



Software Reverse Engineering

COMP 272 | Spring 2022 | University of the Pacific | Jeff Shafer

Anti-RE 2

KNOW YOUR MALWARE 101



Malware



Stuxnet – To Kill a Centrifuge



NATANZ, IRAN -- CLOSE-UP


 INSTITUTE FOR SCIENCE AND
INTERNATIONAL SECURITY

IMAGE CREDIT: DIGITALGLOBE
DATE OF IMAGE: 16 SEPTEMBER 2002

THE GAS CENTRIFUGE URANIUM ENRICHMENT PLANT AT NATANZ, IRAN.

Imagine there was an industrial facility located deep out in the desert...

Stuxnet – To Kill a Centrifuge



... and it was full of gas centrifuges for uranium enrichment

Stuxnet – To Kill a Centrifuge



... and protected by a military who didn't like *you* very much

Stuxnet – To Kill a Centrifuge



... and you wanted the facility to suffer an *unfortunate accident*

Stuxnet – To Kill a Centrifuge



... without using methods (e.g. airstrikes) that would allow *blame* for the “accident” to be placed on you

Idea: Computer Virus



VIRUS

Stuxnet – To Kill a Centrifuge



- How to we get our malware to an isolated, network air-gapped facility in the middle of the Iranian desert?
- People come and go from the facility regularly (e.g. contractors, employees)
- Use spies or other malware to infect **USB keys** that contractors regularly carry into the facility and connect to computers inside

Stuxnet – To Kill a Centrifuge



- **What if the contractors don't have access to all the computers?**
- Malware contains a *worm* that will allow it to spread inside the air-gapped network
- **How can we help ensure malware will spread to all computers inside?**
- Cash in *four zero-day vulnerabilities* that three-letter-agencies were hoarding for a special project
 - Spread from USB: PNK/PIF vulnerability (viewing the icon in Windows Explorer executes the malicious code!)
 - Spread over network: Remote code execution on PC with printer sharing enabled
 - Two privilege escalation vulnerabilities

Stuxnet – To Kill a Centrifuge



- **How do we ensure that our malware isn't detected?**
- Malware is signed by keys stolen (via spies!) from Jmicron and Realtek in Taiwan
 - Driver signing allows kernel-mode rootkit to be installed
- Safeguards
 - Malware will erase itself after specific date
 - Malware will only spread to a few other targets (worm is not aggressive)
 - Malware will become inert if PC isn't *intended target*

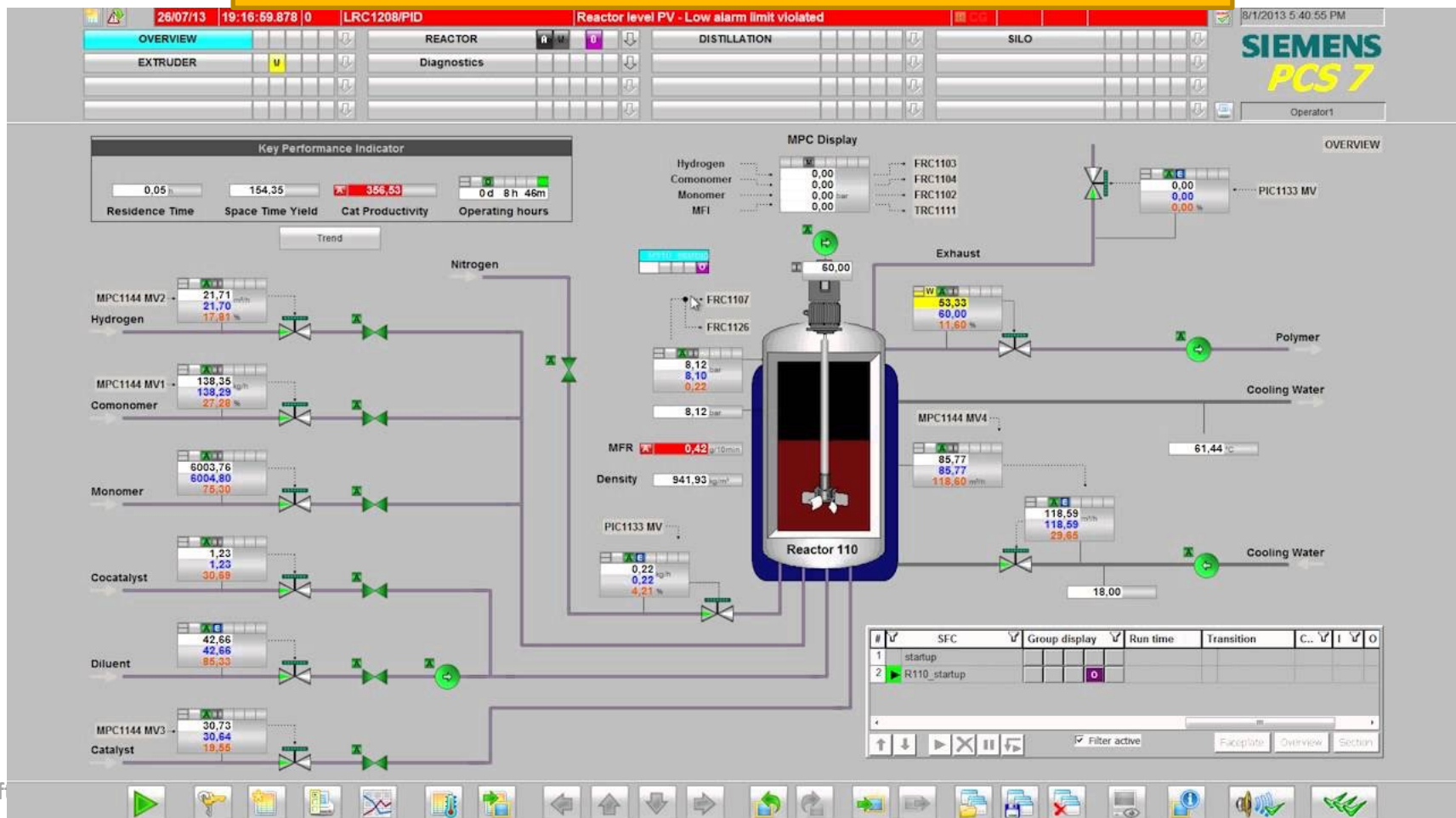
Stuxnet – To Kill a Centrifuge



- Besides spreading, what do we want the malware to do?
- Sabotage uranium enrichment centrifuges
- But make it look like innocent technical malfunctions, poor design, shoddy construction, poor quality materials due to embargo, anything *other than* evil hackers!
- These are high performance devices that require *exacting computer controls* to function properly

Stuxnet – To Kill a Centrifuge

Siemens PCS 7 Distributed Control System



Stuxnet – To Kill a Centrifuge

Siemens WinCC Monitoring and Control System – Runs on Windows!

The screenshot displays the Siemens WinCC flexible Advanced Graphics Designer software interface. The main window shows a monitoring and control screen for a centrifuge system. The interface includes a project tree on the left, a main graphics view with a conveyor layout and control buttons (START, STOP, RESET, MUTE), and a properties panel at the bottom.

Project Tree (Left):

- AutoLoader_HMI(PC 677 19" T)
- Screens
 - Add Screen
 - Template
 - Alarms
 - Control
 - CoriVeyor_Data
 - Conveyor_Destination
 - Main
 - Maintenance
 - Manual
 - Planning
 - PLC
 - Position 1 Timers
 - Position 10 Timers
 - Position 11 Timers
 - Position 12 Timers
 - Position 13 Timers
 - Position 2 Timers
 - Position 3 Timers
 - Position 4 Timers
 - Position 5 Timers
 - Position 6 Timers
 - Position 7 Timers
 - Position 8 Timers
 - Position 9 Timers
 - Position_Errors
 - Route_Data
 - SAP
 - Scanners
 - Timers
 - Tracking
 - Transfer_Flags
 - View
- Communication
 - Tags
 - Connections
 - Cycles
- Alarm Management
 - Analog Alarms
 - Discrete Alarms
 - Settings
- Recipes
- Historical Data

Main Graphics View (Center):

The main graphics view displays a conveyor layout with various positions (POSITION 1 to POSITION 13) and control buttons (START, STOP, RESET, MUTE). A legend titled "Pallet Destination Colour Code" is visible:

- 0 Hold Position
- 1 Joloda Position
- 2 Manual Unload Position

Properties Panel (Bottom):

The properties panel shows the "General" tab for a "One-Direction-Arrow-Sign-X-W1-6L_0" object. The object is a yellow arrow pointing right, with a "Preview" window showing the arrow graphic. The object is associated with the "PALLETT" tag and has a "Set" button and a "Clear" button.

Tools Panel (Right):

The tools panel shows a list of simple objects for drawing, including Line, Polyline, Polygon, Ellipse, Circle, Rectangle, TextField, IO Field, Date-Time Field, Graphic IO Field, Symbolic IO Field, Graphics View, Button *, Switch, and Bar.

Stuxnet – To Kill a Centrifuge

Siemens Step7 Controller Programmer – Runs on Windows!

The screenshot shows the 'HW Konfig' window for a SIMATIC 300(1) station. The rack configuration is as follows:

Steckplatz	Baugruppe	Bestellnummer	Farwware	MPI-Adresse	E-Adresse	A-Adresse	Kommentar
1	PS 307 10A	BES7 307-1KA00-0AA0					
2	CPU 315-2 PN/DP	BES7 315-2EG10-0AB0	V2.3				
X7	MPI/DP				204*		
X2	PN-IO				206*		
3							
4	CP 343-1 IT	6GK7 343-1EX30-0XE0	V1.0		268..283	268..283	
5	DI16wAC120/230V	BES7 321-1FH00-0AA0			4..5		
6	DI16wAC120/230V	BES7 321-1FH00-0AA0			8..9		
7	DO16wAC120V/0.5A	BES7 322-1EH00-0AA0				12..13	
8	DO16wAC120V/0.5A	BES7 322-1EH00-0AA0				16..17	
9	AI2x12Bit	BES7 331-7KB00-0AB0			336..339		
10	AO2x12Bit	BES7 332-5HB00-0AB0				352..355	

The network diagram shows a PROFIBUS DP-Mastersystem (1) connected to a PROFIBUS DP-Mastersystem (2500). An Ethernet (1) PROFINET-IO-System (100) is also shown, connected to various modules like SCALAN, IM151-3, IE-PB, ET 200, SIMO, and MAST.

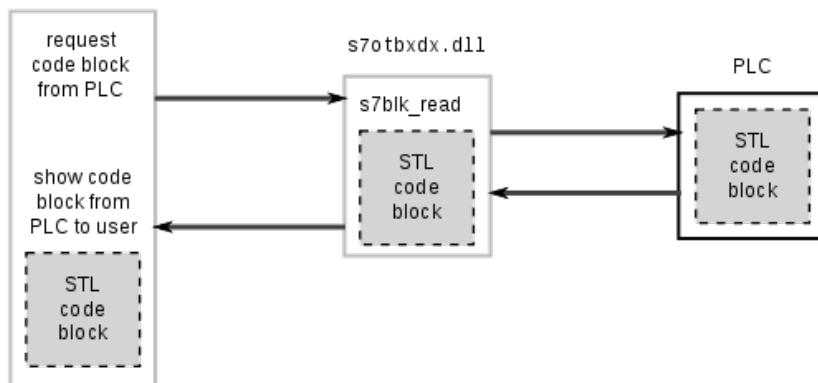
Stuxnet – To Kill a Centrifuge



- Besides spreading, what do we want the malware to do?
- Let's speed up and slow down the centrifuge in dangerous ways, and lie to the monitoring system

Normal Operation:

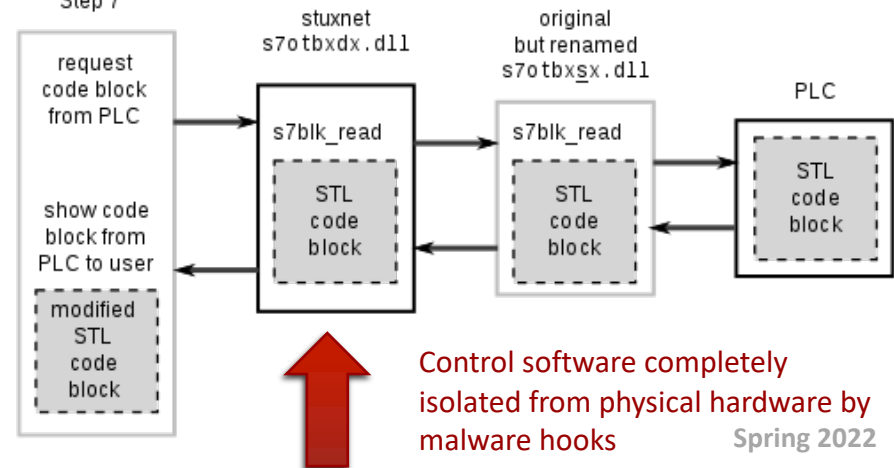
Step 7



Software Reverse Engineering

Malicious Operation (“Hooked”):

Step 7



Control software completely isolated from physical hardware by malware hooks
Spring 2022



Stuxnet – To Kill a Centrifuge



- Required very detailed (inside) knowledge of centrifuge design and construction
 - Centrifuges were 1960's-70's Pakistani designs
- Required very detailed (inside) knowledge of control system monitoring centrifuges
- Malware was tailored for a very specific set of control systems and devices
 - Only attack Siemens S7-300 PLCs controlling variable-frequency drives from two vendors (Vacon and Fararo Paya), spinning between 807Hz and 1210Hz
 - *Most locations in the world? Malware does nothing at all*

Stuxnet – To Kill a Centrifuge

- “To Kill a Centrifuge”
 - <https://www.langner.com/wp-content/uploads/2017/03/to-kill-a-centrifuge.pdf>
- Attack #1 – Induce minor malfunctions (overpressure) intended to degrade plant operations, *delay* nuclear production and **remain undetected**
- Attack #2 – Induce major malfunctions even at the risk of being detected
 - *“History’s first field experiment in cyber-physical weapon technology”*

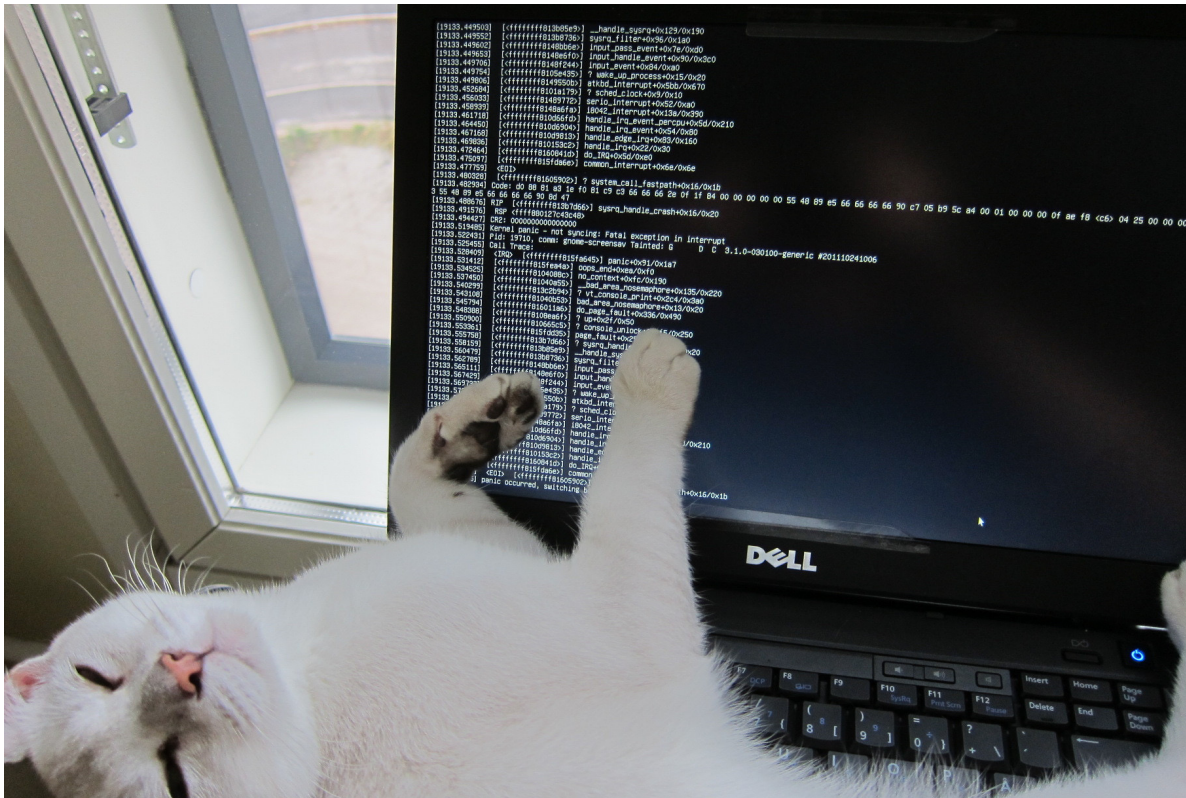


Anti-RE



Life as a Malware Analyst

➤ The malware authors are actively trying to subvert you



➤ *At a minimum, they want to obfuscate their malware to avoid automated detection*

➤ *And they really don't like you analyzing their code either...*

Constant game of cat and mouse





Packers



Recap – Packers

- Method to hide malicious program from detection
 - Might *compress* original malware
 - Might *encrypt* original malware (“crypter”)
 - Might *byte-fiddle* (XOR, ...) original malware



Recap – Packers

- *Here's an executable – Is it packed?*
- Signs
 - Few readable strings
 - Few imports in IAT
 - High entropy in program section (i.e. program sections are “too random”)
 - Normal code entropy: 5-6 bits per byte
 - Packed code entropy: >7 bits per byte
 - You get lucky / malware author is inexperienced
 - Program sections or embedded strings contain name of packer

Recap – Packers

- You only see the decompression routine
 - Real malware is a compressed/encrypted blob
- Goal: See the extracted blob without wasting time understanding intricate details of the unpacker
- Challenge: Each unpacker is different!
 - Different techniques to conceal code
 - Different techniques to resist debuggers

Methods to *Deal With* Packed Malware

- Method 1 – Direct Memory Dump
- Method 2 – Selective Debugging w/Memory Dump
- Method 3 – Don't Dump, Just Debug

Method 1 – Direct Memory Dump

- **Idea:** Dump the malware executable from memory after unpacking
 - No skill required! 😊

- **Demo #1**
 - Disable ASLR via CFF Explorer (“DLL can move”)
 - Detonate malware
 - Attach to active malware with standalone Scylla
 - Fix IAT, Get Imports, and then Dump
 - Result will have both unpacking code + unpacked malware

- **Problem:** Can’t run the resulting dump. Original Entry Point (OEP) still points to original unpacker code
 - *Would have to wildly guess what correct location is*

Method 2 – Selective Debugging

- **Idea:** Run the malware in the debugger until it unpacks and jumps to unpacked code, then dump contents from memory
 - *As practiced in Lab 8*
- **Advantage:** You can observe the Original Entry Point (OEP) and fix the dumped executable
 - Better chance of obtaining a *runnable* executable
 - The better the dumped executable, the more useful it will be in IDA

Method 2 – Selective Debugging

➤ Demo #2

- Disable ASLR via CFF Explorer (“DLL can move”)
- Load malware into debugger (x64dbg)
- Locate end of unpacker and set breakpoint there
 - *Finding this location requires skill/detective work*
- Run to breakpoint, allowing malware to unpack
- Carefully single-step to jump into unpacked code
 - This is the new OEP – You discovered it!
- Dump unpacked process (via OllyDumpEx plugin)
- Fix IAT and OEP (via Scylla plugin, IAT Autosearch, Get Imports)

Finding the End of the Unpacker (1)

- **Thought process for *(potentially)* helpful shortcut**
- **Assumptions**
 - The original binary has no idea it will be packed
 - The packing utility has no idea about the specific binary that will be packed
 - Thus, the unpacker logic, when it uses the stack, has to eventually clean up the stack by the end of the unpacking stub before it jumps to run the now-unpacked binary
- **Shortcut**
 - Set a hardware breakpoint on the first element of the stack
 - Sooner or later (probably sooner), you will arrive at the end of the unpacker right before a jump or call to the unpacked binary

Finding the End of the Unpacker (2)

- **A different thought process for (*potentially*) helpful shortcut**
- **Assumptions**
 - The unpacked binary must go *somewhere* – You need to find that location
 - Perhaps a PE section has a *real-size* of 0 bytes but a *virtual-size* of many bytes?
 - Perhaps the packed binary calls a single memory allocation function (`VirtualAlloc`)?
 - Perhaps there's a huge block of 0's in the file?
- **Shortcut**
 - Set a hardware write breakpoint at the first and last address of your suspected region
 - Run until you hit those breakpoints
 - Look around in the debugger (via “View as Disassembly”)
 - Does it look like *code* got placed in that region? Is the region full now?
 - Cross your fingers and hope that the unpacker is “nearly finished” now
 - Do some aggressive single-stepping or loop skipping (via run until selection) until you see a jump whose target address is inside your suspected region
 - This is the new OEP – You discovered it!

Method 3 – Don't Dump, Just Debug

- **Idea:** Malware unpacker may be too obfuscated to easily find jump to unpacked code, or there may be inscrutable problems fixing IAT
 - Do you **really need** to dump the unpacked file to answer your analysis questions about the malware?
 - Don't bother trying to find the end of the unpacking routine or the unpacked OEP
- Use the debugger to examine the original packed malware after it completes its unpacking work and the malware is running
 - Use behavioral analysis to generate *questions*
 - Use the debugger to *selectively* answer those questions

Method 3 – Don't Dump, Just Debug

➔ Demo #3

- ➔ Disable ASLR via CFF Explorer (“DLL can move”)
- ➔ Load into x64dbg
- ➔ Goal – We want to set a breakpoint on an API that the malware uses (`SetBPX` `FunctionName`)
 - ➔ Option 1: Guess likely API names based on behavioral analysis – Perhaps you observe file I/O or network I/O?
 - ➔ Option 2: Inspect program memory map for likely regions of unpacked executable code (ignoring DLLs, less likely)
- ➔ Run to that breakpoint!
 - ➔ Malware should be unpacked by this point
- ➔ In this *region* you can inspect strings, intermodular calls, etc...
 - ➔ Set hardware breakpoints and reset execution to run to them



Code Injection



Code Injection

- Malware doesn't always have to operate from its own `malware.exe` process
 - Malicious code can be injected into other user-space processes and the original `malware.exe` exits
- Advantage: Makes infection harder to spot, as there are only “normal processes” running on the system
- Code injection may be done by the *unpacker*

Code Injection – API Calls

1. **Get list of processes on system**
`CreateToolhelp32Snapshot, EnumProcesses`
2. **Obtain handle to target process**
`OpenProcess`
3. **Allocate space in memory of target process**
`VirtualAllocEx`
4. **Write injected code into target process**
`WriteProcessMemory`
5. **Run the code**
`CreateRemoteThread`

Many variations exist using normal Win32 API calls

Code Injection – API Calls

- Malware might call undocumented native API (NtXXX or ZwXXX) directly, bypassing the official Windows API functions
- 1. `CreateToolhelp32Snapshot`
-> `NtQuerySystemInformation`
- 2. `OpenProcess`
-> `NtOpenProcess`
- 3. `VirtualAllocEx`
-> `NtAllocateVirtualMemory`
- 4. `WriteProcessMemory`
-> `NtWriteProcessMemory`
- 5. `CreateRemoteThread`
-> `NtCreateThreadEx`



Anti-RE



Debugger Detection

- **Demo #4 – Methods to defeat debugger detection**
 - Manual register tampering
 - Manual code patching
 - Cloaking device (ScyllaHide plugin)