Unveiling the underground world of ANTI-CHEATS

Joel Noguera
Security Consultant at Immunity Inc
@niemand_sec
What are we going to talk about?
FIRST RULE OF THE GAMING CLUB, YOU DON'T CHEAT
(or get caught doing it)
<table>
<thead>
<tr>
<th>Name</th>
<th>Date Modified</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>word.bak</td>
<td>10/5/2018 2:53 PM</td>
<td>BAK File</td>
</tr>
<tr>
<td>word.exe</td>
<td>10/17/2018 6:38 PM</td>
<td>Application</td>
</tr>
<tr>
<td>word.exe.log</td>
<td>10/5/2018 2:54 PM</td>
<td>Text Document</td>
</tr>
</tbody>
</table>
Anti Cheats
Let’s see some numbers...

336,500,000

Monthly Active Users

EAC: 275,000,000
XC3: 500,000
BE: 30,000,000
VAC: 31,000,000
Anti-Cheat Components

- **Anti-Cheat Server**
- **Ring 0 Process (sys)**
  - Anti-Cheat Driver
- **Game (exe)**
  - Main DLL
- **Memory Sections**
  - Stripped DLL
- **Ring 3 Process (exe)**
  - Anti-Cheat Watchdog Process
  - Anti-Cheat External Process
Kernel Driver

- Handle stripping/Access Control
- Register kernel callbacks
- Rejection of Kernel/User mode debugging
- Analysis of privileged process (lsass and csrss)
- Block blacklisted/unsigned drivers
- Monitoring of kernel function calls
DLL inside Games

- Control of access flags to different sections
- Identification of hooks
- Thread Hijacking
- DLL Injection
- Function signatures
- VEH/SEH modification
- Game resources modification
- Detection of virtual environment
External Ring 3 Process

- Process/File Controls
- Blacklisted programs detection
- Manage logic from Driver
- Control of game client and DLL hashes
- Multi-client detection
- Program integrity controls
Cheats
<table>
<thead>
<tr>
<th></th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>External</td>
<td>Quick for small patches</td>
<td>Slow</td>
</tr>
<tr>
<td></td>
<td>Easy to master</td>
<td>Easy to detect</td>
</tr>
<tr>
<td></td>
<td>Can be closed in certain cases</td>
<td>Limited potential</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Requires a Handle (usually)</td>
</tr>
<tr>
<td>Internal</td>
<td>Great performance</td>
<td>Hard to master</td>
</tr>
<tr>
<td></td>
<td>Direct access to memory</td>
<td>Easier to detect if you mess it up</td>
</tr>
<tr>
<td></td>
<td>Hard to detect if you are good enough</td>
<td></td>
</tr>
</tbody>
</table>
Aimbots

Wallhack/ESP

Aimbots
Pro players getting caught? Why not
Parallel Market
Parallel Market

Cheat Prices:
U$S 1 to U$S25
Some up to U$S500

Ex: 2500 paid members
U$S 10 * 2500 = U$S25000
(150000 memberships)

U$S 1,250 M
PER YEAR
(Wait... what?)
Are they fighting back?

Apex claims:

- More than 770k players banned
- Over 300K account creations blocked
- Over than 4k cheat sellers accounts (spammers) banned in 20 days

https://unknowncheats.me/
Analyzing Anti-Cheats
Methodology

Goal:

- Read/Write/Alloc Memory (Internal & External)
- Run Code inside Game’s Process
- Be as stealthy as possible
AC usually control/block/reject new HANDLEs to the game process:

- Driver that protects game and AC processes

Some process need to be whitelisted: `lsass, csrss, AC`

Hijacking techniques come to our rescue:

- Handle Hijacking
- Stealth Handle Hijacking
- Hooking
Hijacking Techniques
Hijacking Techniques - NamedPipe

LSASS (exe)
- Attacker DLL
- Handle 0x4A
- Namedpipe

GAME (exe)

CHEATS (exe)
- Namedpipe

“\Device\NamedPiped\270F59B0075AA3D3”
Hijacking Techniques - NamedPipe

Disadvantages

- Suspicious new HANDLEs
- Hooks to user-mode WIN API
- Thread with suspicious context
- Downgrade of HANDLE privileges
Imagine a world where our shared memory does not leave an open HANDLE and we can cover better our tracks.
Hijacking Techniques - FileMapping

“File mapping object does not close until all references to it are released”

We can call CloseHandle without calling to UnmapViewOfFile.
Hijacking Techniques - File Mapping

We can make it even better by delaying the execution.

Manual spinlocks to avoid mutex/semaphores HANDLEs.

Request / Response Structure

Shared Memory

Spinlock
Hijacking Techniques - FileMapping

Disadvantages

- Suspicious new HANDLEs
- Hooks to user-mode WIN API
- Thread with suspicious context
- Downgrade of HANDLE privileges
Hijacking Techniques - Bypass Hooks

EAC also hook functions on `lsass.exe`:

<table>
<thead>
<tr>
<th>C:\WINDOWS\system32\lsass.exe[928] @ C:\WINDOWS\System32\KERNEL32.DLL</th>
<th>[ntdll.dll]!AllocateVirtualMemory</th>
</tr>
</thead>
<tbody>
<tr>
<td>C:\WINDOWS\system32\lsass.exe[928] @ C:\WINDOWS\System32\KERNEL32.DLL</td>
<td>[ntdll.dll]!ReadVirtualMemory</td>
</tr>
<tr>
<td>C:\WINDOWS\system32\lsass.exe[928] @ C:\WINDOWS\System32\KERNEL32.DLL</td>
<td>[ntdll.dll]!NtWriteVirtualMemory</td>
</tr>
<tr>
<td>C:\WINDOWS\system32\lsass.exe[928] @ C:\WINDOWS\System32\KERNELBASE.dll</td>
<td>[ntdll.dll]!NtAllocateVirtualMemory</td>
</tr>
<tr>
<td>C:\WINDOWS\system32\lsass.exe[928] @ C:\WINDOWS\System32\KERNELBASE.dll</td>
<td>[ntdll.dll]!NtReadVirtualMemory</td>
</tr>
<tr>
<td>C:\WINDOWS\system32\lsass.exe[928] @ C:\WINDOWS\System32\KERNELBASE.dll</td>
<td>[ntdll.dll]!NtWriteVirtualMemory</td>
</tr>
<tr>
<td>C:\WINDOWS\system32\lsass.exe[928] @ C:\WINDOWS\System32\KERNELBASE.dll</td>
<td>[ntdll.dll]!NtAllocateVirtualMemory</td>
</tr>
<tr>
<td>C:\WINDOWS\system32\lsass.exe[928] @ C:\WINDOWS\System32\KERNELBASE.dll</td>
<td>[ntdll.dll]!NtReadVirtualMemory</td>
</tr>
<tr>
<td>C:\WINDOWS\system32\lsass.exe[928] @ C:\WINDOWS\System32\lsasrv.dll</td>
<td>[ntdll.dll]!NtAllocateVirtualMemory</td>
</tr>
<tr>
<td>C:\WINDOWS\system32\lsass.exe[928] @ C:\WINDOWS\System32\lsasrv.dll</td>
<td>[ntdll.dll]!NtReadVirtualMemory</td>
</tr>
<tr>
<td>C:\WINDOWS\system32\lsass.exe[928] @ C:\WINDOWS\System32\lsasrv.dll</td>
<td>[ntdll.dll]!NtWriteVirtualMemory</td>
</tr>
</tbody>
</table>

Why?

- Validate/Control/Track each action done against the game
Hijacking Techniques - Bypass Hooks

```
.. code::

    ZwWriteVM proc
    mov r10, rcx
    mov eax, 3Ah
    syscall
    ret
ZwWriteVM endp

ZwReadVM proc
    mov r10, rcx
    mov eax, 3Fh
    syscall
    ret
ZwReadVM endp
```

```
loc_1800080E5: ; DOS 2+ internal - EXECUTE COMMAND
    int  2eh ; DS:SI -> counted CR-terminated command string
    ret
    NtWriteVirtualMemory endp
```
Hijacking Techniques - Bypass Hooks

Disadvantages

- Suspicious new HANDLEs
- Hooks to user-mode WIN API
- Thread with suspicious context
- Downgrade of HANDLE privileges
Hooking
Hooking Graphic Engines:

- IAT hooking
- JMPs on Prolog functions

What about 3rd party libraries?

- Steam Overlay
- Open Broadcaster Software
Steam Overlay

Redirects execution to gameoverlayrenderer64.dll:$8A480

Open Broadcaster Software

Redirects to graphics-hook64.7FFEB97AE4D0
Hooking - Code Caves and NamedPipes?
Refresher- Bypass Hooks

Disadvantages

- Suspicious new HANDLEs
- Hooks to user-mode WIN API
- Thread with suspicious context
- Downgrade of HANDLE privileges
Moving to kernel...Drivers
Drivers

Cheat developers also develop their own to fight inside the kernel.

Loading a Driver:

- Test Mode
- Sign your own Driver ($$$$$$$)
- Abuse of another driver

GIGABYTE Driver

- CVE-2018-19320 (ring0 memcpy with VA)
- CVE-2018-19321 (read/write arbitrary physical memory)
EAC downgrading the HANDLE

<table>
<thead>
<tr>
<th>Type</th>
<th>Handle</th>
<th>Name</th>
<th>Access</th>
<th>Decoded Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process</td>
<td>0x9A8</td>
<td>ServiceHub_DataWarehouseHost.exe(10652)</td>
<td>0x00000048 VM_OPERATION</td>
<td>VM_OPERATION</td>
</tr>
<tr>
<td>Process</td>
<td>0x146C</td>
<td>sedsvc.exe(7312)</td>
<td>0x00000048 VM_OPERATION</td>
<td>VM_OPERATION</td>
</tr>
<tr>
<td>Process</td>
<td>0x14B8</td>
<td>SearchUI.exe(10180)</td>
<td>0x00000048 VM_OPERATION</td>
<td>VM_OPERATION</td>
</tr>
<tr>
<td>Process</td>
<td>0xE10</td>
<td>SearchIndexer.exe(7103)</td>
<td>0x00000048 VM_OPERATION</td>
<td>VM_OPERATION</td>
</tr>
<tr>
<td>Process</td>
<td>0x185C</td>
<td>ScriptedSandBox64.exe(15372)</td>
<td>0x00000048 VM_OPERATION</td>
<td>VM_OPERATION</td>
</tr>
<tr>
<td>Process</td>
<td>0x1840</td>
<td>SCM.exe(6204)</td>
<td>0x00000048 VM_OPERATION</td>
<td>VM_OPERATION</td>
</tr>
<tr>
<td>Process</td>
<td>0x186C</td>
<td>RuntimeBroker.exe(7604)</td>
<td>0x00000048 VM_OPERATION</td>
<td>VM_OPERATION</td>
</tr>
<tr>
<td>Process</td>
<td>0x15A0</td>
<td>RuntimeBroker.exe(12244)</td>
<td>0x00000048 VM_OPERATION</td>
<td>VM_OPERATION</td>
</tr>
<tr>
<td>Process</td>
<td>0x14FC</td>
<td>RuntimeBroker.exe(10640)</td>
<td>0x00000048 VM_OPERATION</td>
<td>VM_OPERATION</td>
</tr>
<tr>
<td>Process</td>
<td>0x1854</td>
<td>rSesx.exe(6048)</td>
<td>0x00000142 VM_OPERATION</td>
<td>VM_OPERATION</td>
</tr>
<tr>
<td>Process</td>
<td>0x1910</td>
<td>QHSafe Tray.exe(14228)</td>
<td>0x00000048 VM_OPERATION</td>
<td>VM_OPERATION</td>
</tr>
<tr>
<td>Process</td>
<td>0x026C</td>
<td>QuickDefence.exe(3495)</td>
<td>0x00000048 VM_OPERATION</td>
<td>VM_OPERATION</td>
</tr>
<tr>
<td>Process</td>
<td>0x1C34</td>
<td>process.exe(4323)</td>
<td>0x00000048 VM_OPERATION</td>
<td>VM_OPERATION</td>
</tr>
<tr>
<td>Process</td>
<td>0xAD0</td>
<td>PerfWatson2.exe(3880)</td>
<td>0x00000048 VM_OPERATION</td>
<td>VM_OPERATION</td>
</tr>
<tr>
<td>Process</td>
<td>0x1804</td>
<td>PerfWatson2.exe(12083)</td>
<td>0x00000048 VM_OPERATION</td>
<td>VM_OPERATION</td>
</tr>
</tbody>
</table>
Driver - DKOM

1) Search for EPROCESS Struct in kernel

```c
typedef struct {
    CHAR ImageFileName[15];  // Image file name
    DWORD PriorityClass;     // Priority class
} EPROCESS;
```

2) Obtain the ObjectTable (HANDLE_TABLE)

3) Use ExpLookupHandleTableEntry(HandleTable, Handle)

4) Retrieve HANDLE

5) Modify GrantedAccess

6) Overwrite kernel memory

7) Profit
Refresher - Bypass Hooks

Disadvantages

- Suspicious new HANDLEs
- Hooks to user-mode WIN API
- Thread with suspicious context
- Downgrade of HANDLE privileges
One Last Attempt
Driver - Just do it from kernel!

1) Leak handle pointers using NtQuerySystemInformation
   SystemExtendedHandleInformation (0x40) as SYSTEM_INFORMATION_CLASS

2) Locate valid KPROCESS pointer
   _KPROCESS.Header == 0x00B60003

3) Traverse linked list -> _EPROCESS.ActiveProcessLinks

4) Obtain DirectoryBaseTable -> _EPROCESS.PEB.DirectoryBaseTable

5) Obtain target Base Address -> _EPROCESS.SectionBaseAddress

6) Dereference Ring3 virtual addresses

7) Directly modify/read memory
What about the tools?
<table>
<thead>
<tr>
<th>Module</th>
<th>Address</th>
<th>Size</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntdll.dll</td>
<td>00007FCC850000</td>
<td>00000000001D1000</td>
<td>C:\Windows\System32\ntdll.dll</td>
</tr>
<tr>
<td>KERNEL32.DLL</td>
<td>00007FCC860000</td>
<td>00000000000FAB000</td>
<td>C:\Windows\System32\KERNEL32.DLL</td>
</tr>
<tr>
<td>KERNELBASE.dl</td>
<td>00007FCC83E0000</td>
<td>000000000021D000</td>
<td>C:\Windows\System32\KERNELBASE.dll</td>
</tr>
<tr>
<td>apohelp.dll</td>
<td>00007FCC7B40000</td>
<td>00000000007A0000</td>
<td>C:\Windows\System32\apohelp.dll</td>
</tr>
<tr>
<td>combase.dll</td>
<td>00007FCC8330000</td>
<td>00000000002C7000</td>
<td>C:\Windows\System32\combase.dll</td>
</tr>
<tr>
<td>ucrtbase.dll</td>
<td>00007FCC8F80000</td>
<td>0000000000F50000</td>
<td>C:\Windows\System32\ucrtbase.dll</td>
</tr>
<tr>
<td>RpcRT4.dll</td>
<td>00007FCC83E0000</td>
<td>0000000000121000</td>
<td>C:\Windows\System32\RpcRT4.dll</td>
</tr>
<tr>
<td>bcryptPrimitives.dll</td>
<td>00007FCC8360000</td>
<td>00000000006A0000</td>
<td>C:\Windows\System32\bcryptPrimitives.dll</td>
</tr>
<tr>
<td>vccorlib140_app.DLL</td>
<td>00007FCC96D2000</td>
<td>0000000000580000</td>
<td>C:\Program Files\WindowsApps\Microsoft.VCLibs.140.00_14.0.27323.0_x64__6we...</td>
</tr>
</tbody>
</table>
Black Hat Sound Bytes

- Fight at kernel level vs Trivial Bypasses
- Blacklisting all drivers is impossible
- Compatibility with Windows and 3rd applications is a problem
Open Source Projects

ReClass Plugin - Driver Reader

niemand-sec/ReClass.NET-DriverReader

AntiCheat-Testing-Framework

niemand-sec/AntiCheat-Testing-Framework

- CheatHelper & DriverHelper
- DriverDisabler & Synapse Driver exploit (Razer)
- HandleHijackingDLL and HandleHijackingMaster
- NamePipes and FileMapping
- WinApi Hooking Bypass & Lua Hooking
- Handle Elevation and External Driver
THANK YOU!

@niemand_sec

niemand-sec/AntiCheat-Testing-Framework

niemand-sec/ReClass.NET-DriverReader