# Avoiding Memory Scanners

## Customizing Malware to Evade YARA, PE-sieve, and More

Kyle Avery

# Introduction

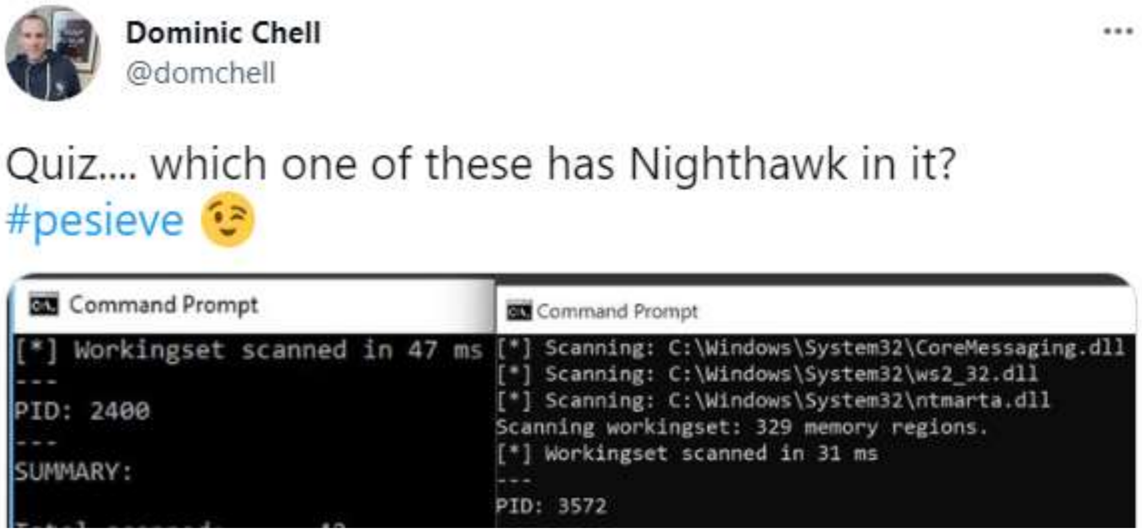
- Offensive Security Lead at H-E-B
- Former BHIS
- Focus on Post-Exploitation

- Twitter: @kyleavery_
- GitHub: kyleavery

# Introduction

- Goals for the talk:
  - Describe memory scanner capabilities
  - Demonstrate bypasses for each
  - Eliminate misconceptions
- Why Cobalt Strike?
  - Common denominator for many red teams
  - Highly targeted by scanning tools

https://github.com/kyleavery/AceLdr

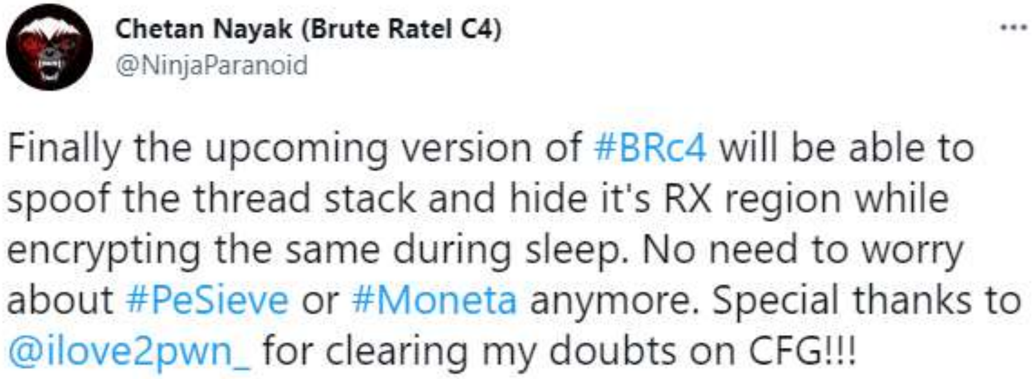


Dominic Chell
@domchell

Quiz.... which one of these has Nighthawk in it? #pesieve 😉

```
[*] Workingset scanned in 47 ms
---
PID: 2400
---
SUMMARY:
```

```
[*] Scanning: C:\Windows\System32\CoreMessaging.dll
[*] Scanning: C:\Windows\System32\ws2_32.dll
[*] Scanning: C:\Windows\System32\ntmarta.dll
Scanning workingset: 329 memory regions.
[*] Workingset scanned in 31 ms
---
PID: 3572
```



Chetan Nayak (Brute Ratel C4)
@NinjaParanoid

Finally the upcoming version of #BRc4 will be able to spoof the thread stack and hide it's RX region while encrypting the same during sleep. No need to worry about #PeSieve or #Moneta anymore. Special thanks to @ilove2pwn_ for clearing my doubts on CFG!!!

# Agenda

- Memory Scanner Background – Notable Capabilities
  - Pattern Matching
  - Memory Attributes
  - Stack Tracing
- Memory Evasion – Bypassing Scanners
  - Encrypting Data
  - Heap Encryption
  - Obfuscating Executable Code
  - Avoiding Sleep
  - Return Address Spoofing
- Introducing AceLdr – Ready-to-Use Evasion Tool

# Memory Scanners – Pattern Matching

- YARA from VirusTotal
  - Text and binary pattern matching with conditional logic
- BeaconEye by @_EthicalChaos_
  - Pattern matching, specifically looking for the Cobalt Strike configuration
- Commercial Security Products
  - YARA Connector and Manager User Guide for EDR - Carbon Black Developer Network
  - Falcon X: Cyber Threat Intelligence & Automation | CrowdStrike

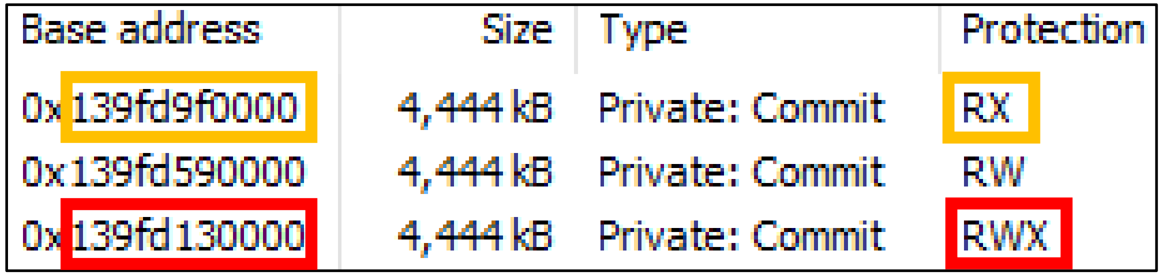
```
rule ExampleYARA
{
  strings:
    $a = {41 42 ?? 44}
    $b = "String"
condition:
    $a or $b
}
```

# Memory Scanners – Memory IOCs

- Moneta by Forrest Orr
  - Private commit memory with executable permissions
  - PEB image bases or threads with start addresses in private commit memory
- PE-sieve by @hasherezade
  - Scans non-executable, inaccessible, and private commit memory for patterns
  - Check return address of all threads for private commit memory addresses
- malfind from Volatility
  - Identifies private commit RWX memory in forensic memory dumps

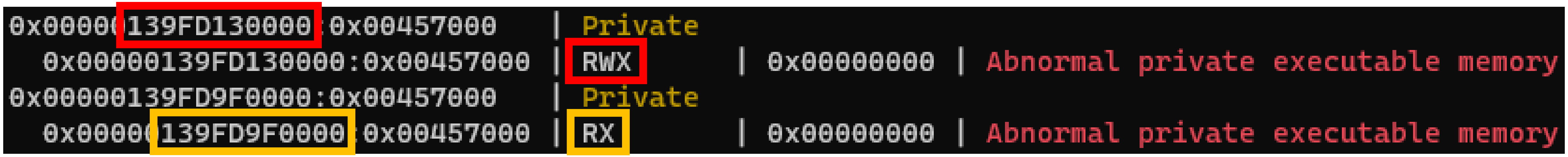
# Memory IOCs – Moneta

```
VirtualAlloc( 0, 1024*4444, MEM_COMMIT, PAGE_READWRITE );
VirtualAlloc( 0, 1024*4444, MEM_COMMIT, PAGE_EXECUTE_READ );
VirtualAlloc( 0, 1024*4444, MEM_COMMIT, PAGE_EXECUTE_READWRITE );
```



| Base address | Size | Type | Protection |
|---|---|---|---|
| 0x139fd9f0000 | 4,444 kB | Private: Commit | RX |
| 0x139fd590000 | 4,444 kB | Private: Commit | RW |
| 0x139fd130000 | 4,444 kB | Private: Commit | RWX |

```
C:\> .\Moneta.exe -p 1234 -m ioc
```



```
0x00000139FD130000:0x00457000    | Private
  0x00000139FD130000:0x00457000  | RWX        | 0x00000000 | Abnormal private executable memory
0x00000139FD9F0000:0x00457000    | Private
  0x00000139FD9F0000:0x00457000  | RX         | 0x00000000 | Abnormal private executable memory
```

# Memory Scanners – Stack Tracing

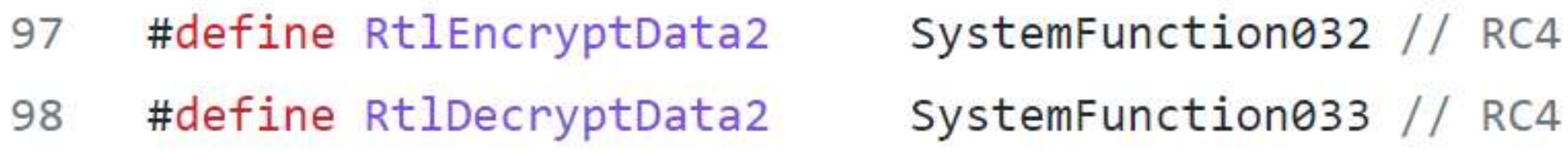
- BeaconHunter by Andrew Oliveau
    - Uses .NET System.Diagnostics to check ThreadWaitReason for ExecutionDelay
- Hunt-Sleeping-Beacons by @thefLinkk
    - Check stack of all threads for NtDelayExecution
- MalMemDetect by Waldo-irc
    - Hooks InternetConnectA/W, NtWaitForSingleObject, and RtlAllocateHeap to check the return address at execution-time
    - Any call to these APIs, or an API that calls them, from private commit memory will be flagged

# Bypassing Memory Scanners

- Each scanner can be bypassed with combinations of encryption and spoofing
- A true bypass does not leave any false positives or detections from the target scanner
- Memory scanners and commercial security products only look for ***some*** of the same IOCs

# Encrypting Data

- Issues with XOR
  - Tools like YARA and PE-sieve can break weak XOR encryption
- Issues with Including an Encryption Routine
  - Code required for encryption/decryption must be executable
- Solution: SystemFunction032
  - In Mimikatz, @gentilkiwi provides an example of using functions from advapi.dll to perform RC4 encryption/decryption



```
#define RtlEncryptData2         SystemFunction032 // RC4
#define RtlDecryptData2         SystemFunction033 // RC4
```

# Heap Encryption

- LockdExeDemo by Waldo-irc
  - PoC that attempts to encrypt all heap entries which aren’t busy
- Secondary Heap
  - PoC to create a new heap at initialization and hook GetProcessHeap ensure all of Beacon’s memory is separate from the host process
- Sleep Mask from Cobalt Strike
  - Cobalt Strike will provide a list of heap memory addresses to the sleep mask stub for encryption

# Obfuscating Executable Code

- Consistently evading tools like Moneta and PE-sieve requires a combination of encryption and non-executable private commit memory
- Most C2 implants sleep for long periods between callbacks
- The code required to change permissions of the target memory must be executable

# Hiding Implants with a Masking Stub

- Sleep Mask from Cobalt Strike
  - A built-in capability of Cobalt Strike to encrypt the implant memory at rest by hooking Sleep()
  - Requires an executable stub to handle the errors – resulting in fewer IOCs from PE-sieve and Moneta
- Shellcode Fluctuation by @mariuszbit/mgeeky
  - PoC usage of VEH to catch errors generated by attempting to execute code from non-executable or inaccessible regions
  - Requires an executable stub to handle the errors – resulting in fewer IOCs from PE-sieve and Moneta

# Hiding Implants with Traditional ROP

- Gargoyle by Josh Lospinoso
  - PoC usage of ROP and APCs to queue a sleep routine without the need for an executable stub
  - ROP gadgets are chosen from image commit memory to avoid executable private commit memory
- YouMayPasser by Waldo-irc
  - Ready-to-use x64 implementation of Gargoyle that uses VEH and hardware breakpoints to hook API calls
- DeepSleep by @thefLinkk
  - PoC “APC-less” Gargoyle implementation

# Hiding Implants with NtContinue

- FOLIAGE by Austin Hudson
  - PoC that creates a new thread and queues a series of user mode APCs
  - A context structure is created for each APC and passed to NtContinue
  - NtTestAlert is put on the top of the context stack which forces the thread back into an alertable state after each step in the chain

# FOLIAGE – Sleep Chain Setup

1. Opens a handle to KsecDD driver for encryption
2. Opens a handle to the current thread to modify the current context
3. Creates a new thread for the APC queue
4. Creates a new event to keep the new thread from exiting
5. Copies the context of the new thread to a new context structure

Full credit to Austin Hudson for this technique

# FOLIAGE – Sleep Chain Queue

6. Queues a series of NtContinue APC calls, each with a context defining a step in the sleep chain, that will return to NtTestAlert
   1. Waits on event to keep thread from exiting
   2. Changes the target memory permissions to RW
   3. Instructs the KsecDD driver to encrypt the memory
   4. Saves the context of the original thread
   5. Sets the context of the original thread to a fake context

Full credit to Austin Hudson for this technique

# FOLIAGE – Sleep Chain Queue

6. Queues a series of NtContinue APC calls, each with a context defining a step in the sleep chain, that will return to NtTestAlert
    6. Sleeps for the specified time with NtDelayExecution
    7. Instructs the KsecDD driver to decrypt the memory
    8. Restores the original thread context
    9. Changes the target memory permissions to RWX
    10. Exits the new thread

Full credit to Austin Hudson for this technique

# FOLIAGE – Sleep Chain Initialization

7. Forces the new thread into an alertable state
8. Signals the event and waits on the thread to prevent original thread from continuing

Full credit to Austin Hudson for this technique

# Hiding Implants with NtContinue Pt. 2

- Ekko by @C5pider / NightHawk from MDSec
  - PoC that queues a series of timers with callbacks to NtContinue

```
CreateTimerQueueTimer(
    &Timer,
    queue,
    NtContinue,                     // Callback to NtContinue
    &CtxVp,                         // Parameter for callback
    100,                            // Delay used to offset actions
    0,
    WT_EXECUTEINTIMERTHREAD         // Queues the callback as an APC
);
```

# Avoiding Sleep

- Tools like BeaconHunter and Hunt-Sleeping-Beacons flag any program waiting on NtDelayExecution
- This can be avoided with a variety of alternatives
  - Waitable Timers – Requires multiple API calls
  - WaitForSingleObject – Using the timeout parameter to delay execution

```
Timer = CreateWaitableTimer(NULL, TRUE, NULL);
SetWaitableTimer(Timer, &Delay, 0, NULL, NULL, FALSE);
WaitForSingleObject(Timer, INFINITE);
```

# Return Address Spoofing at Rest

- Thread Stack Spoofing by @mariuszbit/mgeeky
  - Overwrites the return address with zero, truncating the stack
  - May leak arguments onto the stack, leading to a (technically false) IOC
- Stack frame modification with NtSetContextThread
  - Create or copy a CONTEXT structure, including the instruction pointer, to use during sleep

# Return Address Spoofing at Exec

- CallStackSpoofer by Will Burgess
  - PoC that creates an invalid but unwinding stack and then catches errors with an exception handler
- x64 Return Address Spoofing by @namazso
  - A PoC that temporarily stores a ROP gadget as the return address while executing the intended function, then restores the registers necessary to return to the original caller

# Return Address Spoofing – Wrapper

1. The calling program sets up a custom struct with the address of a JMP RBX gadget, the address of a target function, and a space to store data

```
typedef struct {
  const void* trampoline;  // JMP RBX Gadget
  void* function;          // Target Function
  void* rbx;               // Empty
} PRM, *PPRM;
```

Full credit to @namazso for this technique

# Return Address Spoofing – Wrapper

2. The calling program calls the shellcode as a function, passing in any arguments

```
param.trampoline = findGadget("\xFF\x23");    // JMP RBX
param.function = Sleep;                       // KERNEL32!Sleep

//         Arg1   Arg2   Arg3   Arg4
((PVOID(*)(DWORD, PVOID, PVOID, PVOID, PPRM, PVOID)) Spoof(
  5000, NULL, NULL, NULL, &param, NULL
);
```

Full credit to @namazso for this technique

# Return Address Spoofing – Wrapper

3. Functions with additional arguments (more than four) are also possible

```
// WriteProcessMemory(hProc, lpBase, lpBuf, nSize, *lpBytes);
param.function = WriteProcessMemory;      // KERNEL32!WriteProcessMemory

//         Arg1   Arg2   Arg3   Arg4                Arg5...
((PVOID(*)(PVOID, PVOID, PVOID, PVOID, PPRM, PVOID, PVOID)) Spoof(
  hProc, lpBase, lpBuf, nSize, &param, NULL, lpBytes
);
```

Full credit to @namazso for this technique

# Return Address Spoofing – Shellcode

1. Stores the original return address in a R11
2. Moves structure into RAX, stores JMP RBX gadget as the new return address

```
pop     r11                     ; Pop original return address into r11
add     rsp, 8
mov     rax, [rsp + 24]
mov     r10, [rax]              ; Move JMP RBX gadget into r10
mov     [rsp], r10              ; Set return address to gadget
```

Full credit to @namazso for this technique

# Return Address Spoofing – Shellcode

3. Move intended function into R10
4. Preserves the original return address and RBX value in parameter structure

```
mov     r10, [rax + 8]      ; Move the intended function into r10
mov     [rax + 8], r11      ; Preserve original return address
mov     [rax + 16], rbx     ; Preserve original RBX value
```

Full credit to @namazso for this technique

# Return Address Spoofing – Shellcode

5. Stores the address of parameter structure in RBX (starting with “fixup” address)
6. Jumps to the intended function

```
lea     rbx, [fixup]
mov     [rax], rbx           ; Store address of “fixup” in param struct
mov     rbx, rax             ; Preserve param struct address in rbx
jmp     r10                  ; Jump to the intended function
```

Full credit to @namazso for this technique

# Return Address Spoofing – Shellcode

7. Intended function returns to JMP RBX gadget, jumps to “fixup” address
8. Restores clobbered registers and jumps to the original return address

```
fixup:
  sub    rsp, 16
  mov    rcx, rbx          ; Restore address of param struct
  mov    rbx, [rcx + 16]   ; Restore original value of RBX
  jmp    QWORD [rcx + 8]   ; Jump to the original return address
```

Full credit to @namazso for this technique

# AceLdr

https://github.com/kyleavery/AceLdr

- Bypasses every referenced scanner
- Easy to use – import a single CNA script
- Encryption using SystemFunction032
- Dynamic memory encryption using a secondary heap
- Code obfuscation and encryption using FOLIAGE
- Delayed execution using WaitForSingleObject
- Return address spoofing at execution for InternetConnectA/W, NtWaitForSingleObject, and RtlAllocateHeap

# Final Notes

- The techniques demonstrated in AceLdr and other PoCs can be easily ported to other projects
- Each technique demonstrated is meant to bypass existing scanners, but new detection methods may come out
- Certain scanners and techniques were intentionally left out because they detect a specific implementation of a bypass or bypass a specific scanner

# References

https://github.com/SecIdiot/FOLIAGE

https://github.com/waldo-irc/YouMayPasser

https://github.com/Cracked5pider/Ekko

https://github.com/waldo-irc/MalMemDetect

https://www.unknowncheats.me/forum/anti-cheat-bypass/268039-x64-return-address-spoofing-source-explanation.html

https://www.arashparsa.com/bypassing-pesieve-and-moneta-the-easiest-way-i-could-find/

https://www.forrest-orr.net/post/masking-malicious-memory-artifacts-part-ii-insights-from-moneta