Hijacking Arbitrary .NET Application Control Flow

Topher Timzen
Topher Timzen
Security Researcher, Intel

Security Trainer

@TTimzen
TopherTimzen.com
Why are we Here?

Tools Released

Use .NET to attack

Using Objects on the Heap
CLR Attacks

Controlling the Common Language Runtime

Accessing raw objects on Managed Heap

Manipulate AppDomains

• Controlling all Loaded Code
• Controlling Just-In-Time Compilation
Attack With ASM

Manipulate Resources

Attack methods at ASM level

- Hybrid C# -> ASM -> C# call chains

Alter application control flow
Runtime

.NET Process
CLR (2.0/4.0) & AppDomains
Assemblies (.EXE and .DLL(s))

Objects
  Properties
  Fields
  Instance Methods

Classes
  Methods
  Logic
The Tools

Gray Frost

&

Gray Storm
Gray Frost
Gray Frost

Payload delivery system
C++ .NET CLR Bootstrapper
- Creates or injects 4.0 runtime
- Capability to pivot into 2.0 runtime
- Contains raw payload
Gray Frost

Payload delivery system
C++ .NET CLR Bootstrapper
- Creates or injects 4.0 runtime
- Capability to pivot into 2.0 runtime
- Contains raw payload

2 Rounds
- GrayFrostCpp
- GrayFrostCSharp
  - C# Payload
Round 1

.NET Process
Round 2

.NET Process
Round 2

.NET Process

GrayFrostCSharp
Round 2

.NET Process

payload void
main()
Round 2

.NET Process

Payload
Pivoting Between Runtimes

.NET Process
Pivoting Between Runtimes

GrayFrostCpp

Mscoree
Pivoting Between Runtimes

GrayFrostCpp
Pivoting Between Runtimes

GrayFrostCpp

GrayFrostCSharp
Pivoting Between Runtimes

GrayFrostCpp

GrayFrostCSharp
Pivoting Between Runtimes
Pivoting Between Runtimes

GrayFrostCpp

GrayFrostCSharp
Gray Storm
Gray Storm

Reconnaissance and In-memory attack payload

Features

- Attacking the .NET JIT
- Attacking .NET at the ASM level
- ASM and Metasploit payloads
- Utilize objects on the Managed Heap
Gray Storm Usage
Controlling the JIT

Method Tables contain address of JIT stub for a class’s methods.

During JIT the Method Table is referenced

We can control the address

Lives after Garbage Collection
Controlling the JIT
Controlling the JIT

Method A

Hacker Method

Hacker Method JIT Stub

Method B

Method B JIT Stub
Controlling the JIT

Method A

Hacker Method
Control Flow Attacks

.NET uses far and relative calls

- `0xE8; Call [imm]` (x86)
- `0xFF 0x15; Call dword segmentRegister[imm]` (x86)

```
relCall = dstAddress - (currentLocation + lenOfCall)
```
ASM Payloads

Address of a method known through Reflection

Overwrite method logic with new ASM

Steal stack parameters

Change events
ASM Payloads

Change return TRUE to return FALSE

- Password validation
- Key & Licensing validation
- SQL Sanitization

Destroy security Mechanisms

Overwrite logic

Update Mechanisms
public bool VerifyLicense(string licenseKey, string email) {
    if (string.IsNullOrEmpty(licenseKey) || string.IsNullOrEmpty(email)) {
        return false;
    }

    this.License = this.Decrypt(licenseKey);
    if (this.License != null && this.License.Email != null &&
        this.License.Product != null) {
        bool flag = this.License.Email.Equals(email, StringComparer.OrdinalIgnoreCase);
        bool flag2 = this.License.Product == "Licensed";
        return flag && flag2;
    }
    return false;
}

public bool VerifyLicense(string licenseKey, string email) {
    return true;
}
ASM Payloads

Metasploit

Hand Rolled

Portable Environment Block (PEB) changes
### Portable Environment Block

```
0:005> !peb
PEB at 7efde000
    InheritedAddressSpace:      No
    ReadImageFileExecOptions:   No
    BeingDebugged:              Yes
    ImageBaseAddress:           012b0000
    Ldr                      77b40200
    Ldr.Initialized:            Yes
    Ldr.InInitializationOrderModuleList: 00273a80 00301d48
    Ldr.InLoadOrderModuleList:   002739e0 00301d38
    Ldr.InMemoryOrderModuleList: 002739e8 00301d40

<table>
<thead>
<tr>
<th>Base TimeStamp</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>12b0000 54f4a118 Mar 02 09:42:48 2015</td>
<td>C:\Users\Blob\DllInjector.exe</td>
</tr>
<tr>
<td>77a40000 521ea8e7 Aug 28 18:50:31 2013</td>
<td>C:\Windows\SysWOW64\ntdll.dll</td>
</tr>
<tr>
<td>73a10000 4b90752b Mar 04 19:06:19 2010</td>
<td>C:\Windows\SYSTEM32\MSCOREE.DLL</td>
</tr>
<tr>
<td>75fc0000 53159a85 Mar 04 01:19:01 2014</td>
<td>C:\Windows\syswow64\KERNEL32.dll</td>
</tr>
</tbody>
</table>
```
Object Hunting in Memory
Managed Heap

Storage point for .NET Objects

New reference objects added to heap

Garbage Collector removes dead objects
Managed Heap

Storage point for .NET Objects

New reference objects added to heap

Garbage Collector removes dead objects

Let’s manipulate it!
Object Hunting in Memory

Objects are IntPtrs
Point to Object Instance on Managed Heap
All instantiated objects of the same class share the same Method Table
Object Hunting in Memory

Objects are IntPtrs
Point to Object Instance on Managed Heap
All instantiated objects of the same class share the same Method Table

Reflection + Object Hunting = Win
Finding Objects at Runtime

i. Construct an object and find location of Managed Heap

ii. Signature instantiated type

iii. Scan Managed Heap for object pointers

iv. Convert object pointers to raw objects
Finding Objects at Runtime

i. Construct an object and find location of Managed Heap

ii. Signature instantiated type

iii. Scan Managed Heap for object pointers

iv. Convert object pointers to raw objects
Construct an Object

Use Reflection to invoke a constructor

Can instantiate any object

• Being mindful that some constructors can invoke a lot of code

If a constructor takes other objects, nullify them

https://gist.github.com/tophertimzen/010b19fdebde77f251414
Find Location of Managed Heap

<table>
<thead>
<tr>
<th>STACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>024e9fe8 (Object)</td>
</tr>
<tr>
<td>00000005</td>
</tr>
<tr>
<td>00000001</td>
</tr>
<tr>
<td>00000000</td>
</tr>
<tr>
<td>IntPtr = 5</td>
</tr>
</tbody>
</table>

https://gist.github.com/tophertimzen/812aa20dbe23cb42756d
Find Location of Managed Heap

STACK

024e9fe8 (Object)
00000005
00000001
00000000
IntPtr = 5

Managed Heap

https://gist.github.com/tophertimzen/812aa20dbe23cb42756d
Find Location of Managed Heap

<table>
<thead>
<tr>
<th>STACK</th>
<th>024e9fe8 (Object)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00000005</td>
</tr>
<tr>
<td></td>
<td>00000001</td>
</tr>
<tr>
<td></td>
<td>00000000</td>
</tr>
<tr>
<td></td>
<td>IntPtr = 5</td>
</tr>
</tbody>
</table>

https://gist.github.com/tophertimzen/812aa20dbe23cb42756d
Find Location of Managed Heap

STACK

<table>
<thead>
<tr>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>024e9fe8 (Object)</td>
</tr>
<tr>
<td>00000005</td>
</tr>
<tr>
<td>00000001</td>
</tr>
<tr>
<td>00000000</td>
</tr>
<tr>
<td>IntPtr = 024e9fe8</td>
</tr>
</tbody>
</table>

https://gist.github.com/tophertimzen/812aa20dbe23cb42756d
Finding Objects at Runtime

i. Construct an object and find location of Managed Heap

ii. Signature instantiated type

iii. Scan Managed Heap for object pointers

iv. Convert object pointers to raw objects
Signature Instantiated Type

Object Instances contain a Method Table pointer to their corresponding type.

(x86)
Bytes 0-3 are the Method Table (MT)
Bytes 4-7 in MT is Instance Size
Signature Instantiated Type

Object Instances contain a Method Table pointer to their corresponding type.

(x64)

Bytes 0-7 are the Method Table (MT) Bytes 8-11 in MT is Instance Size
Finding Objects at Runtime

i. Construct an object and find location of Managed Heap

ii. Signature instantiated type

iii. Scan Managed Heap for object pointers

iv. Convert object pointers to raw objects
Scan Managed Heap

Scan down incrementing by size of object

Scan linearly up to top of heap

Compare object’s Method Table to the reference

If they match, get IntPtr address of object
Scan Managed Heap

Scan down incrementing by size of object

Scan linearly up to top of heap

Compare object’s Method Table to the reference

If they match, get IntPtr address of object

Use ASM!
Using ASM!

```csharp
public delegate IntPtr mtDEL(IntPtr objectIN);

static public byte[] getMethodTablex86 = new byte[]
{
    0x8b, 0x44, 0x24, 0x04, //mov eax, [esp+4] (arg1 on the stack = instanceObject)
    0x8b, 0x00, //mov eax, [eax] (first 4 bytes are the MethodTable)
    0xc3 //ret (return methodTable for comparison)
};

getMethodTable = getObjectMethodTable(obj, getMethodTablex86);

public static IntPtr getObjectMethodTable(IntPtr objectIN, byte[] methodFinder)
{
    IntPtr ptr = assemblyHelpers.VirtualAlloc(methodFinder);
    mtDEL getMT = (mtDEL)GetDelegateForFunctionPointer(ptr, typeof(mtDEL));

    IntPtr methodTable = getMT(objectIN);
    assemblyHelpers.VirtualFree(p, 0, 0x8000);
    return methodTable;
}
```
Finding Objects at Runtime

i. Construct an object and find location of Managed Heap
ii. Signature instantiated type
iii. Scan Managed Heap for object pointers
iv. Convert object pointers to raw objects
Convert Object ptr -> raw obj

```csharp
public static object GetInstance(IntPtr ptrIN)
{
    object refer = ptrIN.GetType();
    IntPtr pointer = ptrIN;
    unsafe
    {
        *(&pointer - 1) = *(&pointer);
    }
    return refer;
}
```

STACK

- Refer (System(IntPtr)
- pointer(024ea00c)

https://gist.github.com/tophertimzen/1da2b0aab6245ed1c27b
Convert Object ptr -> raw obj

```csharp
public static object GetInstance(IntPtr ptrIN)
{
    object refer = ptrIN.GetType();
    IntPtr pointer = ptrIN;
    unsafe
    {
        *((&pointer - 1)) = *((&pointer));
    }
    return refer;
}
```

STACK

```
L
pointer(024ea00c)
H
pointer(024ea00c)
```

https://gist.github.com/tophertimzen/1da2b0aab6245ed1c27b
Convert Object ptr -> raw obj

```csharp
public static object GetInstance(IntPtr ptrIN)
{
    object refer = ptrIN.GetType();
    IntPtr pointer = ptrIN;
    unsafe
    {
        *(&pointer - 1) = *(&pointer);
    }
    return refer;
}
```

STACK

```
L
H
```

- Refer (GrayStorm.testClass)
- pointer(024ea00c)

https://gist.github.com/tophertimzen/1da2b0aab6245ed1c27b
Superpowers and Things?

- Change Keys
- Change Fields / Properties
- Call Methods
  - With arguments!
Automation
GrayFrost can be used with automated payloads

```
GrayKernel>autoFrost.py
Usage: autoFrost.py <C# Payload.exe> <GrayFrost.sln file>

GrayKernel>autoFrost.py autoThink.exe grayfrost\\GrayFrost.sln
[+] Building Payload into embeddable array
[+] Writing GrayFrostCSharp\\payload.cs
[+] Building GrayFrostCSharp
[+] Writing Slate.h
[+] Building GrayFrost{32,64}.dll
[+] GrayFrost finished building
```
Automations == <3
Constructing Attack Chains
How to construct attack chains

Gray Wolf / IL Decompiler
- Find Methods, Fields & Properties of interest
- Locate meaningful objects
- Discover high level control flow

Gray Storm “Debugging” functionality
- Breakpoint at constructors or methods from Method Pointers
- Use with WinDbg

Utilize DLL Hijacking!
Hybrid .NET/ASM Attacks

- Hybrid C#/ASM code in .NET
- Encrypting .NET payloads and unwinding
- Encrypting ASM Payloads
Payload System

C# is easy

Can use Gray Frost in any application

Low and High level gap is easy
.NET Hacking Space

Small

Few tools
  Mostly hacking WoW
  Lots of PowerShell

Lots of decompilers and not many attackers!
Conclusion

- Arbitrary .NET applications can be injected and changed
- New .NET attack possibilities
- New tools that support automation
- Get Gray Frost and Storm

[github.com/graykernel]
Questions?

Contact Me

- @TTimzen
- https://www.tophertimzen.com

Get Gray Frost and Storm

- github.com/graykernel

White Papers

- Hijacking Arbitrary .NET Application Control Flow
- Acquiring .NET Objects from the Managed Heap