black hat ASIA 2023

MAY 11–12

BRIEFINGS

You can Run, but you can't Hide

Finding the Footprints of Hidden Shellcode

John Uhlmann
@jdu2600

whoami

Security Research Engineer at Elastic

- Elastic Defend (“EDR”) developer
- Elastic Security Labs Blogger
- https://www.elastic.co/blog/author/john-uhlmann

elastic

elastic security labs

Agenda

1. Why do security products scan memory?
2. Memory scanning & evasion recap
3. Detection opportunities for hidden shellcode
   - Detection via immutable code page principle violations
   - Detection via CFG bitmap anomalies
4. Hunting via process behaviour summaries

Why do security products scan memory?

- On Windows x64, Microsoft has –
- hardened the kernel,
- claimed the hypervisor, and
- made private executable memory an indefensible boundary for kernel-mode security products.

- This just leaves memory scanning.
- It’s not perfect, but it’s still a valuable defensive layer.

Overview of memory scanners

Generic Scanners

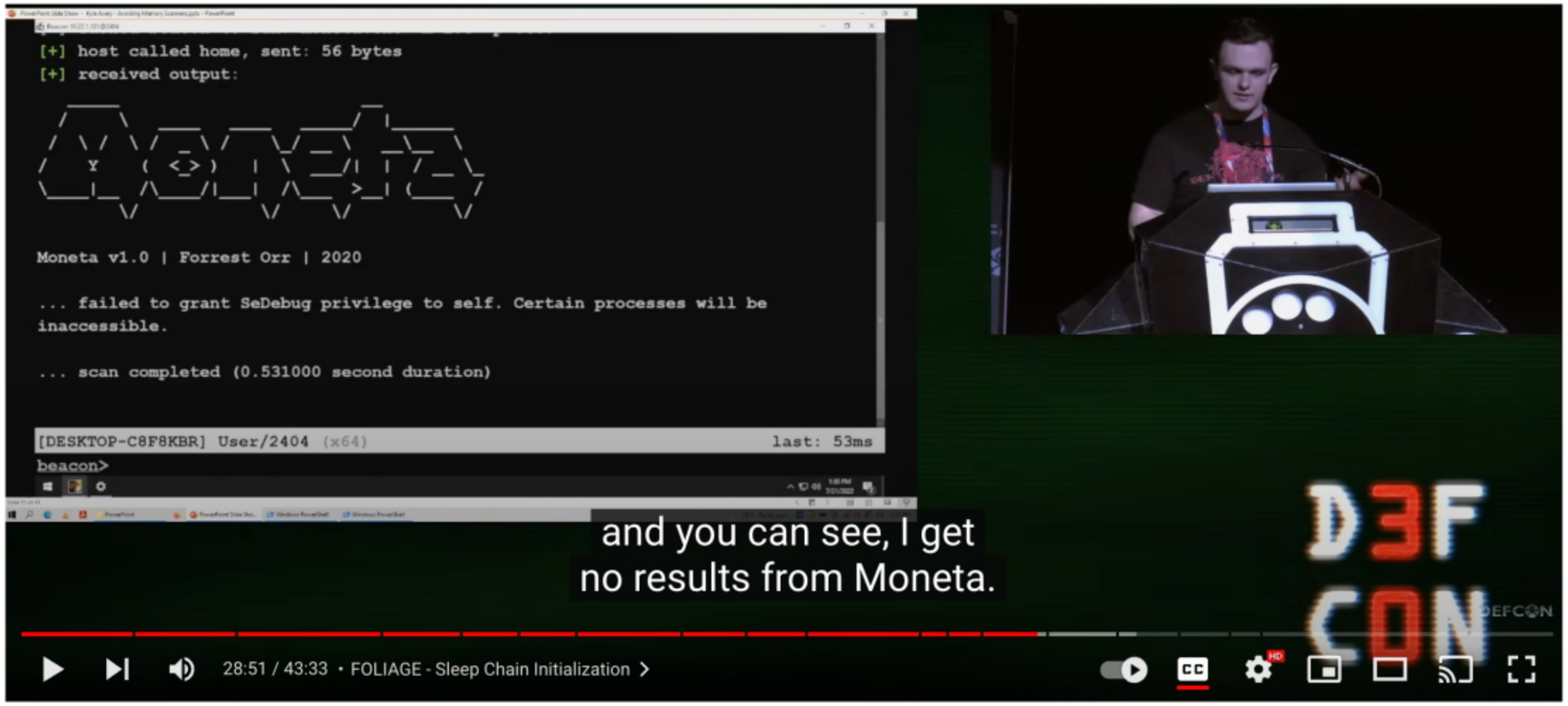
- YARA - memory content signatures
- PE-sieve - image metadata anomalies and content heuristics
- Moneta - memory metadata anomalies

Evasion recap

- Gargoyle - memory protection fluctuation via APC timer and ROP chain
- obfuscate-and-sleep - encrypted state fluctuation via post-sleep stub
- FOLIAGE - encrypted state fluctuation via APC timers and context manipulation
- Shellcode Fluctuation - memory protection fluctuation via post-sleep indirect stub
- DeepSleep - memory protection fluctuation via post-sleep ROP chain
- Ekko - encrypted state fluctuation via timer queues and context manipulation

- Scheduled Tasks ;-)

Evasion recap



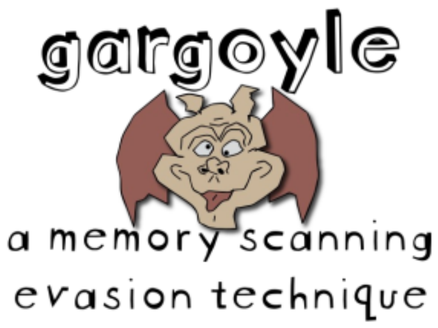
Kyle Avery - Avoiding Memory Scanners: Customizing Malware to Evade YARA, PE-sieve, and More

https://forum.defcon.org/node/241824

Evasion – key concept

"a common technique for reducing computational burden is to limit analysis on executable code pages only" - Josh Lospinoso

https://lospi.net/security/assembly/c/cpp/developing/software/2017/03/04/gargoyle-memory-analysis-evasion.html




```
VirtualProtect(pShellcode, sizeof(shellcode), PAGE_READWRITE, &OldProtect);
```

Niche memory scanners

- Patriot - anomalous thread CONTEXT structures
- Hunt-Sleeping-Beacons - anomalous Wait call stacks
- TickTock - anomalous timer-queue timers

Immutable code page principle violations

- Once code pages are written they should never change.
- The memory protection progression for code pages should only be RW to RX.
- Microsoft-Windows-Threat-Intelligence PROTECTVM_LOCAL ETW events
- `IsExecutable(LastProtectionMask) && !IsExecutable(ProtectionMask)`
- (Optionally) Anomalous call stack detection

```
Microsoft Visual Studio Debug Console
[*] Enabling Microsoft-Windows-Threat-Intelligence (KEYWORD_PROTECTVM_LOCAL)
[*] Monitoring for VirtualProtect() for 30 seconds
[.] java.exe 0000014F66300000 RW- => RWX
[.] Ekko.exe 00007FF7962A0000 RW- => RWX
[.] Ekko.exe 00007FF7962A0000 RWX => RW-
[.] Ekko.exe 00007FF7962A0000 RW- => RWX
[!] Ekko.exe 00007FF7962A0000 is fluctuating
[.] Ekko.exe 00007FF7962A0000 RWX => RW-
[*] Done
```

An interesting discovery

Gabriel Landau

I think I just made an interesting discovery. If you VirtualAlloc(RWX), it modifies the CFG bitmap accordingly. If you then VirtualAlloc(RW), the CFG bitmap stays as-is.

Gabriel Landau

We may be able to use this to find DLL hollowing and gargoyle-style?

Control Flow Guard bitmap recap

- Time efficient lookup of valid indirect call targets
- One bitmap per process
- Each 2 bits corresponds to 16 virtual addresses
- x64 bitmap is 2TB – mostly shared or reserved
- PE files bring their own bitmap
- Copied to the correct offset in process bitmap during image load
- Permissive backwards compatibility for JIT
- Memory manager simply marks all executable private addresses as valid targets

CFG bitmap anomalies

- The VAD tree only stores original protection and current protection.
- The CFG bitmap (inadvertently) records the location of all private memory addresses that are, **or have previously been**, executable during the lifetime of the process.
- This can be used to flag memory regions that have been changed from executable to non-executable.

Select Microsoft Visual Studio Debug Console

```
===== Hidden Executable Pages - scanning all processes =====
ShellcodeFluctuation.exe(912) - 1 hidden allocations
 * 0000023F5F6C0000 MEM_PRIVATE
   - 0000023F5F6C0000 +0x001000 RW- --- 1/1 hidden pages
```

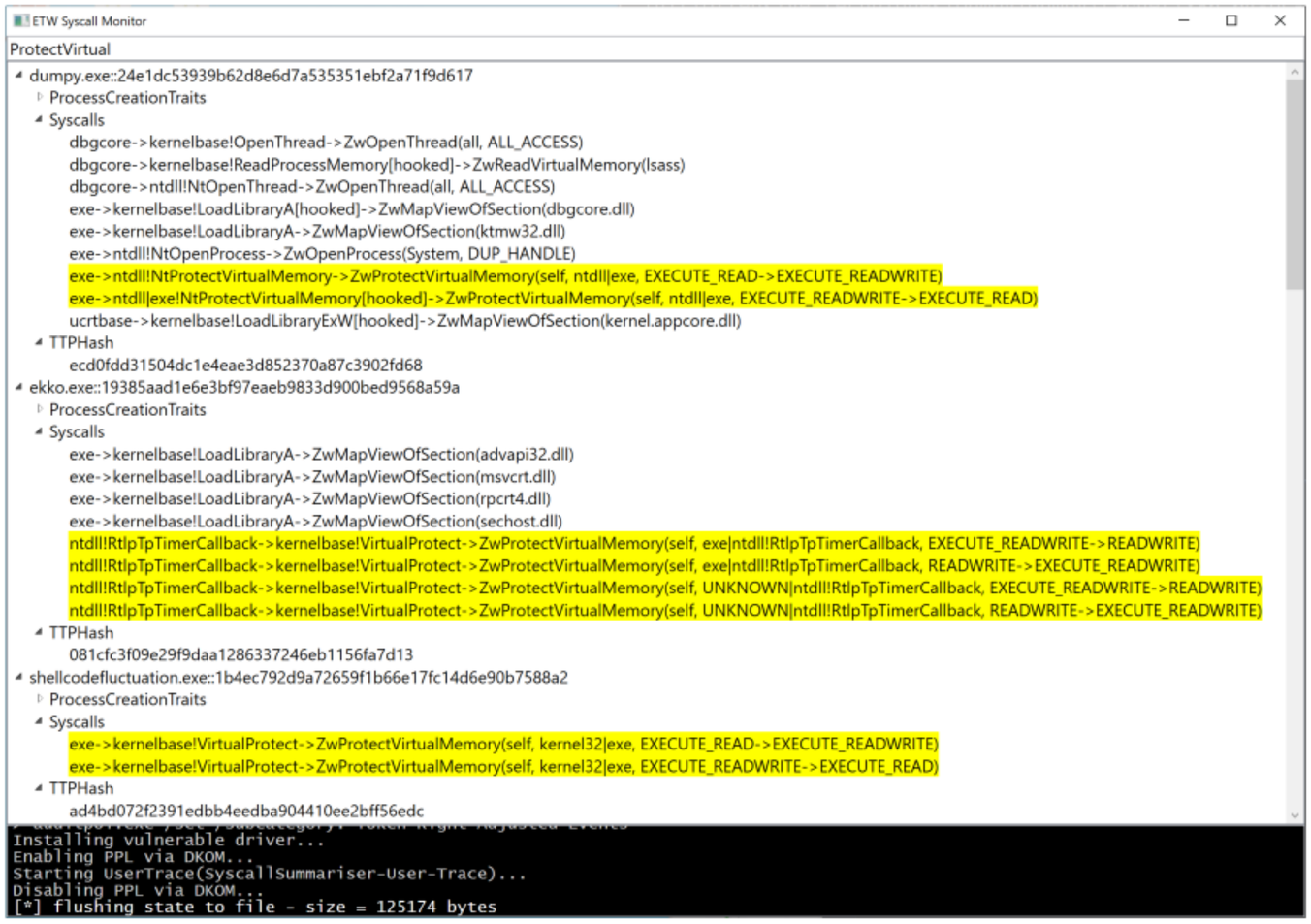
Evasion opportunities

- Protection fluctuation approaches are actually quite noisy.

- Hide your code pages in plain sight.
- Obfuscate them against current signatures ahead of time.
- Encrypt your data pages when not in use.

- Or launch in a new process every time.
- Scheduled Tasks etc.

Hunting via process behaviour summaries



Black Hat Sound Bytes

- Threat-Intelligence ETW can be used to detect violations of the immutable code page principle.
- The CFG bitmap can be used to detect shellcode hidden at a point-in-time via changed memory protections such as Gargoyle.
- Kernel telemetry can be used to construct process behaviour summaries – which can be used to identify behavioural outliers for more detailed investigation.

- But, without intervention from Microsoft, private executable memory will likely remain an indefensible boundary for kernel-mode security products.

Questions

Tools

- https://github.com/jdu2600/EtwTi-FluctuationMonitor
- https://github.com/jdu2600/CFG-FindHiddenShellcode
- https://github.com/jdu2600/Etw-SyscallMonitor

Detection References

- https://github.com/VirusTotal/yara
- https://github.com/hasherezade/pe-sieve
- https://github.com/forrest-orr/moneta
- https://www.elastic.co/security-labs/hunting-memory
- https://www.elastic.co/blog/detecting-cobalt-strike-with-memory-signatures
- https://github.com/joe-desimone/patriot
- https://github.com/thefLink/Hunt-Sleeping-Beacons
- https://github.com/WithSecureLabs/TickTock

Evasion References

- https://github.com/JLospinoso/gargoyle
- https://www.cobaltstrike.com/blog/cobalt-strike-3-12-blink-and-youll-miss-it/
- https://github.com/realoriginal/foliage
- https://github.com/mgeeky/ShellcodeFluctuation
- https://github.com/thefLink/DeepSleep
- https://github.com/Cracked5pider/Ekko
- https://www.blackhillsinfosec.com/avoiding-memory-scanners/

OS References

- https://en.wikipedia.org/wiki/W%5EX
- https://learn.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-virtualprotect
- https://github.com/jdu2600/Windows10EtwEvents/blame/master/manifest/Microsoft-Windows-Threat-Intelligence.tsv
- https://www.elastic.co/security-labs/finding-truth-in-the-shadows