2,1,local_410,
                                (LPCSTR)0x0,(LPDWORD)0x0,(LPCSTR)0x0,(LPCSTR)0x0,(LPCSTR)0x0);
      uVar1 = local_c;
      if (hService != (SC_HANDLE)0x0) {
        StartServiceA(hService,0,(LPCSTR *)0x0);
        CloseServiceHandle(hService);
        local_c = 1;
        uVar1 = local_c;
      }
    }
    else {
      StartServiceA(local_10,0,(LPCSTR *)0x0);
      CloseServiceHandle(local_10);
      uVar1 = 1;
    }
    CloseServiceHandle(local_8);
  }
  return uVar1;
}

```

Σχήμα 3.16: Μια συνάρτηση που διαχειρίζεται Services.

Αξίζει να σημειωθεί ότι η χρήση των Service είναι κοινή τακτική των κακόβουλων λογισμικών για persistence μεταξύ reboots.

3.2.4 Χρήση των API και συνδυασμός

Όπως ήδη συζητήσαμε, το πρόγραμμα κάνει χρήση μερικών API και βιβλιοθηκών, όπως της κρυπτογραφικής βιβλιοθήκης, μέσω του “advapi32.dll”, βασικές διαχείρισης συστήματος μέσω του “kernel32.dll” και άλλων.

Δεν είναι ενδεικτικό κακόβουλης συμπεριφοράς, αλλά είναι επίσης μια συνιστώσα που μπορούσε να μας βοηθήσει στην ανάλυση.

4 Αποτελέσματα & Συζήτηση

4.1 Αποτελέσματα ανάλυσης

Ο συνδυασμός των παραπάνω τεσσάρων σημείων είναι λογικό να μας οδηγήσει στη διαπίστωση ότι αν το εκτελέσιμο δεν είναι malware, σίγουρα ενεργεί ύποπτα και με επικίνδυνες πράξεις στο σύστημα. Η ανάγνωση και διαχείριση άγνωστων τύπων αρχείων, η απόκρυψη και αφαίρεση περιορισμών στην ανάγνωση και εγγραφή των αρχείων, σε συνδυασμό με την ύπαρξη κρυπτογραφικών δυνατοτήτων και δημιουργία Services σκιαγραφίζουν μια είκονα λογισμικού που δεν πρέπει να εκτελεστεί στο σύστημα-θύμα.

Για να κατηγοριοποιήσουμε το λογισμικό ως κακόβουλο ή μη καθοριστικά, πρέπει να υπάρξει και το στάδιο της Δυναμικής Ανάλυσης, όπου παρατηρείται ενεργά η συμπεριφορά του. Όσον αφορά τη Στατική Ανάλυση του αρχείου όμως, το εκτελέσιμο πρόκειται για ένα **κακόβουλο**.

4.2 Το Εκτελέσιμο υπό Ανάλυση

4.2.1 Ιστορικό

Στην πραγματικότητα, το λογισμικό που εξετάζουμε πρόκειται για μέρος του διάσημου πλέον ransomware WannaCry, που εμφανίστηκε και επιτέθηκε σε υπολογιστές το 2017, και εκτιμάται ότι μόλυνε πάνω από 300.000 υπολογιστές σε 150 διαφορετικές χώρες [7].

Σχήμα 4.1: Εικόνα που εμφανίζεται μετά την επίθεση του κακόβουλου λογισμικού.

Το λογισμικό εκμεταλλευόταν το EternalBlue, ένα vulnerability στο πρωτόκολλο SMBv1 της Microsoft, που ήταν γνωστό πολύ καιρό από την NSA, αλλά διατηρούνταν κρυφό για δικούς της σκοπούς. Μετά την διαρροή του, ωστόσο, αναγκάστηκε να ενημερώσει την Microsoft, η οποία έστειλε ενημερώσεις στους ευάλωτους υπολογιστές - όμως το ransomware κατάφερε να διαχυθεί μέσω unpatched συστημάτων [7].

4.2.2 tasksche.exe

Το συγκεκριμένο εκτελέσιμο που εξετάζουμε, όπως είπαμε, πρόκειται για κομμάτι του WannaCry - συγκεκριμένα το εκτελέσιμο tasksche.exe, και δεν πρόκειται για το πρώτο βήμα/σημείο εισόδου του malware στο σύστημα.

Παρακάτω δείχνουμε όλα τα σχετικά αρχεία του λογισμικού [7]:

Filename	Size (bytes)	Description	Filetype
mssecsvc.exe	3723264	Loader + Worm Component	EXE
tasksche.exe	3514368	Loader	EXE
Unavailable	65536	Encryptor	DLL
@WanaDecryptor@.exe	245760	Decryptor	EXE

Πίνακας 4.1: Λεπτομέρειες αρχείων σχετικά με το WannaCry ransomware.

Ο συγκεκριμένος πίνακας σχολιάζει το αρχείο *tasksche.exe* ως Loader, αλλά χωρίς το Worm Component το οποίο είναι επιπλέον μέρος του *mssecsvc.exe* - και εκμεταλλεύεται την ευπάθεια που συζητήσαμε προηγουμένως.

Σκοπός του είναι η φόρτωση του εργαλείου κρυπτογράφησης *t.wnry* (το οποίο το ίδιο είναι κρυπτογραφημένο) και σε συνδυασμό με το configuration file *c.wnry* να κρυπτογραφήσει το σύνολο των αρχείων του υπολογιστικού συστήματος που μπορεί να αναγνωρίσει και να διαχειριστεί.

Filename	MD5 Hash	Description
r.wnry	3e0020fc529b1c2a061016dd2469ba96	Text ransom notes
t.wnry	ad4c9de7c8c40813f200ba1c2fa33083	Zip file containing Tor files
s.wnry	5dcaac857e695a65f5c3ef1441a73a8f	Encrypted encryption tool
taskdl.exe	4fef5e34143e646dbf9907c4374276f5	*.WNCRYT file deletion tool
taskse.exe	8495400f199ac77853c53b5a3f278f3e	Utility used to launch decryption tool
u.wnry	7bf2b57f2a205768755c07f238fb32cc	Decryption tool
b.wnry	c17170262312f3be7027bc2ca825bf0c	Ransom image (BMP)
c.wnry	ae08f79a0d800b82fcbe1b43cdbdbefc	Configuration data

Πίνακας 4.2: WannaCry XIA Resources Λεπτομέρειες [7]

Επιπλέον δημιουργεί 2 διαφορετικά Service, με τυχαία ονόματα, ώστε να επιτύχει persistence του malware μεταξύ των reboots του συστήματος.

5 Κανόνες YARA

Εφόσον έχουμε εντοπίσει 4 άξονες ύποπτης συμπεριφοράς του προγράμματος, μπορούμε να δημιουργήσουμε αντίστοιχους κανόνες YARA για μελλοντικό εντοπισμό του λογισμικού ως κακόβουλο.

Οι παρακάτω κανόνες είναι ενδεικτικοί, και όχι απαραιτήτως βέλτιστη, εφόσον πιθανόν να προκαλούν πολλά false positives.

5.1 Αναζήτηση και Ανάλυση Συμβολοσειρών

Το πιο ενδεικτικό ως προς τη φύση ενός προγράμματος ως κακόβουλο είναι τα strings που περιέχει, και ακολουθώντας αυτή τη διαπίστωση διαμορφώνουμε τον παρακάτω κανόνα YARA:

```
rule wannacry_hardcoded_strings {
        meta:
                description = "Detect WannaCry tasksche.exe hardcoded strings"

        strings:
                $stringWANACRY = "WANACRY!" fullword
                $stringWNcry = "WNcry@2ol7" fullword
                $stringwnryFileExt = ".wnry" nocase
                $stringExecutable = "tasksche.exe" fullword

        condition:
                any of them
}
```

Σχήμα 5.1: Ο κανόνας που αντιστοιχεί στην ανίχνευση συμβολοσειρών.

5.2 Απόκρυψη και Απελευθέρωση Δικαιωμάτων Διαχείρισης και Πρόσβασης Αρχείων

Αν και όχι άμεσο χαρακτηριστικό του malware WannaCry, αξίζει να ανιχνευθούν οι εντολές πέρι file access rights manipulation που πραγματοποιεί το *tasksche.exe* αρχείο.

```
rule suspicious_file_commands {

        meta:
                description = "Detect suspicious hiding of files or granting of rights"

        strings:
                $cmd1 = "attrib +h ." ascii
                $cmd2 = "icacls . /grant Everyone:F /T /C /Q" ascii

        condition:
                any of them
}
```

Σχήμα 5.2: Ο κανόνας που αντιστοιχεί στην ανίχνευση μεταχείρισης των δικαιωμάτων αρχείων και απόκρυψής τους.

5.3 Δημιουργία και Εκτέλεση πολλών Service Χρήση των API και συνδυασμός

Οι δύο τελευταίες ενδείξεις είναι και οι πιο αδύναμες, και πιο επιρρεπείς σε false positives. Επομένως, γίνονται πιο αυστηρές οι συνθήκες ενεργοποίησης τους απαιτώντας την συν-υπαρξή τους.

```
rule creating_services {
    meta:
        description = "Detect whether the program is creating services"

    strings:
        $createService = "CreateServiceA" ascii
        $startService = "StartServiceA" asciiSS

    condition:
        $createService or $startService
}
```

Σχήμα 5.3: Ο κανόνας που αντιστοιχεί στην δημιουργία και εκκίνηση Services.

```
rule suspicious_API_Usage {
    meta:
        description = "Detect whether the program has suspicious API usage"

    strings:
        $cryptApi1 = "CryptEncrypt" // cryptography
        $cryptApi2 = "CryptGenKey"
        $cryptApi3 = "CryptImportKey"
        $cryptApi4 = "CryptDestroyKey"
        $cryptApi5 = "CryptAcquireContextA"
        $fileApi1 = "CreateFileA" // CRUD and files
        $fileApi2 = "DeleteFileA"
        $fileApi3 = "WriteFile"
        $fileApi4 = "SetCurrentDirectoryW"
        $processApi1 = "CreateProcessA"

    condition:
        $processApi1 or (
        any of ($cryptApi*) or
        2 of ($fileApi*)
        )
}
```

Σχήμα 5.4: Ο κανόνας που αντιστοιχεί σε συνδυαστική χρήση ύποπτων API.

```
rule suspicious_services_and_API_usage {
    meta:
        description = "Flag as suspicious if there's both suspicious API usage and creation of services"

    condition:
        creating_services and suspicious_API_Usage
}
```

Σχήμα 5.5: Ο συνδυασμός των προηγούμενων δύο κανόνων.

6 Συμπεράσματα

Έχοντας εξετάσει το ύποπτο λογισμικό, μπορούμε πλέον, με χρήση του εργαλείου Ghidra, να εξετάσουμε αναπαραστάσεις του source code που οδήγησαν στο συγκεκριμένο εκτελέσιμο - το λεγόμενο reverse engineering. Η ενδελεχής ανάλυση αυτού του ανακατασκευασμένου κώδικα είναι απαραίτητη για ανεύρεση strings που να φανερώνουν, πιθανώς, κακόβουλη φύση, καθώς και εντολές που εκτελεί το πρόγραμμα που είναι εξ' ολοκλήρου ύποπτες.

Ταυτόχρονα, μας γίνονται φανερές οι αδυναμίες της ανεύρεσης κακοηθούς συμπεριφοράς παρατηρώντας μόνο τον κώδικα ενός εκτελέσιμου αρχείου, καθώς και τις αδυναμίες αντίστοιχα στατικών κανόνων YARA - οι οποίοι είναι επιρρεπείς στα false positives.

Παρ' όλα αυτά, η Στατική Ανάλυση είναι γρήγορη, και σίγουρα πιο αποτελεσματική στην άμεση και "συντηρητικής-φύσεως" λήψη αποφάσεων όταν πρόκειται για το ερώτημα της κακοβουλίας ενός προγράμματος. Αν υπάρχουν ξεκάθαρες ενδείξεις malware, όπως hardcoded strings, οι κανόνες YARA που έχουν δημιουργηθεί θα τις εντοπίσουν και θα το κάνουν flag. Η δυναμική παρακολούθηση της συμπεριφοράς ενός προγράμματος εξακολουθεί να είναι εξίσου σημαντική, όμως, και τελικά πιο κεντρική για τον καθορισμό κακόβουλων προγραμμάτων.

7 Βιβλιογραφία

- Google Cloud, ”WannaCry Malware Profile,” Google Cloud Blog, May 23, 2017. [Online]. Available: https://cloud.google.com/blog/topics/threat-intelligence/wannacry-malware-profile/. [Accessed: Jan. 15, 2025].
- Wikipedia, ”WannaCry ransomware attack,” Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/WannaCry_ransomware_attack. [Accessed: Jan. 15, 2025].
- Avast, ”What Is WannaCry?” Avast Blog. [Online]. Available: https://www.avast.com/c-wannacry. [Accessed: Jan. 15, 2025].
- ”Writing Rules,” YARA Documentation, [Online]. Available: https://yara.readthedocs.io/en/stable/writingrules.html. [Accessed: Jan. 15, 2025].
- W. Stallings, Cryptography and Network Security: Principles and Practice, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2003
- Y. Nativ and S. Shalev, theZoo: A Live Malware Repository, GitHub repository, [Online]. Available: https://github.com/ytisf/theZoo. [Accessed: Jan. 15, 2025].
- Y. Nativ and S. Shalev, ”theZoo - A Live Malware Repository,” theZoo, [Online]. Available: https://thezoo.morirt.com/. [Accessed: Jan. 15, 2025].