MODULE STOMPING

Aliz Hammond, F-Secure Countercept

BSidesSG 2019

ABOUT ME

- Aliz Hammond
- Based in Singapore
- Experience in fuzzing, binary exploitation, Windows kernel
- Researcher supporting Threat Hunters

MODULE STOMPING

- The attack itself
- Detection
  - Existing techniques
  - My new technique
  - Other things you can use this research for

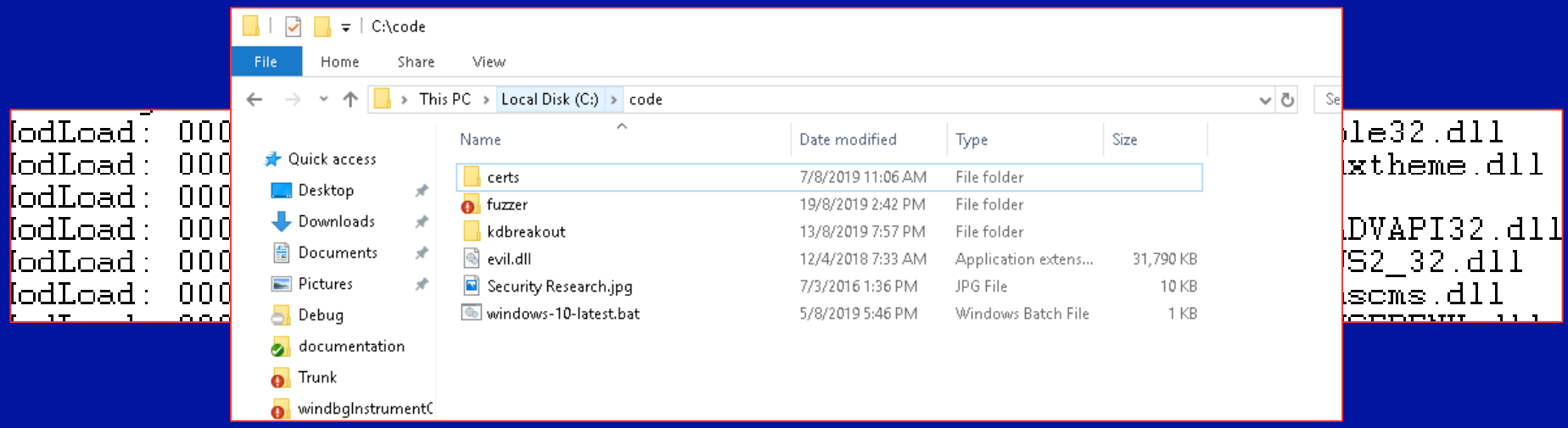
THE ATTACK

THE ATTACK

- Way of hiding malicious code

THE ATTACK

- Write DLL payload to disk, inject it into some legit process
  1. Write your evil code into C:\windows\temp\evil.dll
  2. OpenProcess
  3. CreateRemoteThread(… LoadLibrary("C:\windows\temp\evil.dll"))



THE ATTACK

- Reflective load
  1. Create a thread in a legit process as before
  2. Allocate memory in the target process
  3. Set this memory to be executable
  4. Download your payload code into it
- No disk artefacts
  - But does leave m

Windows 10 Creators Update can detect reflective Dynamic-Link Library (DLL) loading in a variety of high-risk processes, including browsers and productivity software, Microsoft says.

This is possible because of function calls (*VirtualAlloc* and *VirtualProtect)* related to procuring executable memory, which generate signals for Windows Defender Advanced Threat Protection (Windows Defender ATP).

Reflective DLL loading, the software giant explains, relies on loading a DLL into a process memory without using the Windows loader. First described in 2008, the method allows for the loading of a DLL into a process even if the DLL isn't registered with the process.

SECURITYWEEK.COM

THE ATTACK

- Enter – “Module stomping”
  1. Open a legit process
  2. Inject a thread via CreateRemoteThread
  3. Injected thread loads a legitimate but unnecessary system DLL
  4. Overwrite the module with our own malicious module

THE ATTACK

- All module loads are of legit files
- All executable memory backed by system DLLs
- No disk artefacts
- Easy to do yourself
- In threat emulation toolkits

EXISTING DETECTION METHODS

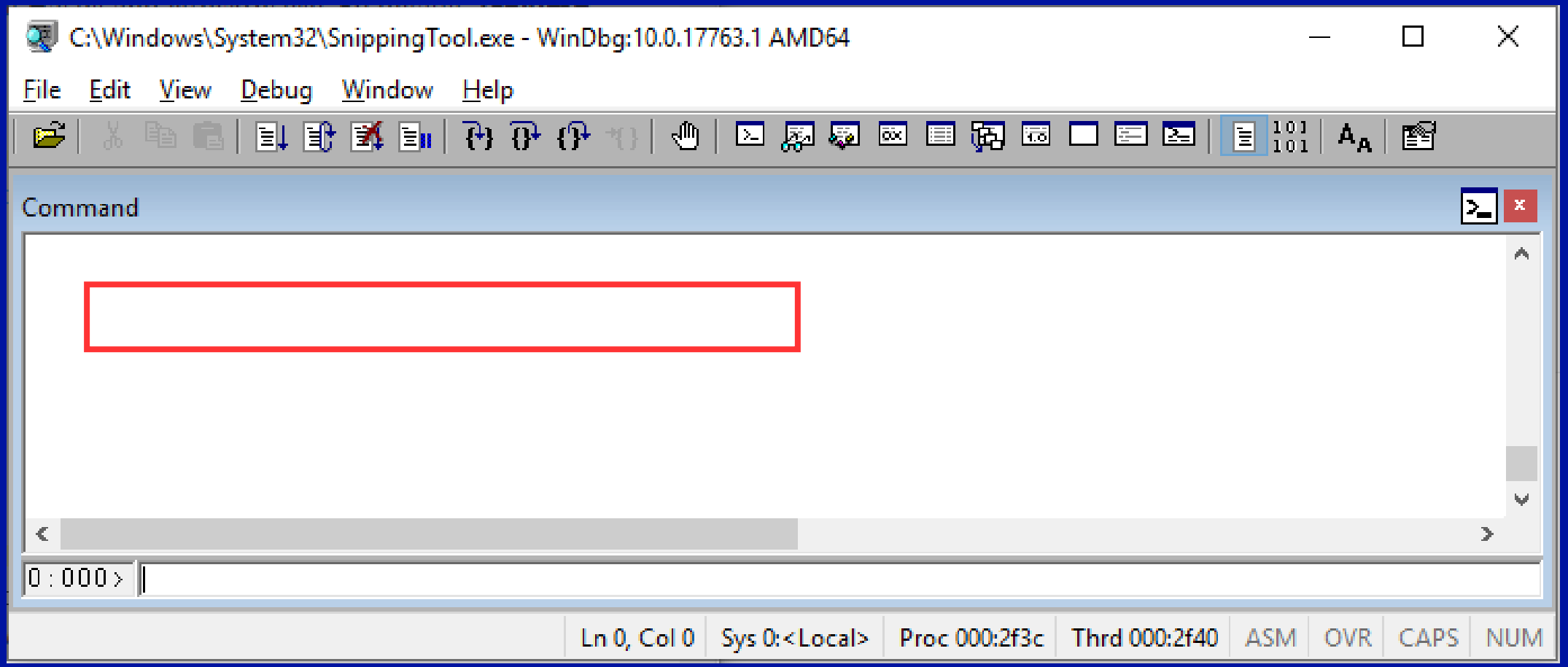
DETECTION

- Compare all modules in memory to their counterparts on disk
  - Very slow
  - Impossible in some cases
- Download legit version
  - Even slower
  - Impossible in some cases

DETECTION

- Windbg's "!ChkImg"
  - Downloads original image, checks against it

DETECTION



DETECTION

- Windbg's "!ChkImg"
  - Downloads original image, checks against it
  - Only useful when debug symbols are available
    - Almost all of Windows
    - Chrome
    - Firefox
  - Not designed for adversarial use

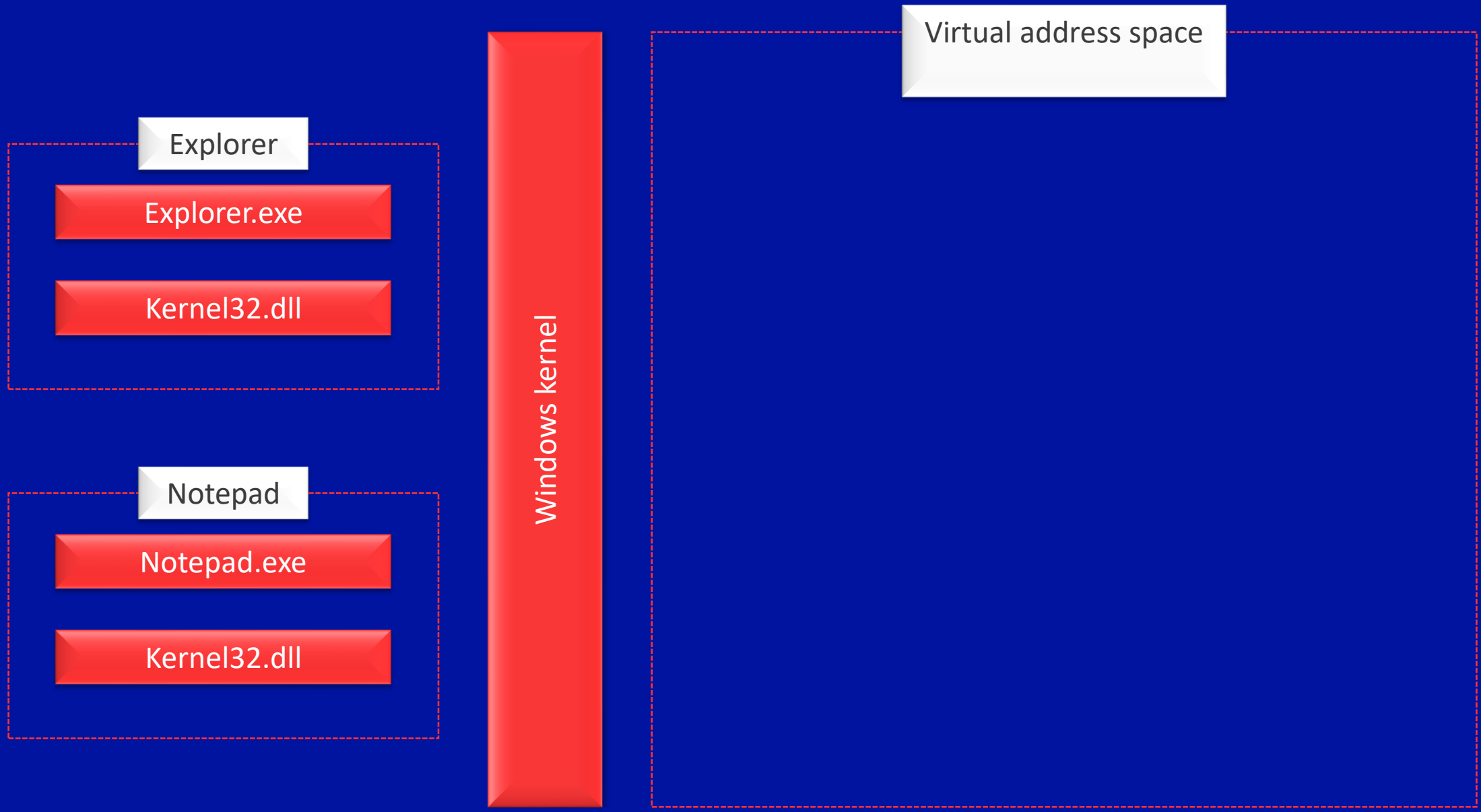
- Still useful for offline analysis

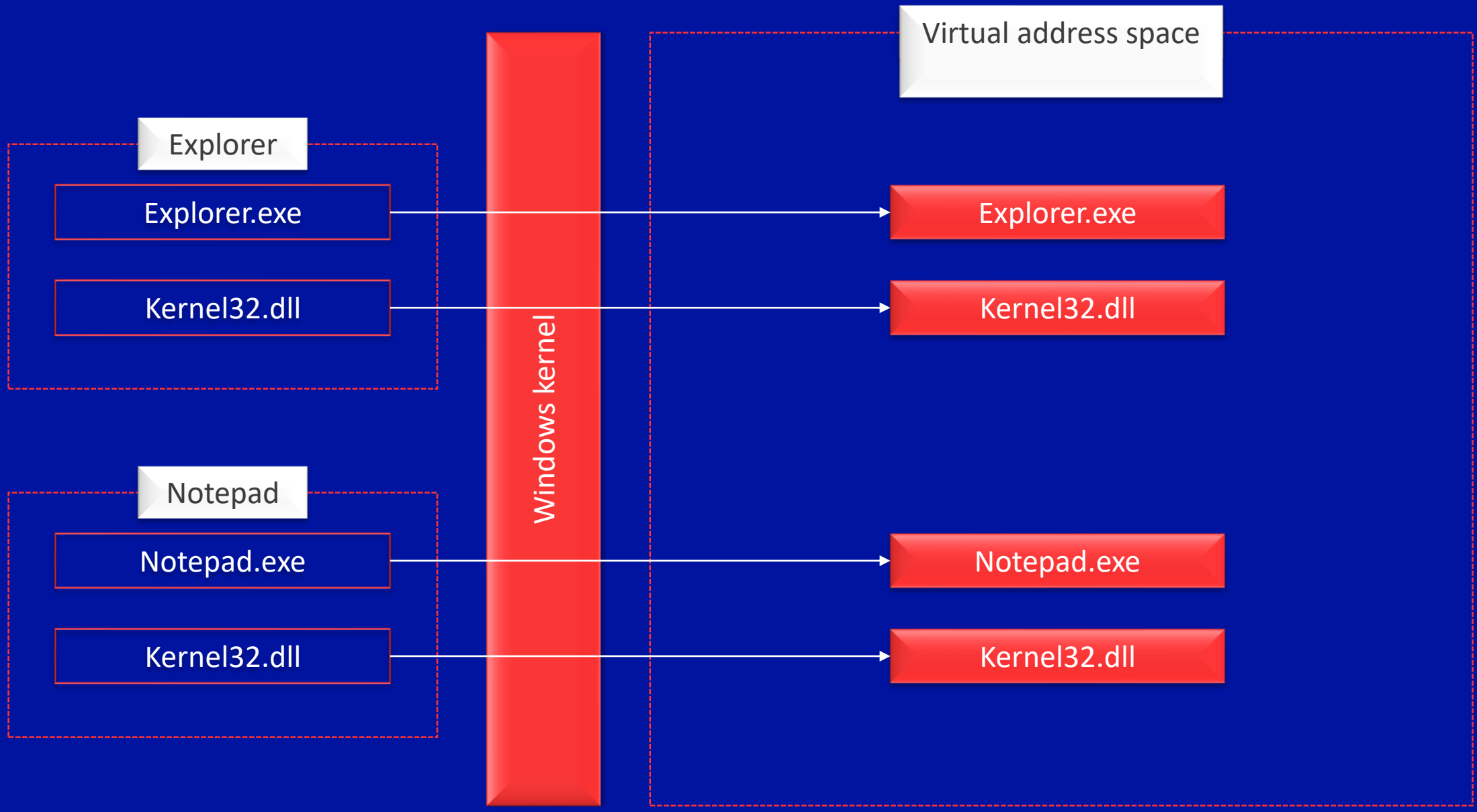
DETECTION

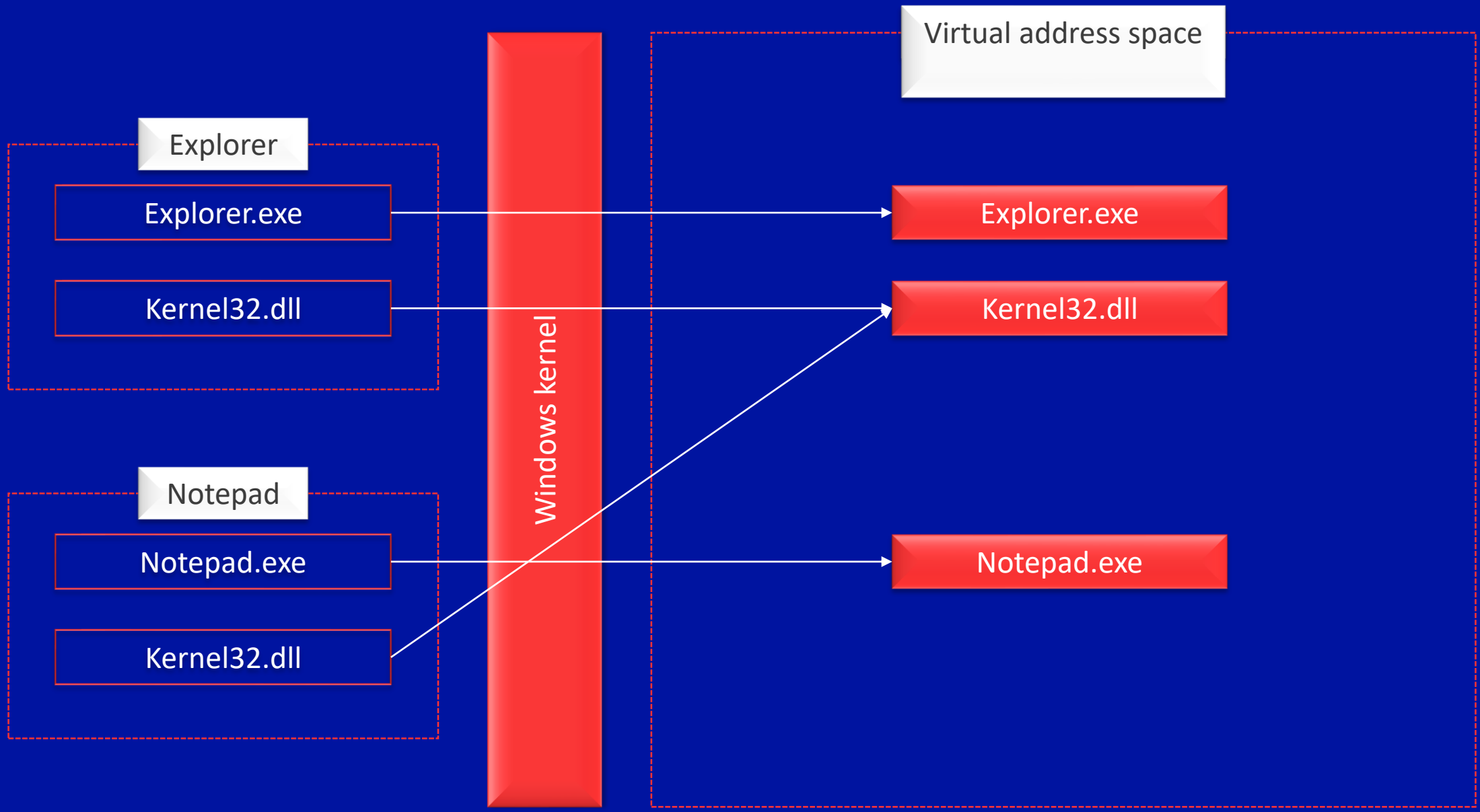
DETECTION

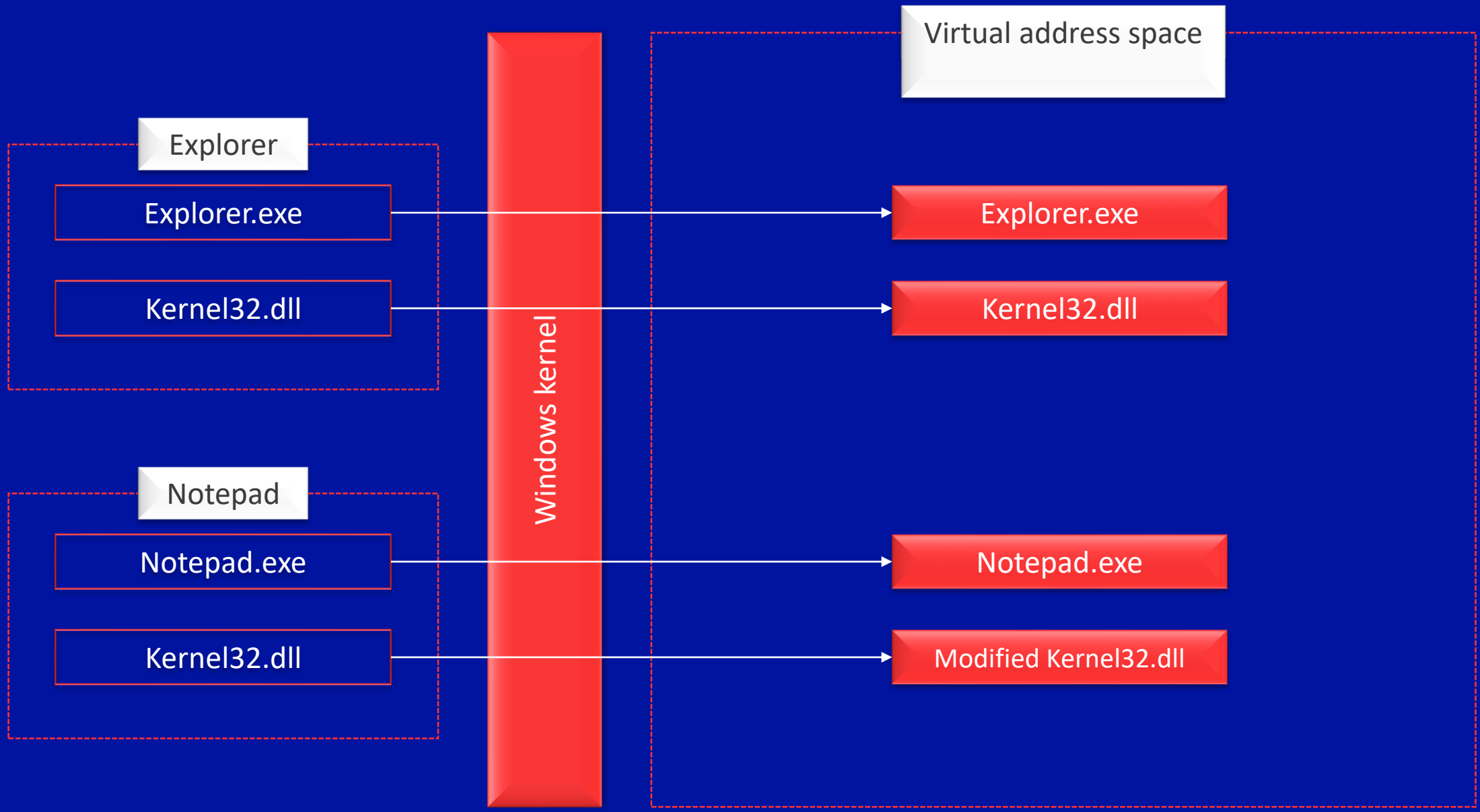
- Wouldn’t it be nice if we could:
  - Detect in real-time
    - Fast enough for a background scanner
  - In a reasonably secure manner
    - Hardened against usermode attackers
  - Without needing debug symbols
  - Without needing the original module on disk

MEMORY MANAGEMENT THEORY









DETECTION

- Does Windows keep track of this information anywhere?
  - Yes it does!
  - It's in the kernel
  - But it's undocumented ☹
  - But it's pretty safe to access ☺
- The "PFN Database"
  - PFN meaning "Page Frame Number"
- A flat array of structs
- Windbg has "!pte" and "!pfn" commands

```
1: kd> !pte KERNEL32!BeepImplementation
                                           VA 00007ff9fbbe2160
PXE at FFFF89C4E27137F8    PPE at FFFF89C4E26FFF38    PDE at FFFF89C4DFFE7EE8    PTE at FFFF89BFFCFDDF10
contains 0A00000002C44867  contains 0A00000070EDB867  contains 0A0000006430F867               0000005098C025
pfn 2c44       ---DA--UWEV  pfn 70edb      ---DA--UWEV  pfn 6430f      ---DA--UWEV  pfn 5098c      ----A--UREV

1: kd> !pfn 5098c
    PFN 000          address FFFFEB0000F1CA40
    flink       00000001  blink / share count 00000001  pteaddress FFFF89BFFCFDDF10
    reference count 0001    used entry count  0000      Cached    color 0   Priority 5
    restore pte B3C1300002070  containing page 06430F  Active    M
    Modified
```

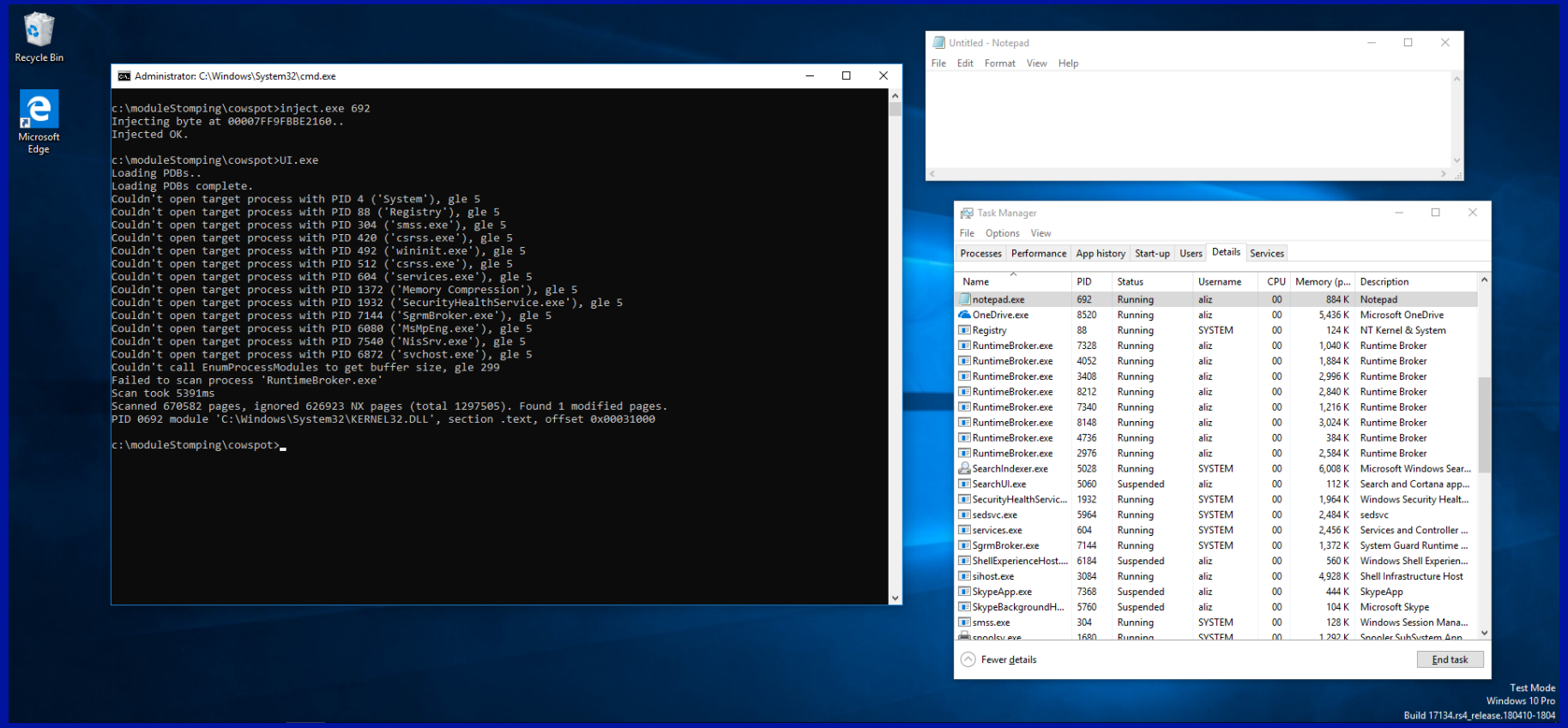
```
1: kd>
```

LIVE DETECTION

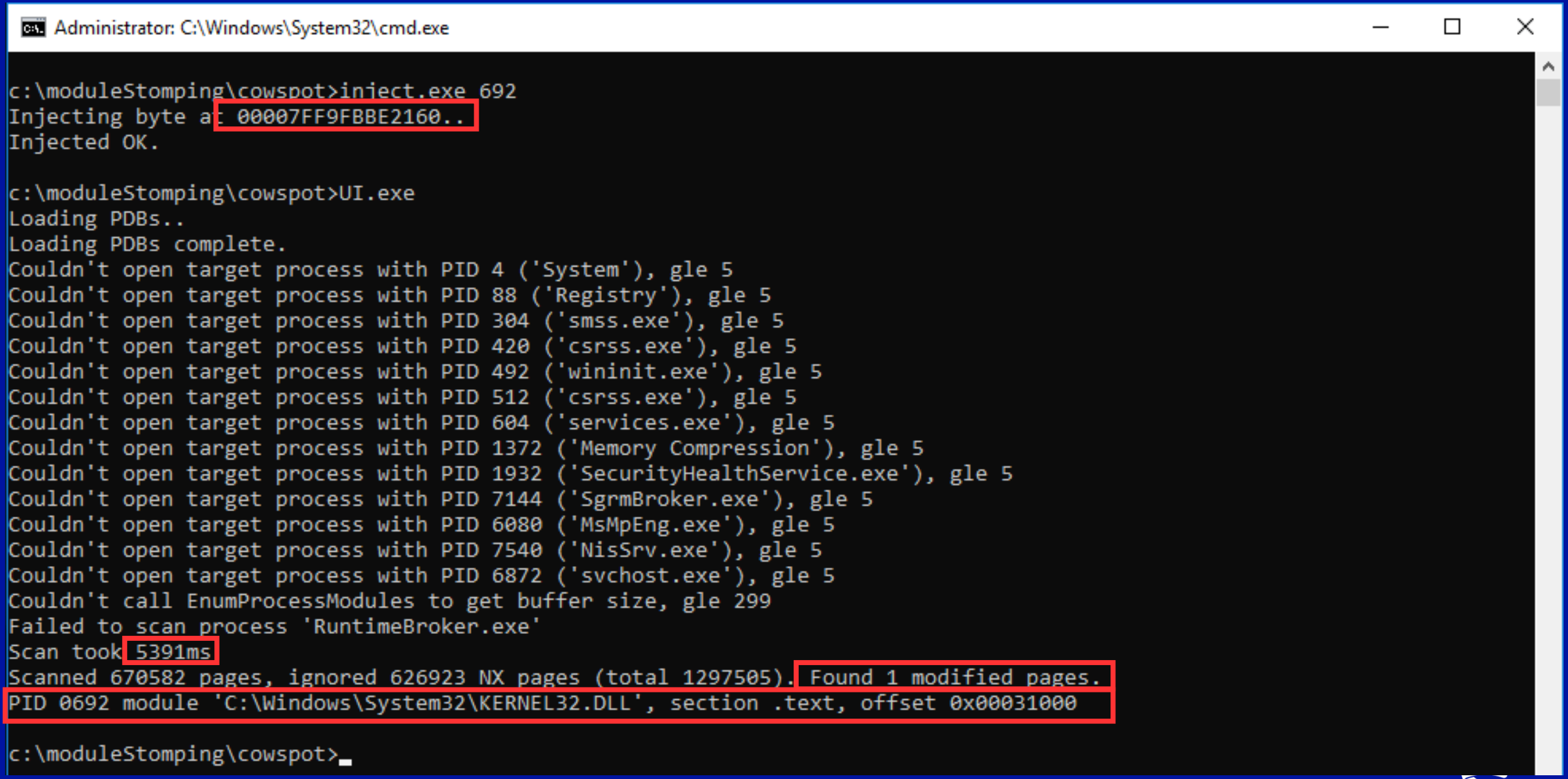
LIVE DETECTION

- All we need to do is query the PFN database
- It's in kernel space, so we write a simple driver to query it
- The address and structure is undocumented
  - Use debug symbols to acquire the address
    - "DIA SDK" is installed with VS, underutilised capability IMO!
  - This part done by userspace component
- Surprisingly easy to implement

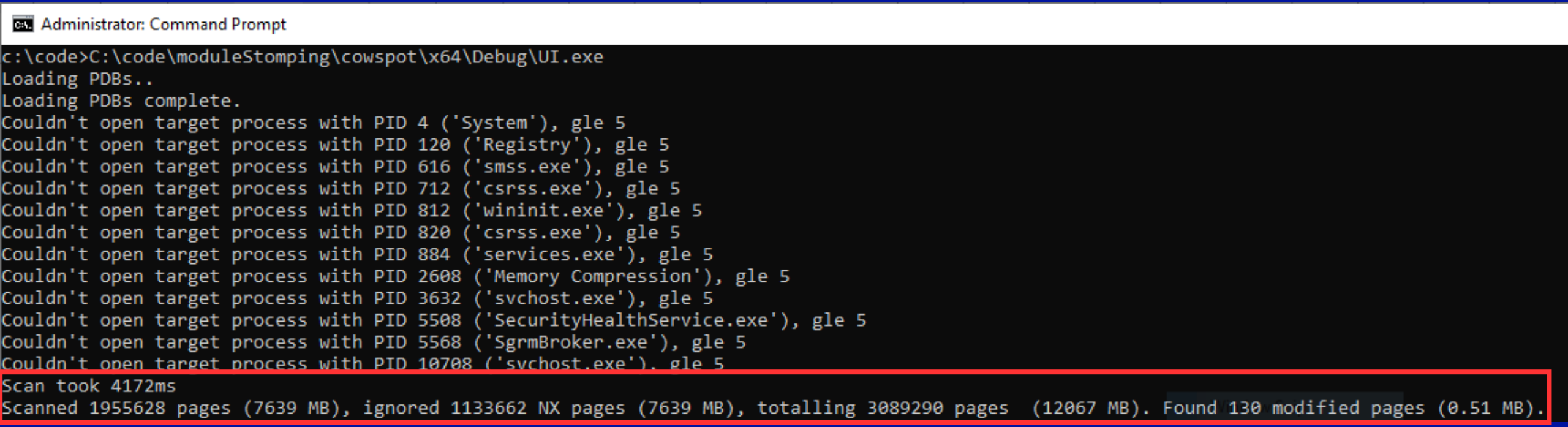
LIVE DETECTION



F-Secure



LIVE DETECTION



LIVE DETECTION

- 12GB RAM in use at time of scan
  - ~7GB executable
- Scan took 4.1 seconds
- Found 130 modified pages (~500KB)

FUTURE WORK

FUTURE WORK

- You can download and run the driver
- You can call it from your own C code
- Probably useful for more than just module stomping
  - Specific code patches can become signatures

```
“Name”: “Generic_amsi_bypass”:
      modules = [ “amsi.dll” ]
      offsets = [ 0x1234 ]
```

- It’d be great if this was useful to people

FUTURE WORK

- Temporary downsides
  - Driver is self-signed right now
    - You need to boot in Test Mode, which means disabling Secure Boot
  - Not heavily tested
  - It’s in kernel space, it’s possible it’ll BSoD your box
    - Pretty unlikely because the kernel component is so thin and simple
- These problems will go away

FUTURE WORK

- Inherent downsides
  - We need to pull down symbols to locate the PFN database
    - We can build a database of offsets and build it into the binary if need be
  - Passing location of PFN to driver introduces a security weakness
  - We depend on PFN not changing
  - There are false positives
    - AVs that patch things, JIT engines, …
    - Uncommon enough that we can memcpy these ranges

SUMMARY

SUMMARY

- We can now scan for modified file-backed memory in real-time
  - So can you
- Use it to detect patches in code quickly

- Code is available online
- Blogpost is available online
  - In three parts – first two all about the attack, third about detection

- Please do things with it so I can justify making it better

https://blog.f-secure.com/category/threats-research
https://github.com/countercept/ModuleStomping
@AlizTheHax0r