Dirty Little .NET Hooker

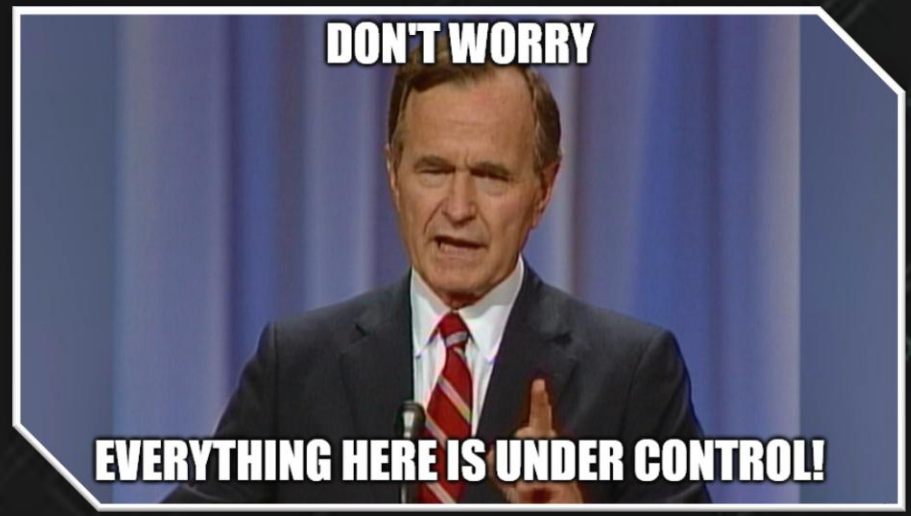
Hooking .NET applications with Frida for Red Team Ops

Who am I?

- Red Team Lead @ Ancestry
- Over a decade of experience
  - BlackHat – Dark Side Ops Course (Trainer)
  - Physical Security Entry Specialist
  - Global red team operations

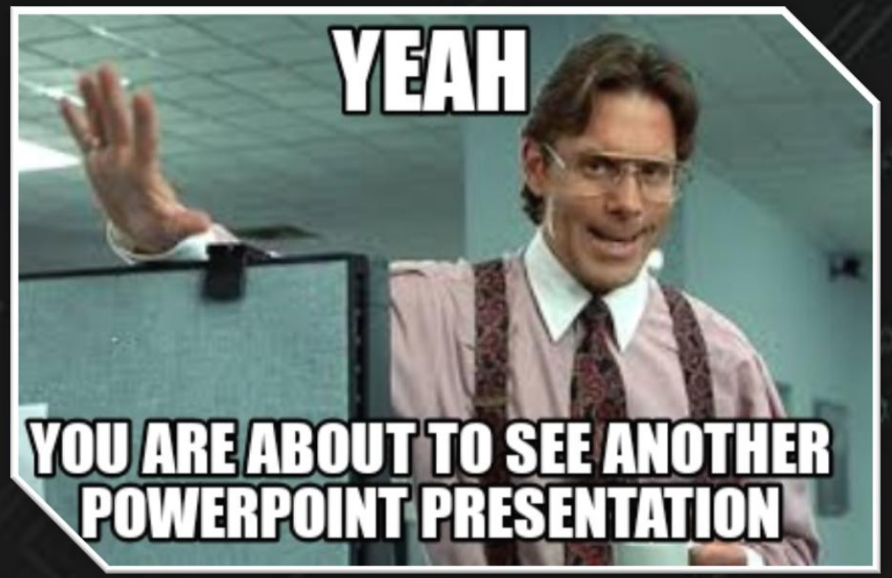
Procedural Items

- Take meaningful notes
- All materials on my GitHub
  - github.com/watson0x90
- Socials Handle:
  - Watson0x90
- Don’t worry
  - Lots of slides
  - Demos
  - Just enjoy!

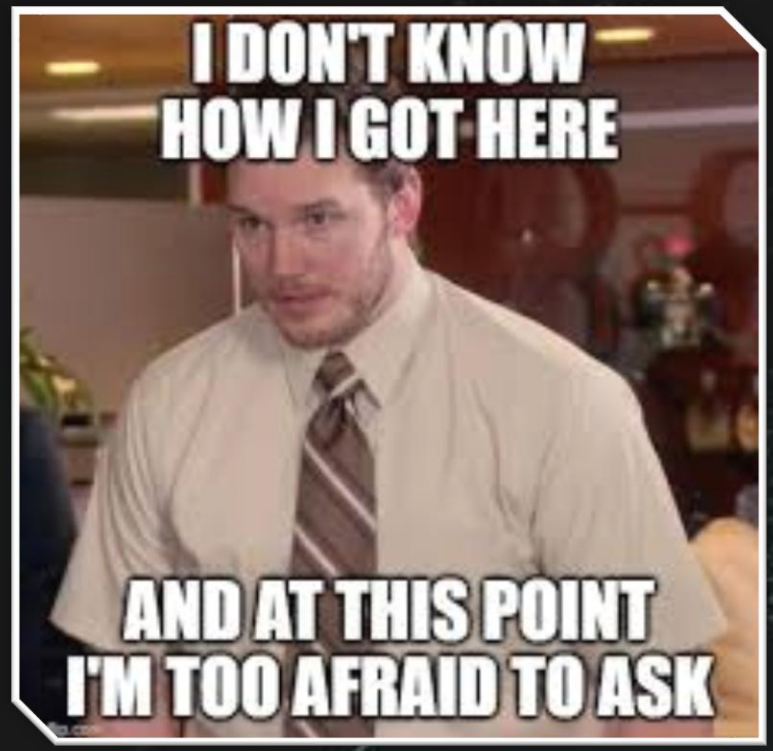


Overview

- How I got here
- Windows API Hooking
- The .NET Obstacle
- Managed vs Unmanaged Code
- Intro to Frida & Fermion!
- DirtyLittleDotNetHooker
- Demos!
- The end? ...

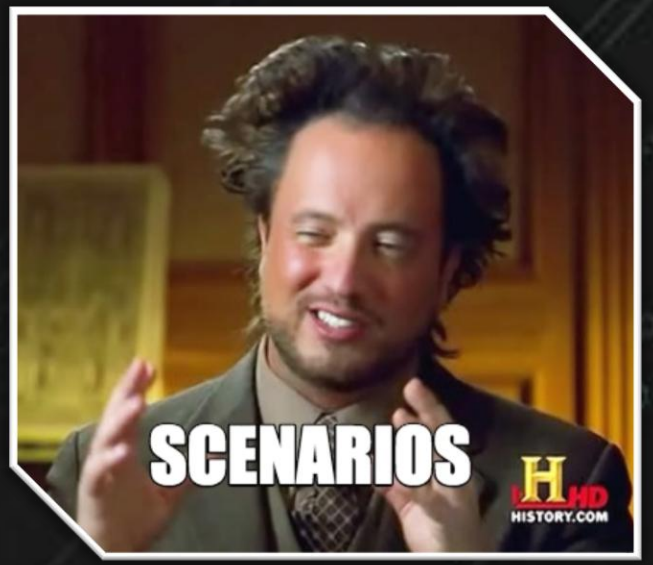


How I got here...

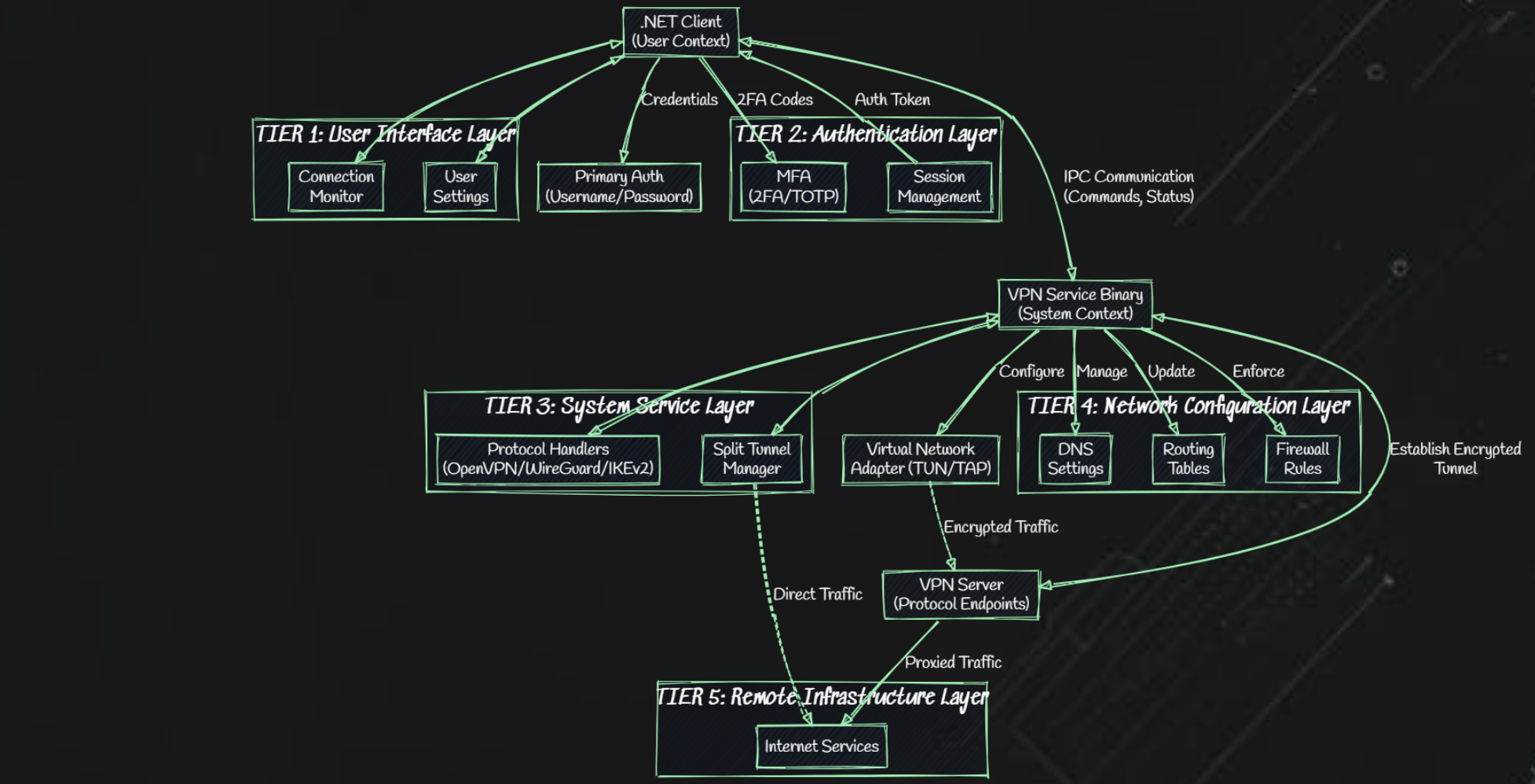


Scenario

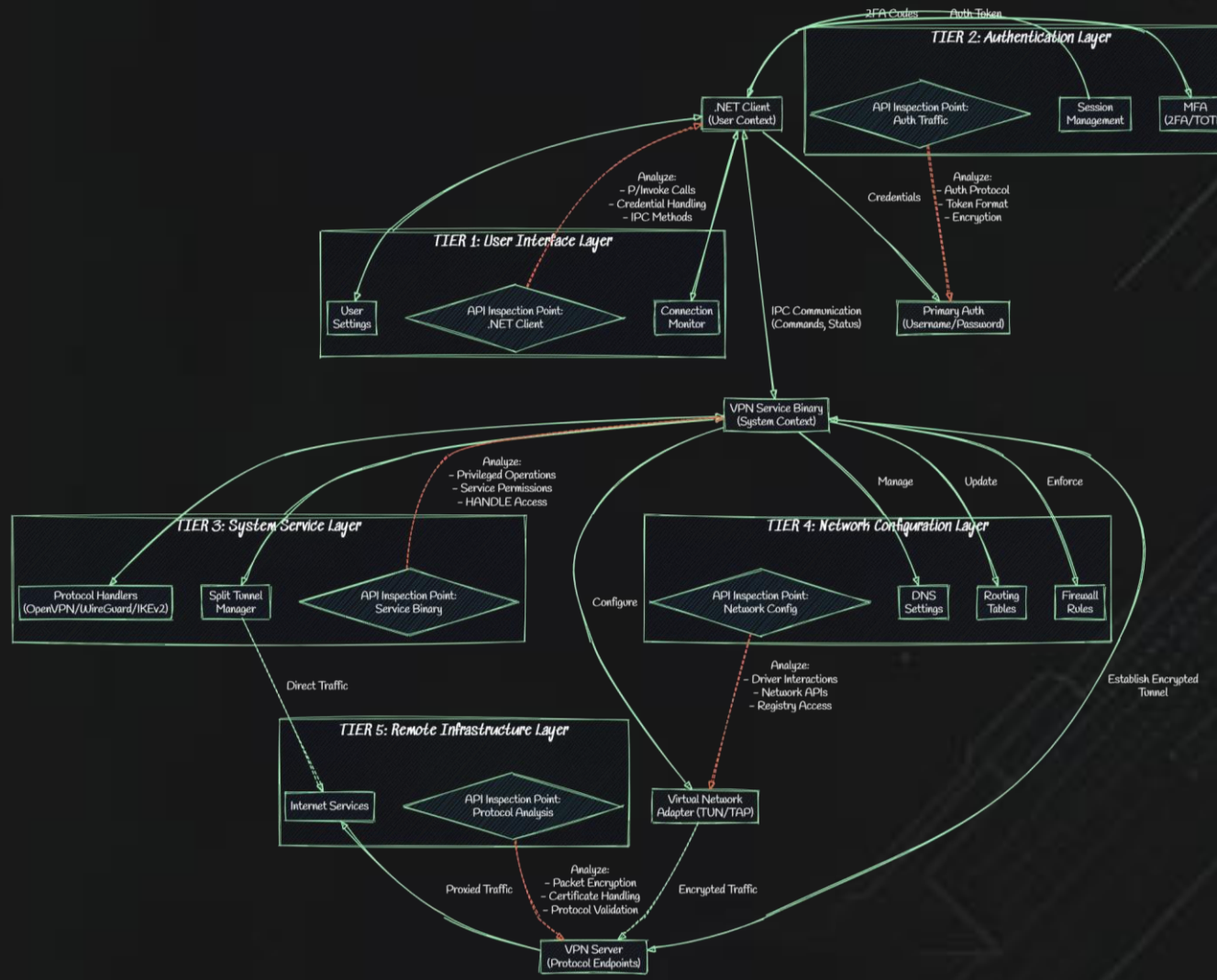
- Red Team Application Security Evaluation
  - Goals
    - Identify security vulnerabilities and weaknesses
    - Evaluate the effectiveness of the product
    - Simulate realistic attacker scenarios
    - Assess software resilience to targeted attacks
    - Provide actionable recommendations for improvement
    - Enhance overall software security posture
  - What was the product?
    - VPN Application
      - Service Binary
      - Client Application (.NET App)



Rough VPN Architecture – Tiered Layout



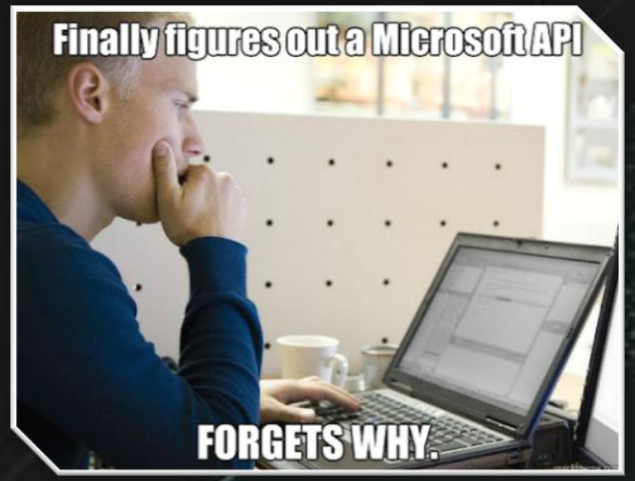
API Hooking Testing Strategy



API Hooking in Security Testing

- Insight into Application Behavior
  - Understand internal logic and data flows.
  - Reveal hidden or undocumented functionalities.
- Identify Vulnerabilities
  - Detect unsafe API calls and insecure data handling.
  - Identify weaknesses that could potentially lead to exploitation.
- Manipulate Application Behavior
  - Simulate attacker-controlled input and responses.
  - Test application resilience against unexpected scenarios.
- Monitor and Intercept Sensitive Operations
  - Track critical security-relevant calls
    - (authentication, encryption, etc.).
  - Analyze data handling during sensitive operations.
- Bypass or Assess Security Controls
  - Evaluate the effectiveness of the implemented security measures.
  - Demonstrate bypass techniques to inform defensive improvements.

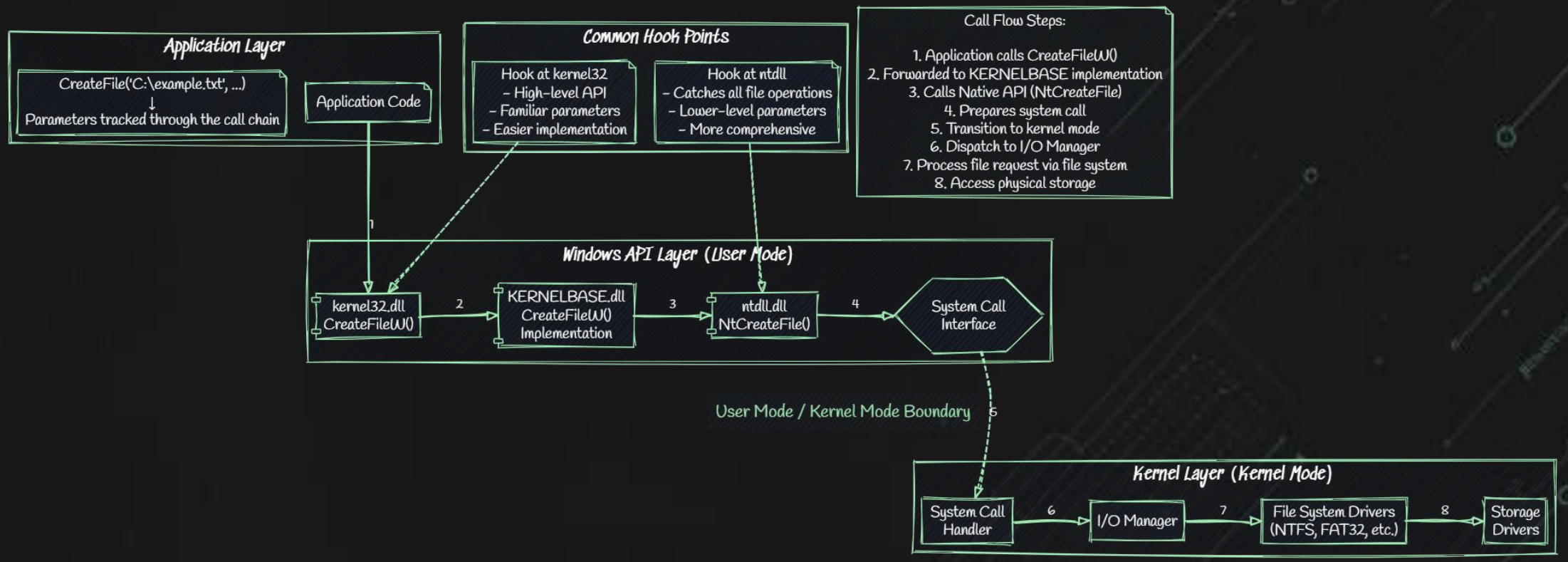
Windows API Hooking



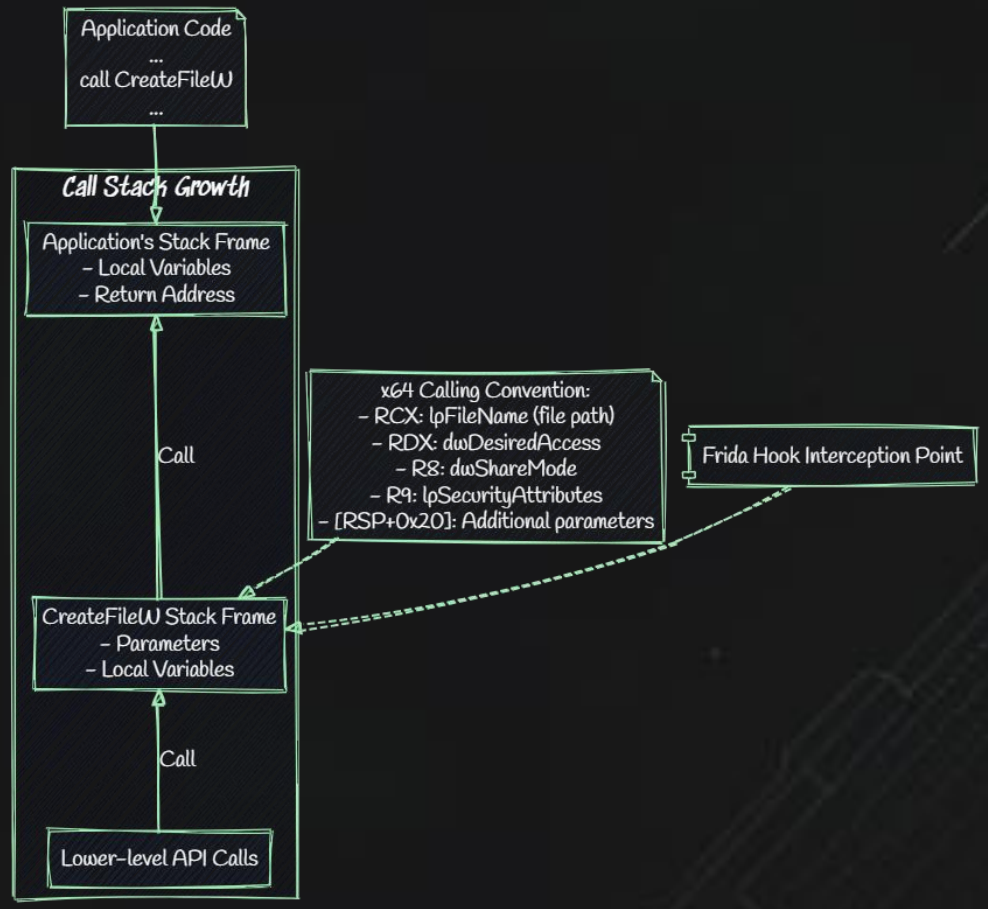
Windows API Hooking – The Why

- Layered Architecture:
  - Windows API calls traverse multiple layers before reaching kernel
- User Mode Layers:
  - Application → kernel32.dll → KERNELBASE.dll → ntdll.dll
- Mode Transition:
  - System call interface bridges user and kernel modes
- Kernel Mode Layers:
  - System call handler → I/O Manager → File system drivers → Storage drivers
- Strategic Hook Points:
  - kernel32.dll (high-level, familiar parameters) vs. ntdll.dll (comprehensive, catches all operations)

Windows API Hooking



Windows API Hooking



Windows API Hooking Demo

- Windows API Hooking using Frida
  - More about Frida later
- Hooking:
  - CreateFileW
  - WriteFile

```
// CreateFileW function (Kernel32.dll)
// Creates or opens a file or I/O device.
// |--> file, filestream, directory, namedpipe, etc...

HANDLE CreateFileW(
  [in]           LPCWSTR              lpFileName,
  [in]           DWORD                dwDesiredAccess,
  [in]           DWORD                dwShareMode,
  [in, optional] LPSECURITY_ATTRIBUTES lpSecurityAttributes,
  [in]           DWORD                dwCreationDisposition,
  [in]           DWORD                dwFlagsAndAttributes,
  [in, optional] HANDLE               hTemplateFile
);
```

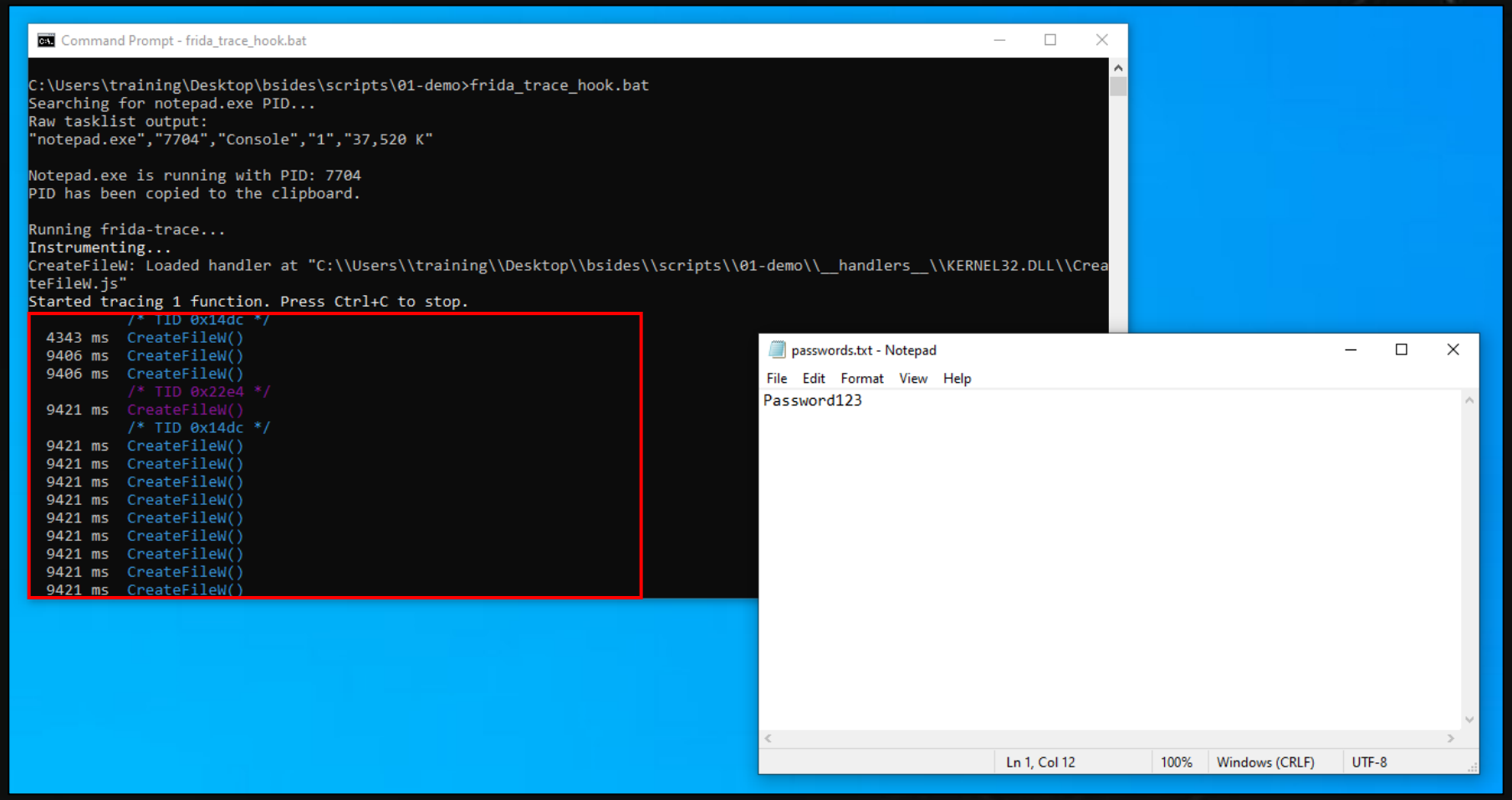
```
// WriteFile function (Kernel32.dll)
// Writes data to the specified file or input/output (I/O) device.
// |--> buffer -> file handle

BOOL WriteFile(
  [in]                HANDLE       hFile,
  [in]                LPCVOID      lpBuffer,
  [in]                DWORD        nNumberOfBytesToWrite,
  [out, optional]     LPDWORD      lpNumberOfBytesWritten,
  [in, out, optional] LPOVERLAPPED lpOverlapped
);
```

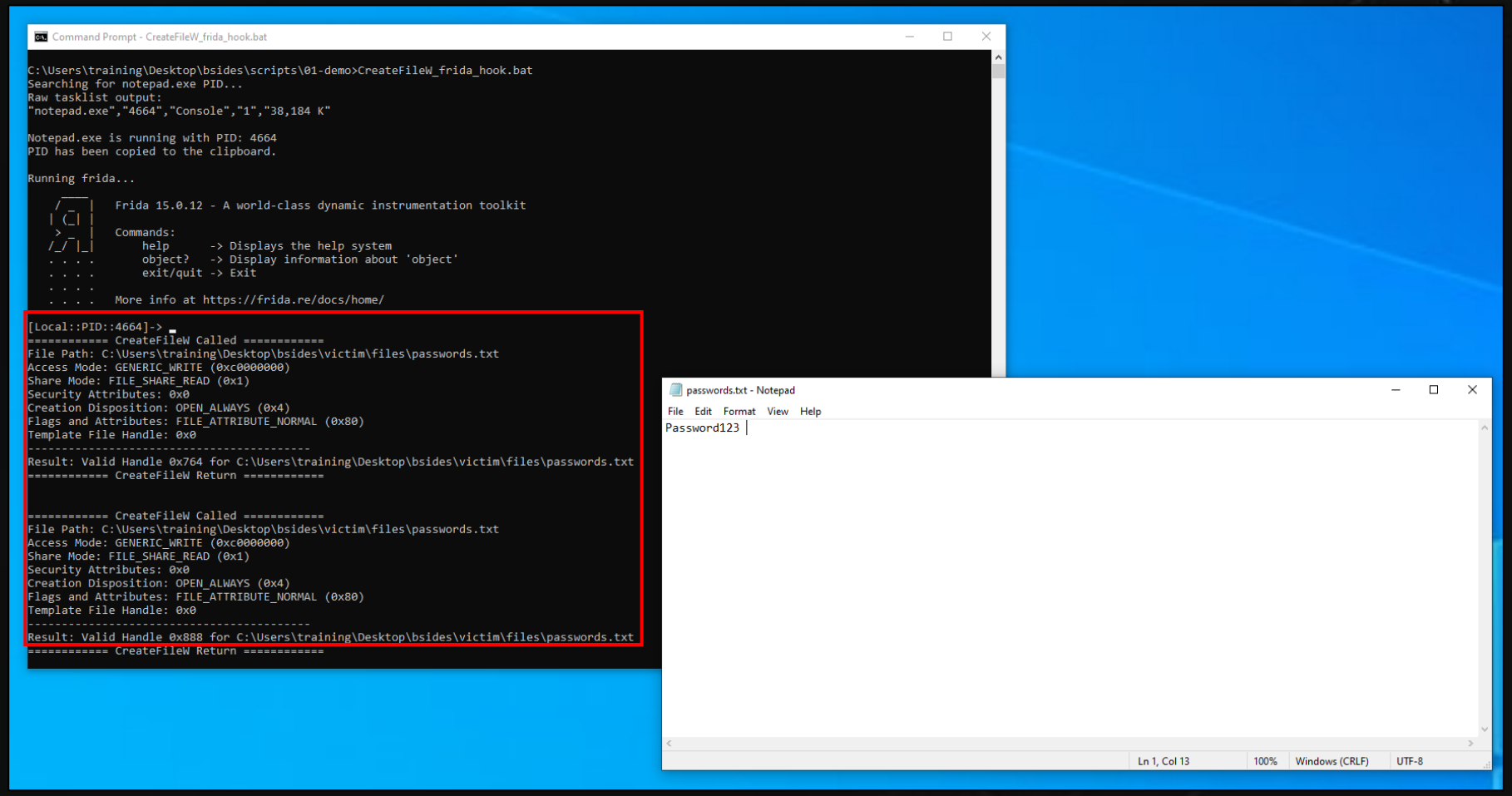
https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-createfilew
https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-writefile

Windows API Hooking Demo

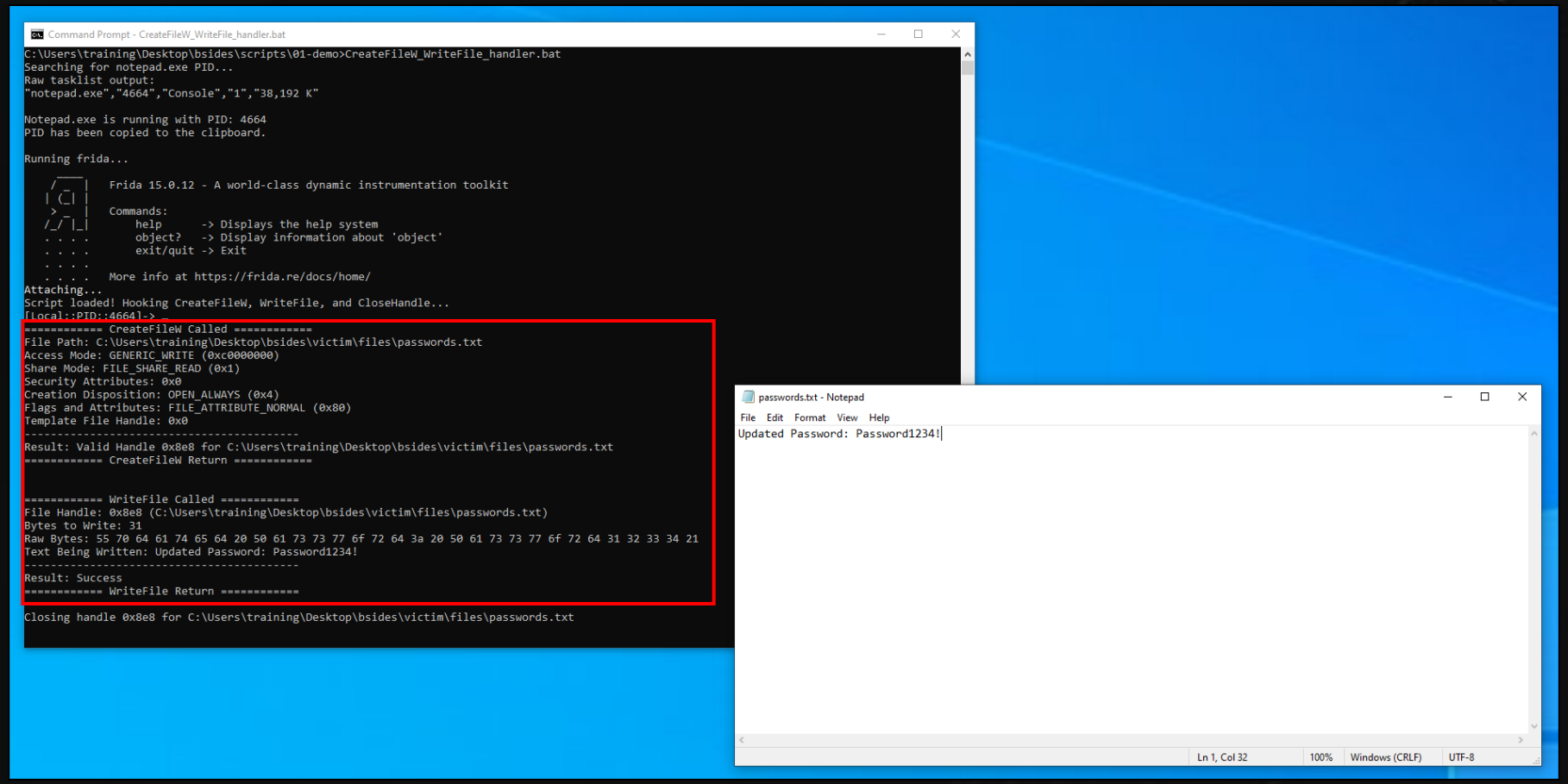
Windows API Hooking Demo - CreateFileW



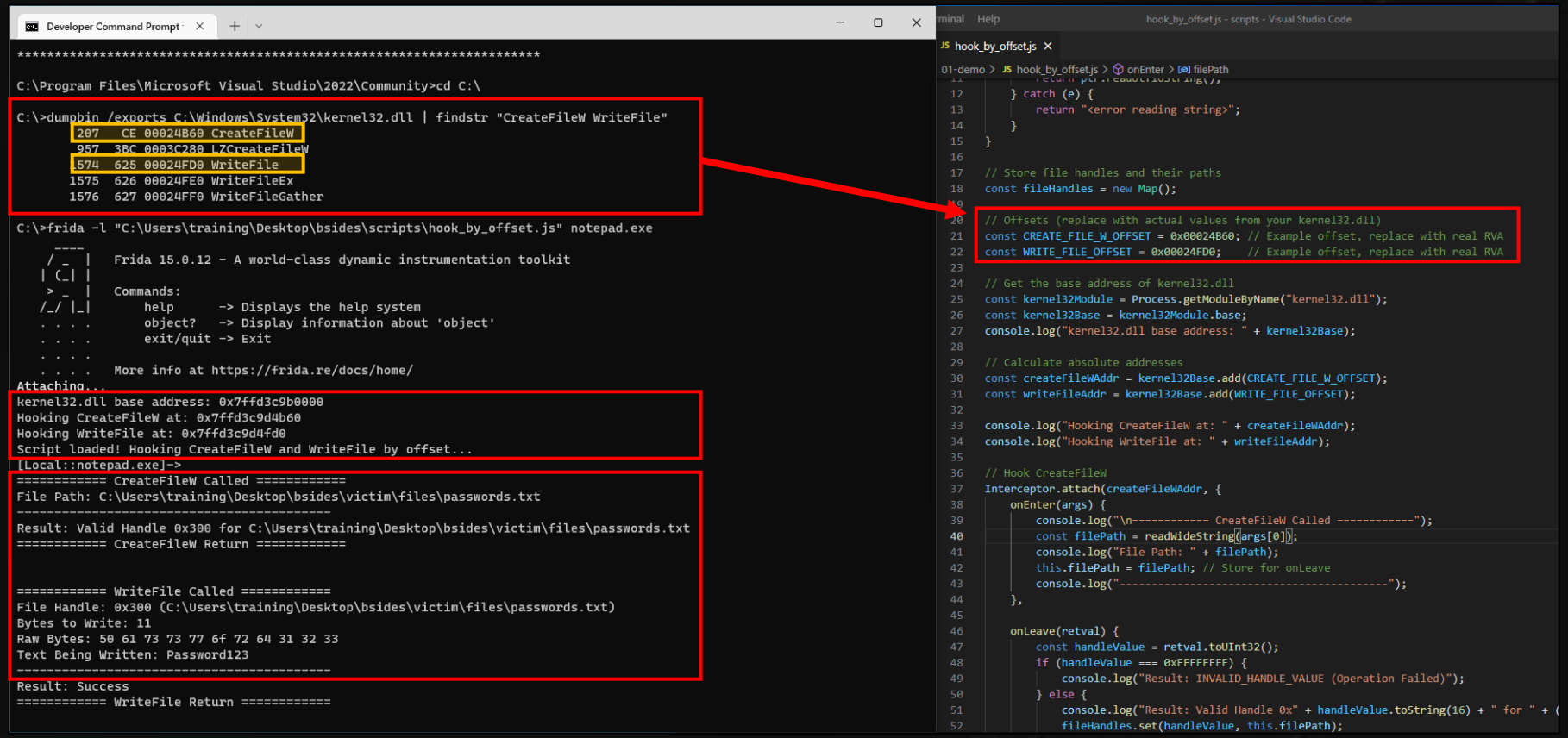
Windows API Hooking Demo - CreateFileW



Windows API Hooking Demo – CreateFileW, WriteFile

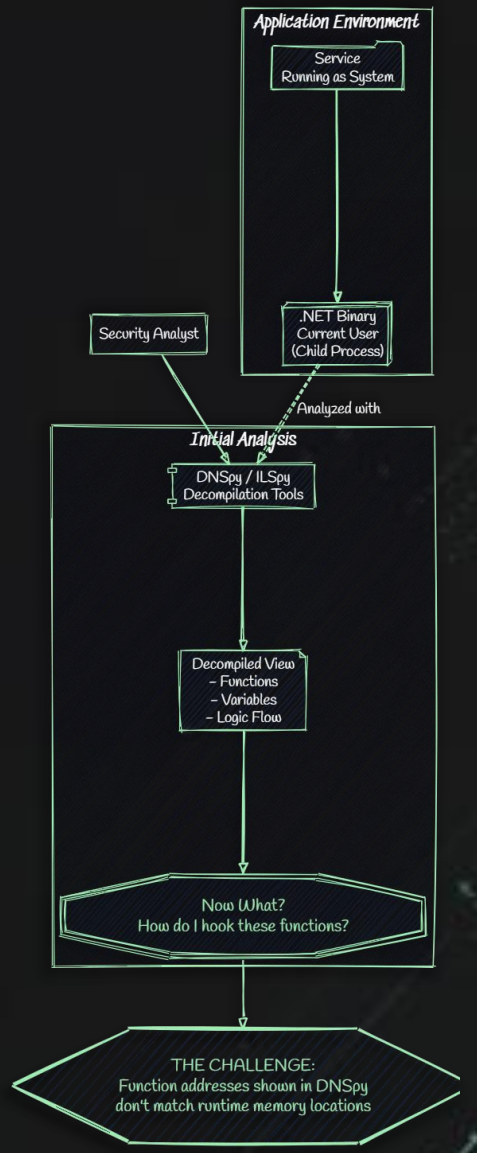


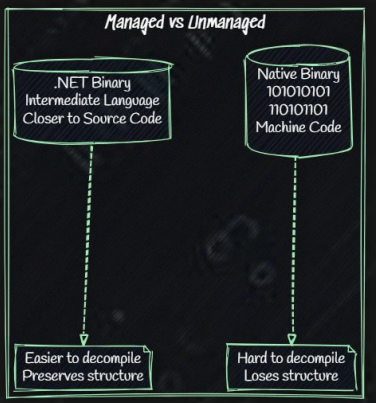
Windows API Hooking Demo – CreateFileW, WriteFile



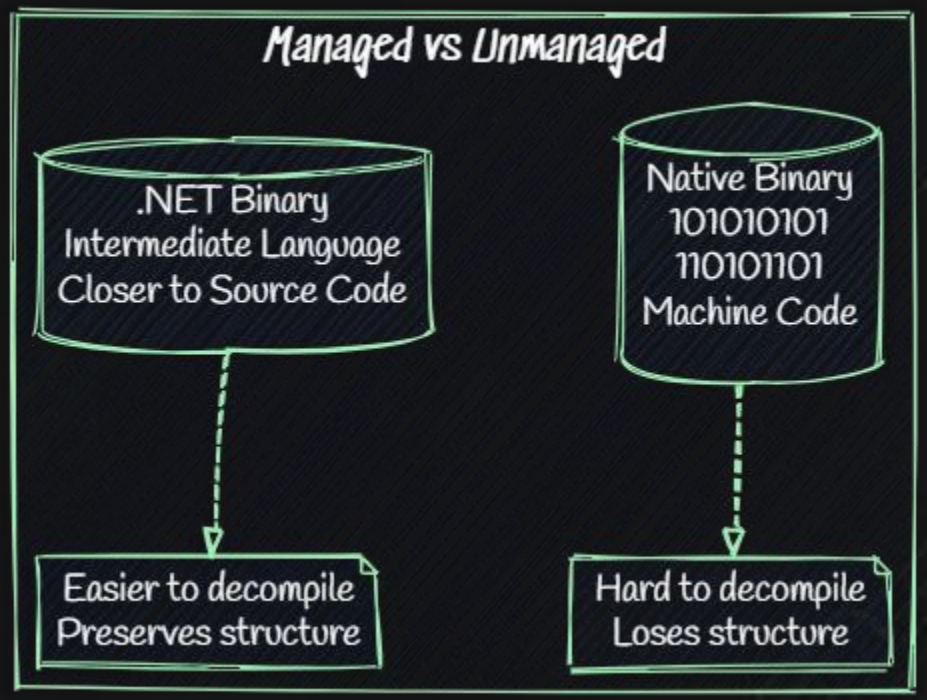
The .NET Obstacle

The 30,000-foot view

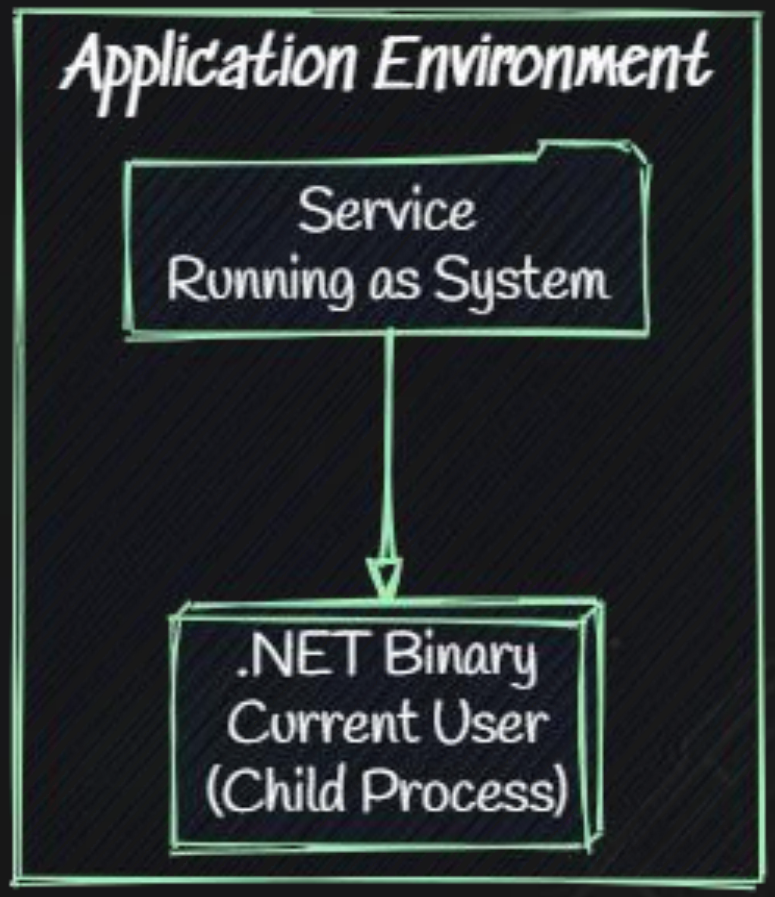




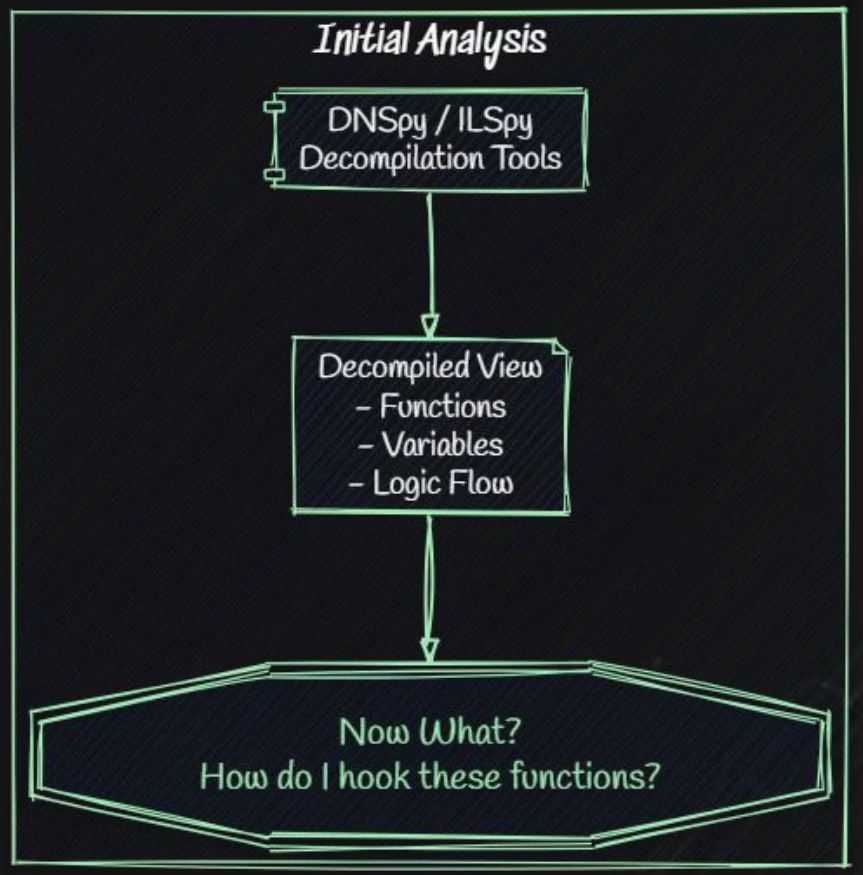
The .NET Obstacle – Managed vs Unmanaged



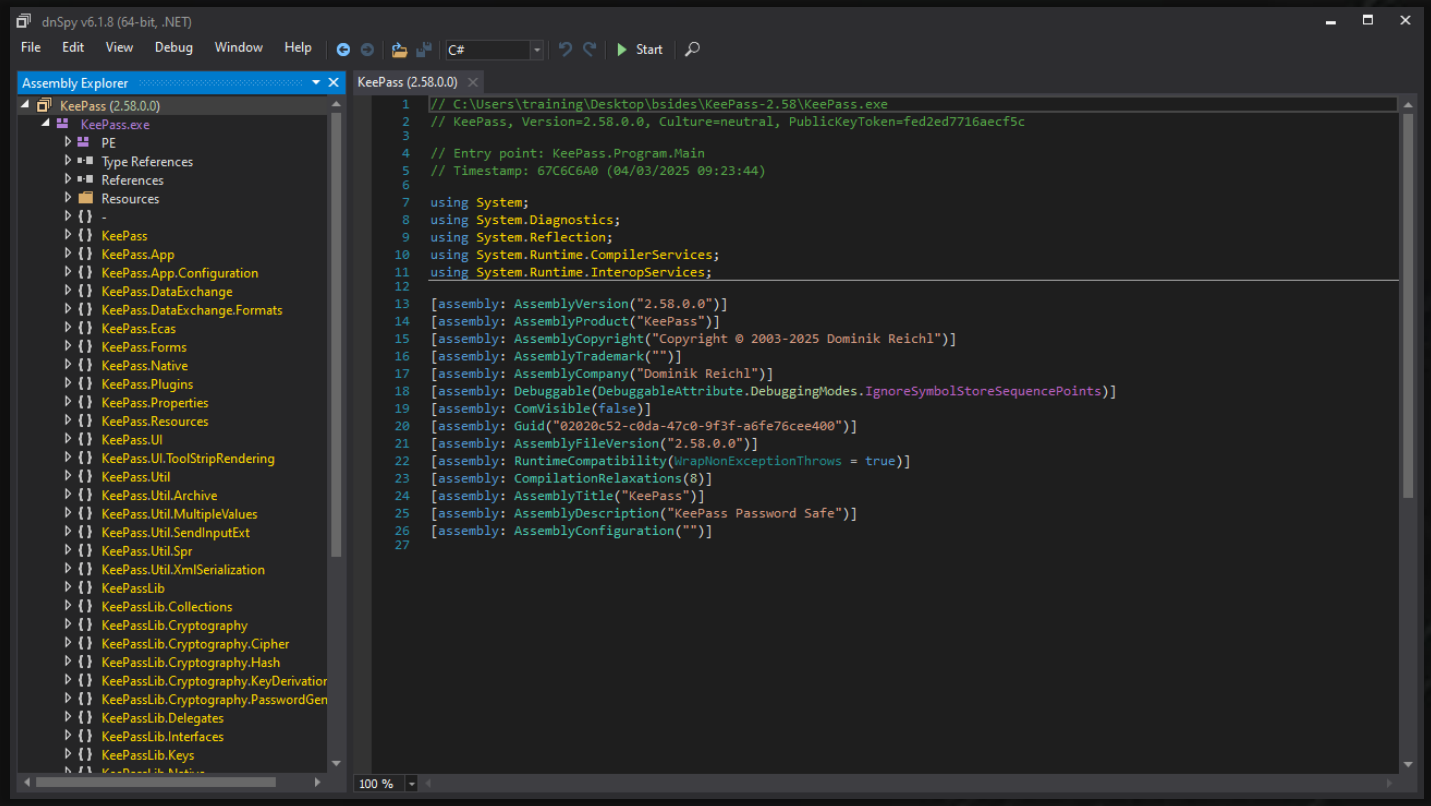
The .NET Obstacle – Application Environment



The .NET Obstacle – Initial Analysis

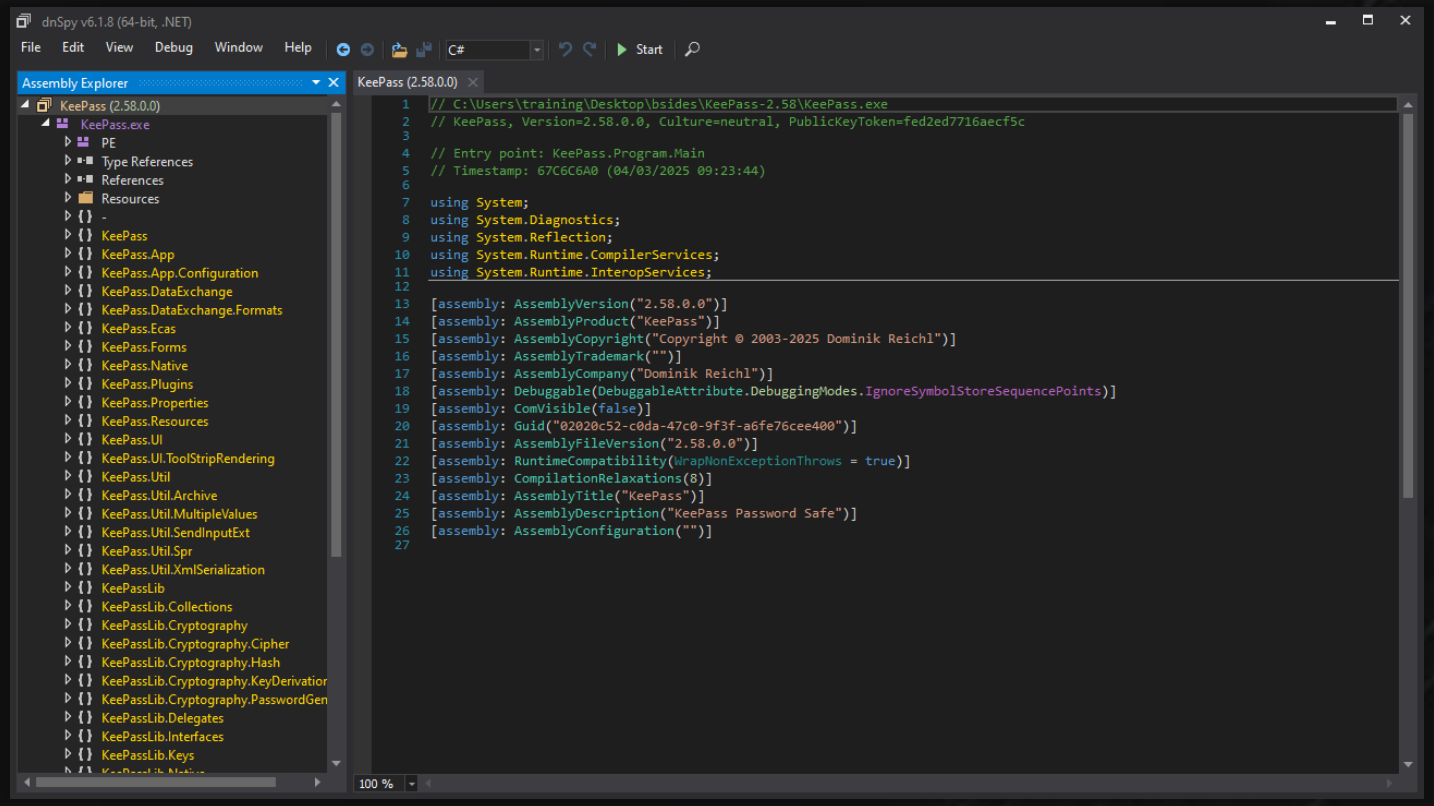


The .NET Obstacle - KeePass DNSpy Example



https://github.com/dnSpyEx/dnSpy

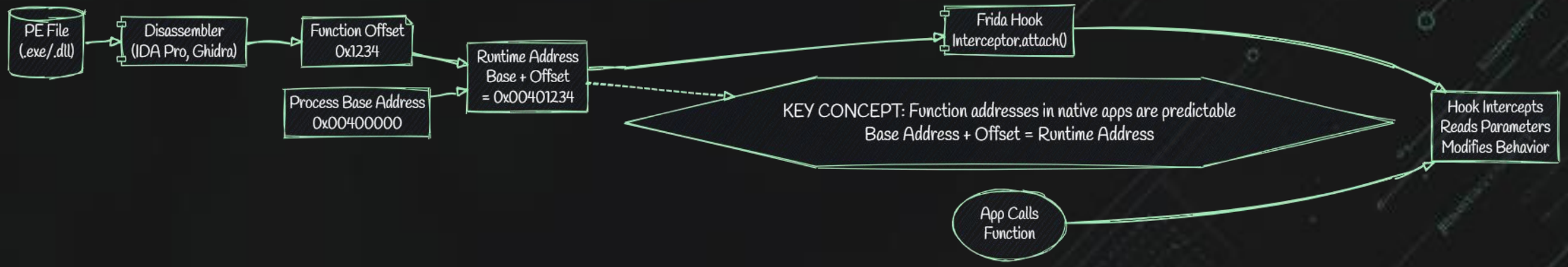
The .NET Obstacle - KeePass DNSpy Example



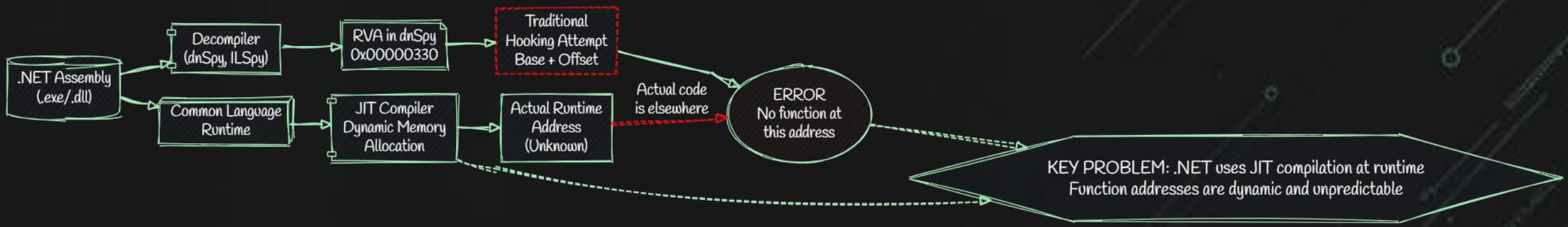
The .NET Obstacle - KeePass DNSpy Example

```
258
259         // Token: 0x06001508 RID: 5384 RVA: 0x00095310 File Offset: 0x00093510
260         [STAThread]
261         public static void Main(string[] args)
262         {
263             try
264             {
265                 Program.MainPriv(args);
266             }
267             catch (Exception ex)
268             {
269                 Program.ShowFatal(ex);
270             }
271             finally
272             {
273                 try
274                 {
275                     Program.CommonTerminate();
276                 }
277                 catch (Exception)
278                 {
279                 }
280                 GC.KeepAlive(Program.g_mGlobalNotify);
281             }
282         }
283
284         // Token: 0x06001509 RID: 5385 RVA: 0x000953D4 File Offset: 0x000935D4
285         private static void MainPriv(string[] args)
286         {
287             Program.g_bMain = true;
288             Program.g_bDesignMode = false;
289             Program.InitAppContext();
290             Program.g_cmdLineArgs = new CommandLineArgs(args);
291             if (Program.g_cmdLineArgs[AppDefs.CommandLineOptions.Debug] != null)
292             {
293                 PwDefs.DebugMode = true;
294             }
295             string text = Program.g_cmdLineArgs[AppDefs.CommandLineOptions.WorkaroundDisable];
296             if (!string.IsNullOrEmpty(text))
297             {
298                 MonoWorkarounds.SetEnabled(text, false);
299             }
300             text = Program.g_cmdLineArgs[AppDefs.CommandLineOptions.WorkaroundEnable];
301             if (!string.IsNullOrEmpty(text))
302             {
303                 MonoWorkarounds.SetEnabled(text, true);
304             }
305             try
306             {
307                 DpiUtil.ConfigureProcess();
```

The .NET Obstacle – Traditional Hooking



The .NET Obstacle – Traditional Hooking (no worky)



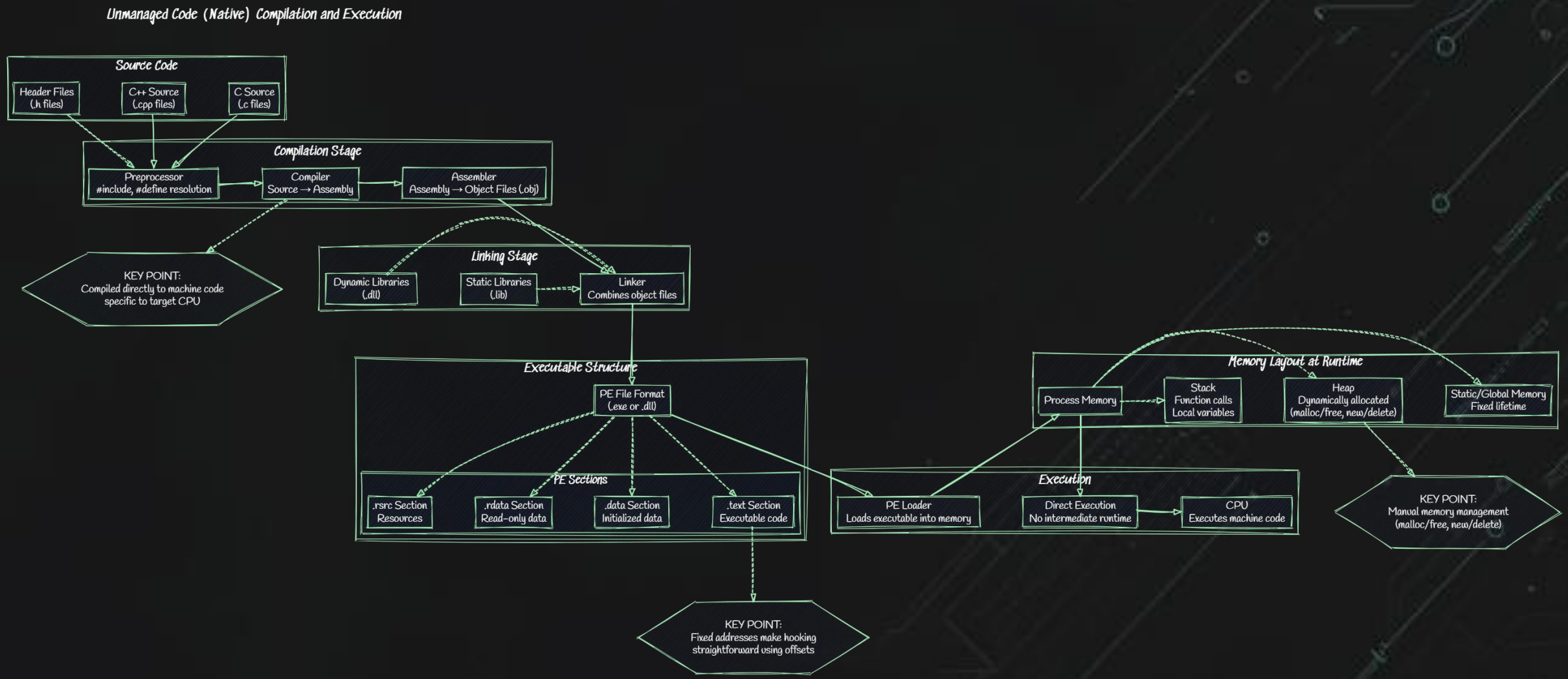
Unmanaged vs Managed Code

Oh, the joy!

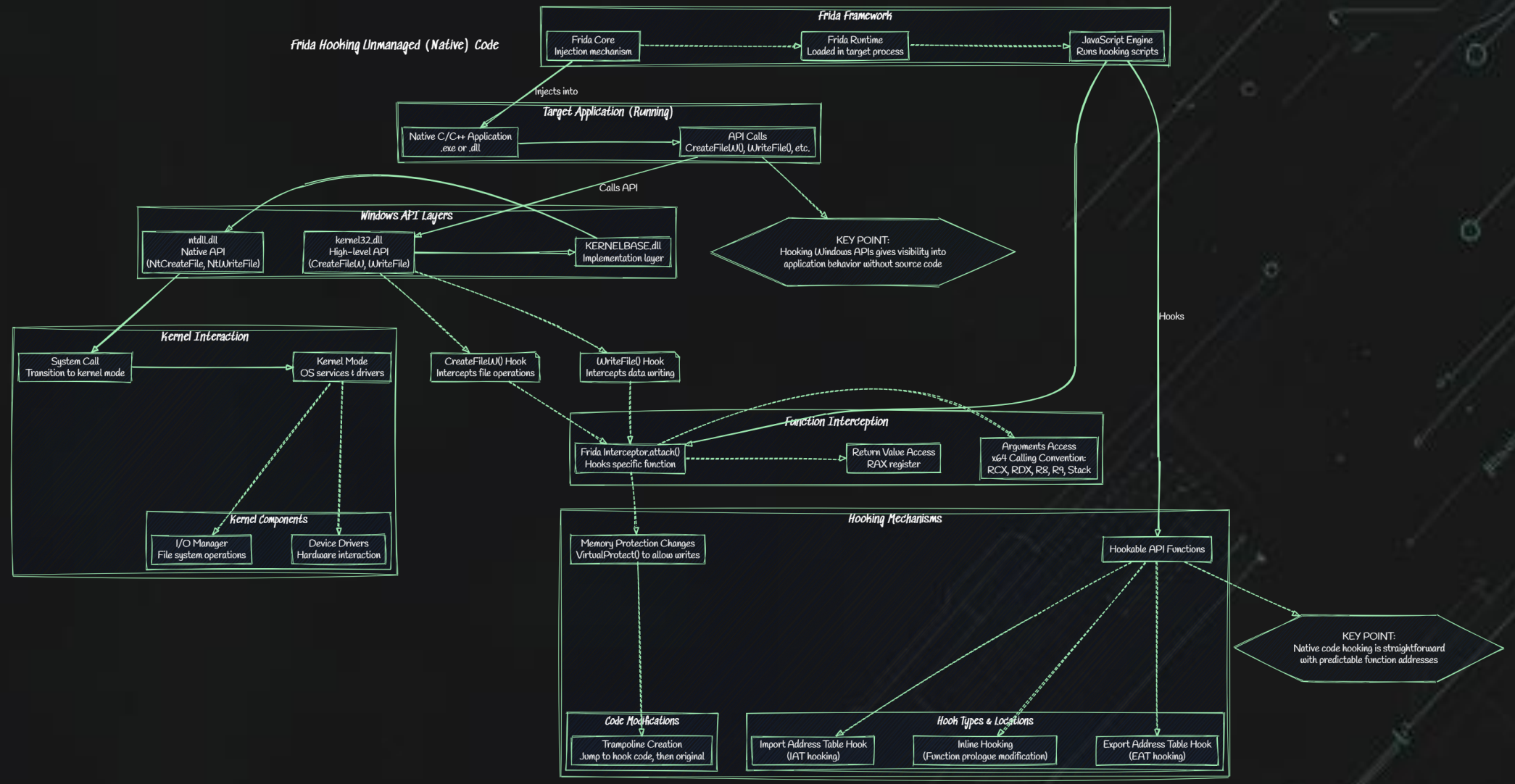
Unmanaged Code – Overview

- Definition:
  - Code that compiles directly to machine code and executes directly on the CPU
- Compilation Path:
  - Source code → Assembly → Object files → Executable with machine code
- Key Characteristics:
  - Static memory addresses determined at compile/link time
  - Manual memory management (malloc/free, new/delete)
  - Predictable function locations (Base + Offset)
  - Same code path on every execution
- API Interactions:
  - Direct calls to Windows APIs at known addresses
  - Straightforward to hook using traditional techniques
  - Predictable calling conventions and memory layout

Unmanaged Code – Compilation & Execution



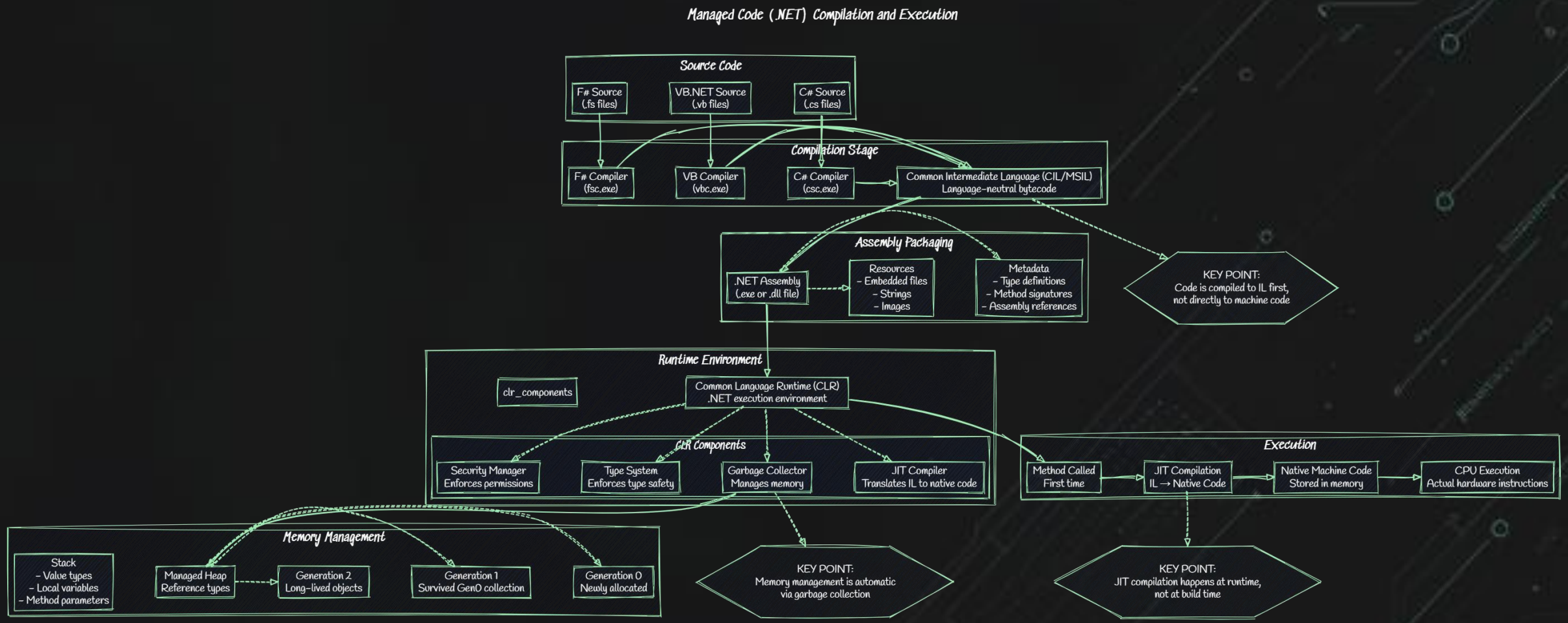
Unmanaged Code – Hooking Flow



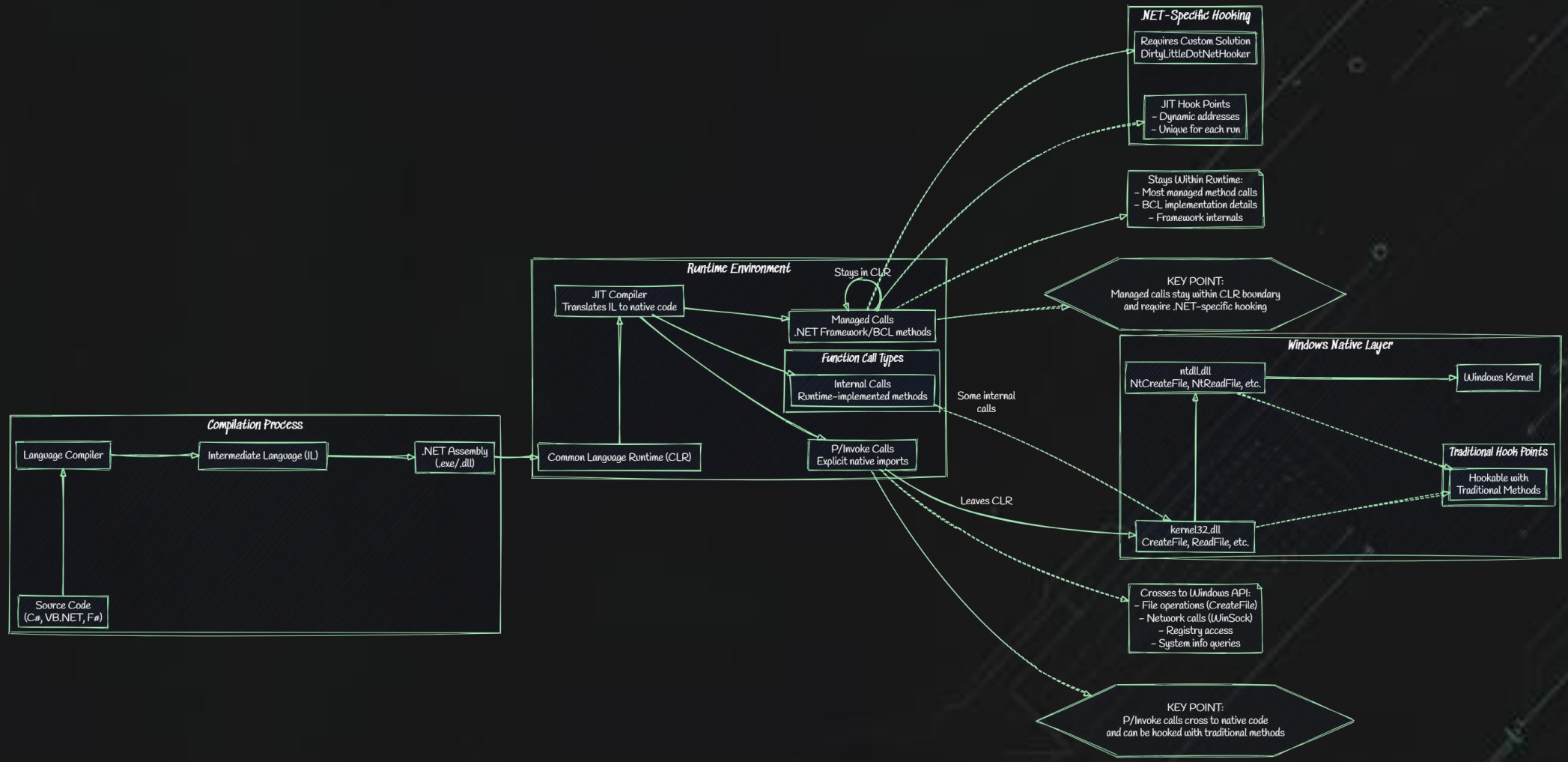
Managed Code

- **Definition**:
  - Code that executes within a runtime environment (CLR) that manages memory, security, and execution
- **Compilation Path:**
  - Source code → Intermediate Language (IL) → JIT compilation to native code at runtime
- **Key Characteristics:**
  - Dynamic memory addresses determined at runtime through JIT compilation
  - Memory management is handled automatically via garbage collection
  - Rich metadata preserved in assemblies
  - Function locations can change between application runs
- **API Interactions:**
  - Some calls stay entirely within CLR (require specialized hooking)
  - Others cross to Windows API through P/Invoke (can use traditional hooking)

Managed Code - Compilation & Execution



Managed Code - Compilation & Execution



Intro to Frida

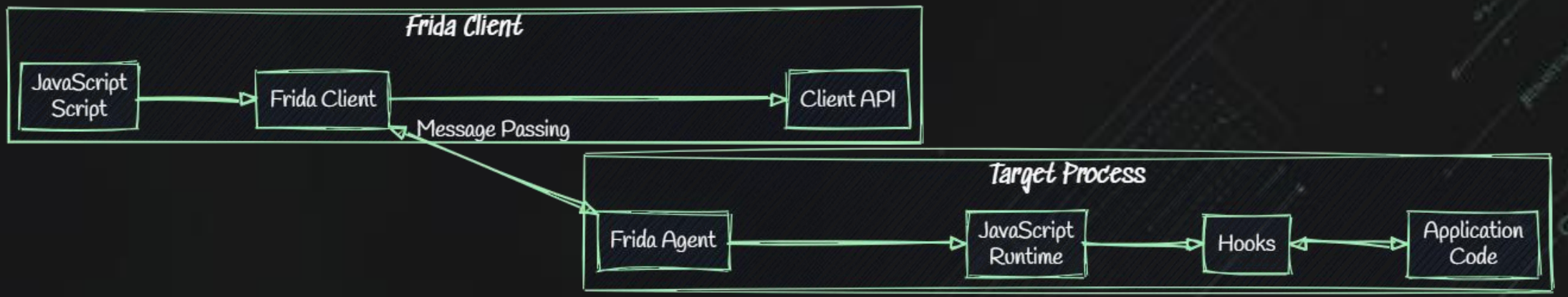
Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers.

Frida

- Website:
  - https://frida.re
- Dynamic instrumentation toolkit
- Inject JavaScript into native apps
  - Gum & GumJS
    - Inline hooking
    - Stealthy code tracing
    - Memory monitoring
- Cross-platform
  - Windows, macOS, Linux, iOS, Android
- Example Use Cases
  - Malware Analysis:
    - Observe suspicious API calls or decrypt runtime payloads.
  - Penetration Testing/Red Teaming:
    - Modify behavior, bypass authentication, or log credentials.
  - Reverse Engineering:
    - Identify function arguments and return values dynamically.

Frida – How does it work?

- Agent: Injected into the target process
- Client: Controls the agent via API
- Message passing for communication
- JavaScript runtime inside the target process



Frida – Quick Reference

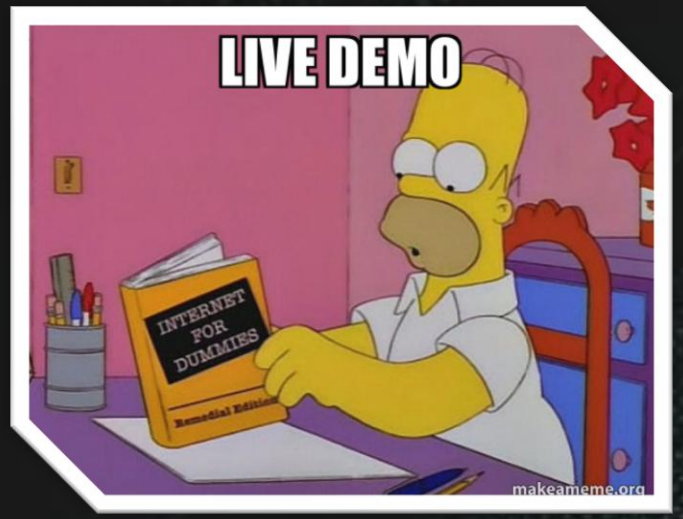
- Install
  - pip install frida frida-tools
  - frida - -version
- Commands
  - List Processes
    - frida-ps
    - frida-ps -Uai "*chrome*"
  - Trace API Calls
    - frida-trace -i "CreateFile*" notepad.exe
    - frida-trace -i "WSA*" -p <PID>
  - Inject JavaScript into Processes
    - frida -n notepad.exe -l script.js
    - frida -f calc.exe -l script.js --no-pause

- Script Example

```
Interceptor.attach(Module.getExportByName('kernel32.dll', 'CreateFileW'), {
  onEnter(args) {
    console.log('CreateFileW called with:', args[0].readUtf16String());
  },
  onLeave(retval) {
    console.log('CreateFileW returned:', retval);
  }
})
```

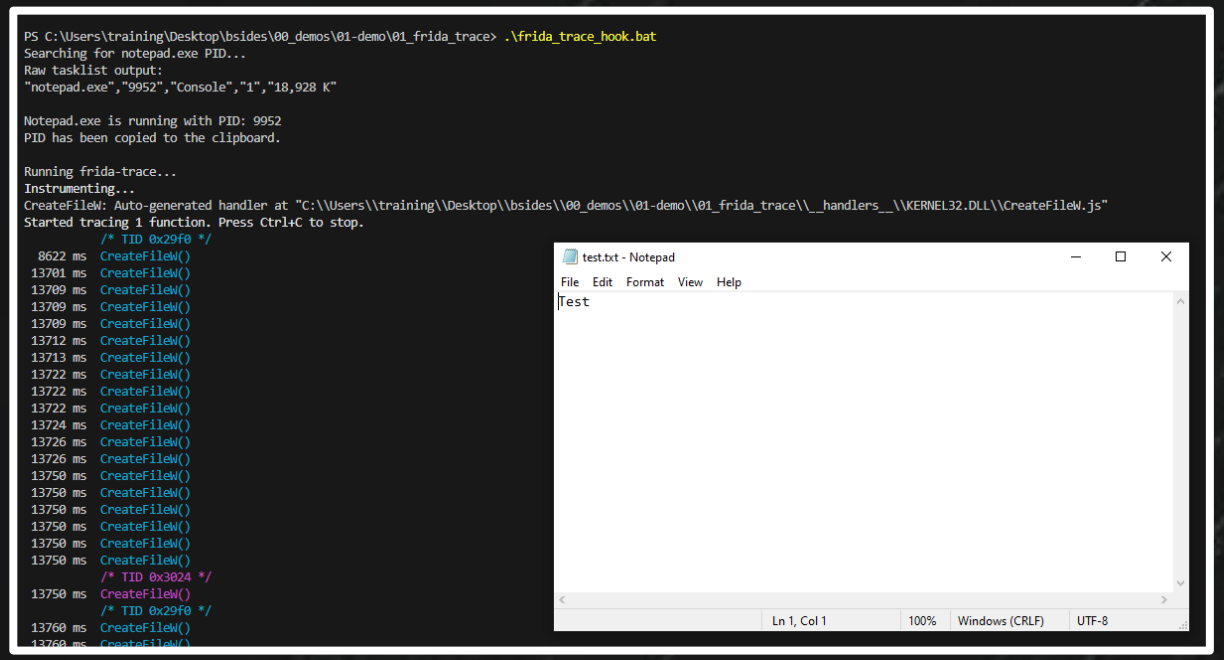
- Docs:
  - https://frida.re/docs/quickstart
  - https://frida.re/docs/presentations

Frida Demo



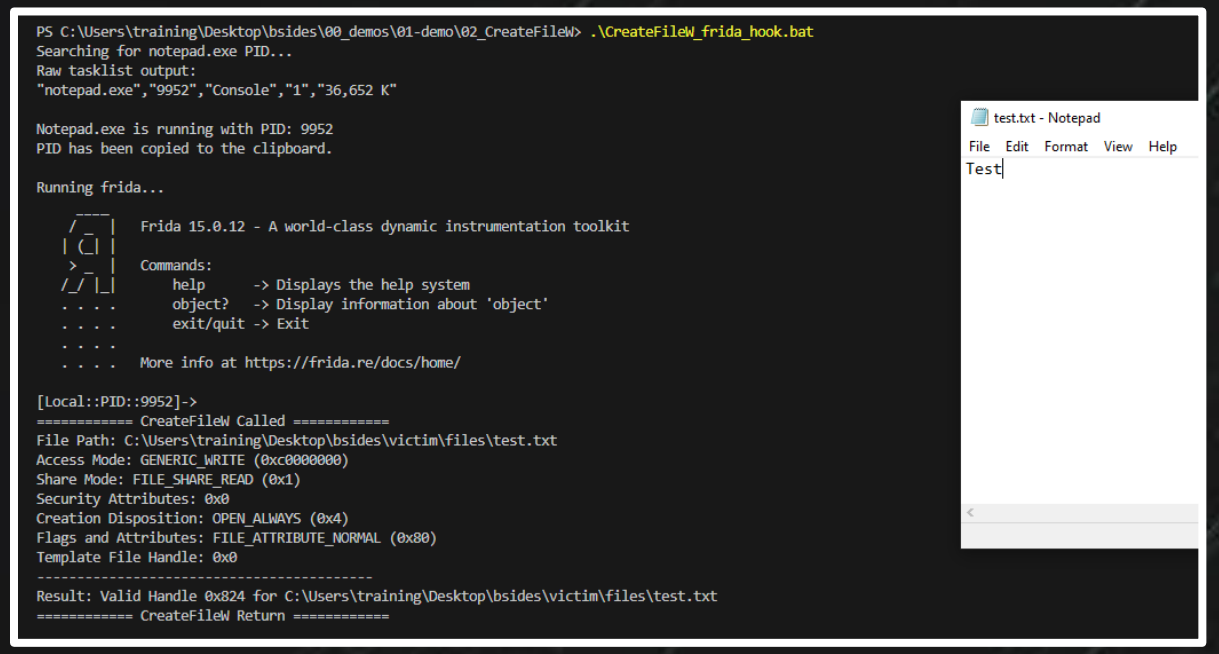
Frida – Demo Review

- Frida-Trace
  - frida-trace -i CreateFileW -X KERNELBASE.dll -p !PID!
- Simplified tool
- No control on output



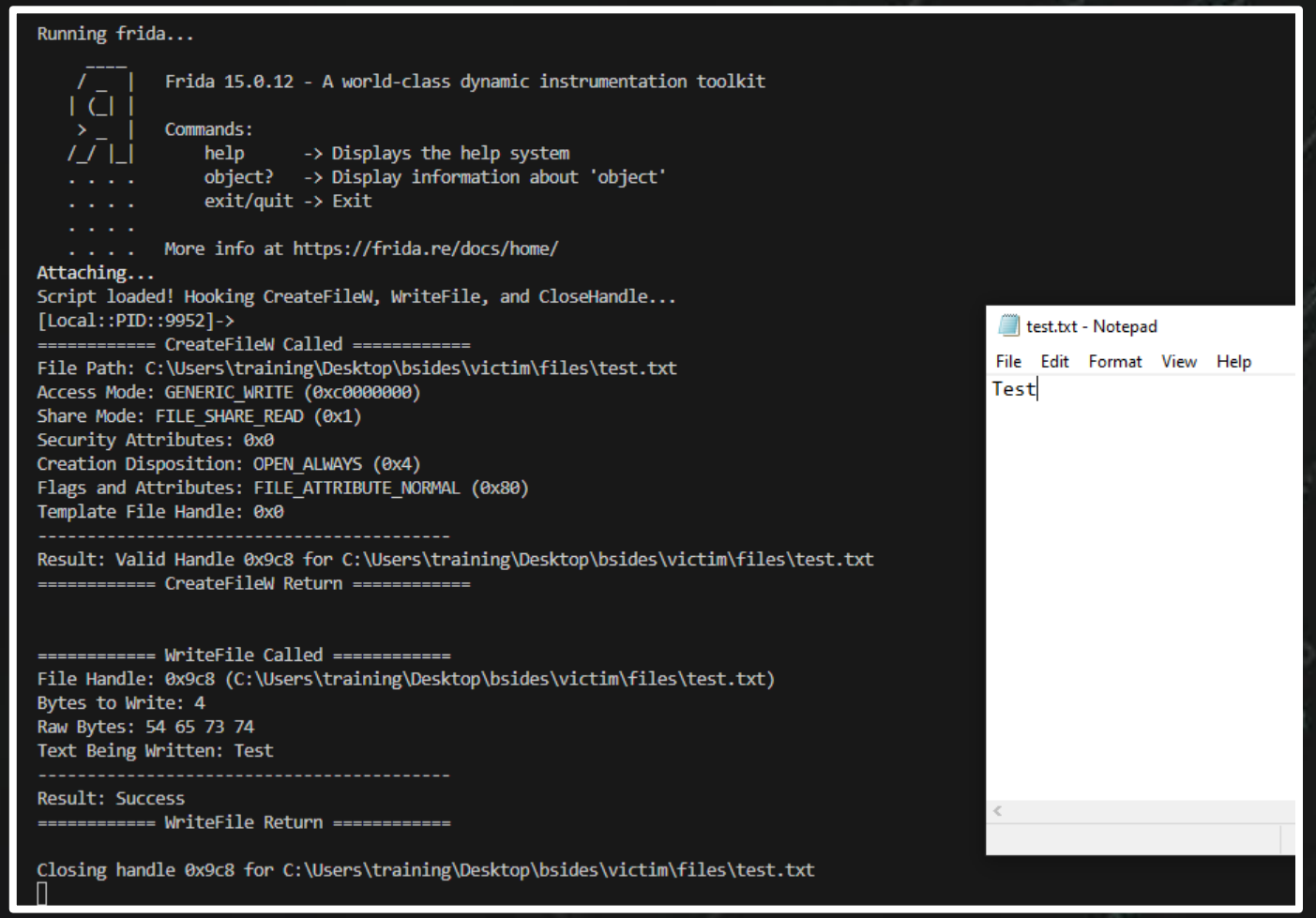
Frida – Demo Review

- Create script to hook CreateFileW and return information
- Interactively inject and run JavaScript
- Attach to or create a new process



Frida – Demo Review

- Hook multiple APIs
  - CreateFileW
  - WriteFile

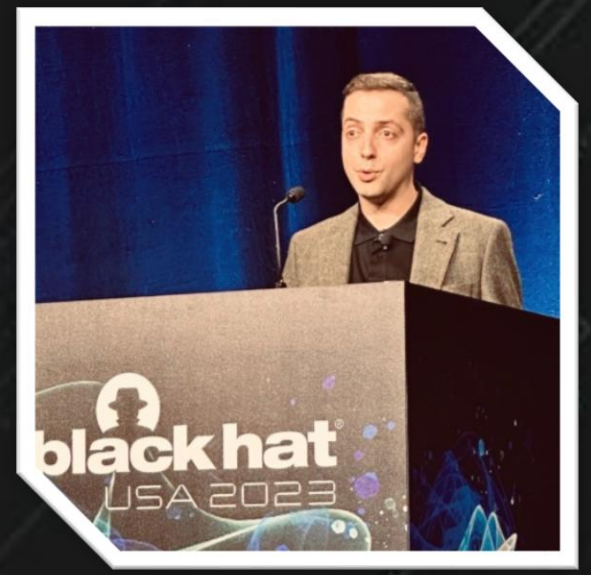


Intro to Fermion

Electron application that wraps
frida-node and monaco-editor

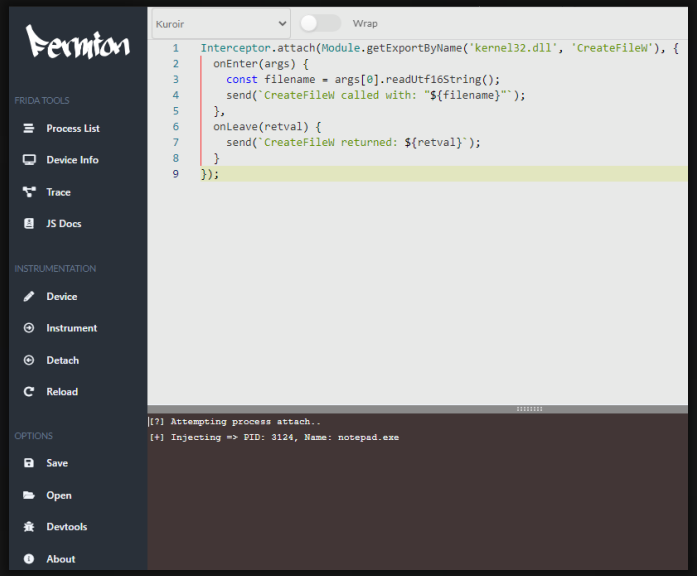
Fermion

- Author:
  - Ruben Boonen (b33f)
  - linkedin.com/in/rboonen
- Github:
  - github.com/FuzzySecurity/Fermion
- Description:
  - An electron application that wraps frida-node and monaco-editor.
- Training:
  - labs.calypso.pub
  - Windows Instrumentation With Frida



Fermion – Quick Reference

- Install
  - Download from GitHub
- Usage



- Script Example

```
Interceptor.attach(Module.getExportByName('kernel32.dll', 'CreateFileW'), {
  onEnter(args) {
    const filename = args[0].readUtf16String();
    send(`CreateFileW called with: "${filename}"`);
  },
  onLeave(retval) {
    send(`CreateFileW returned: ${retval}`);
  }
});
```

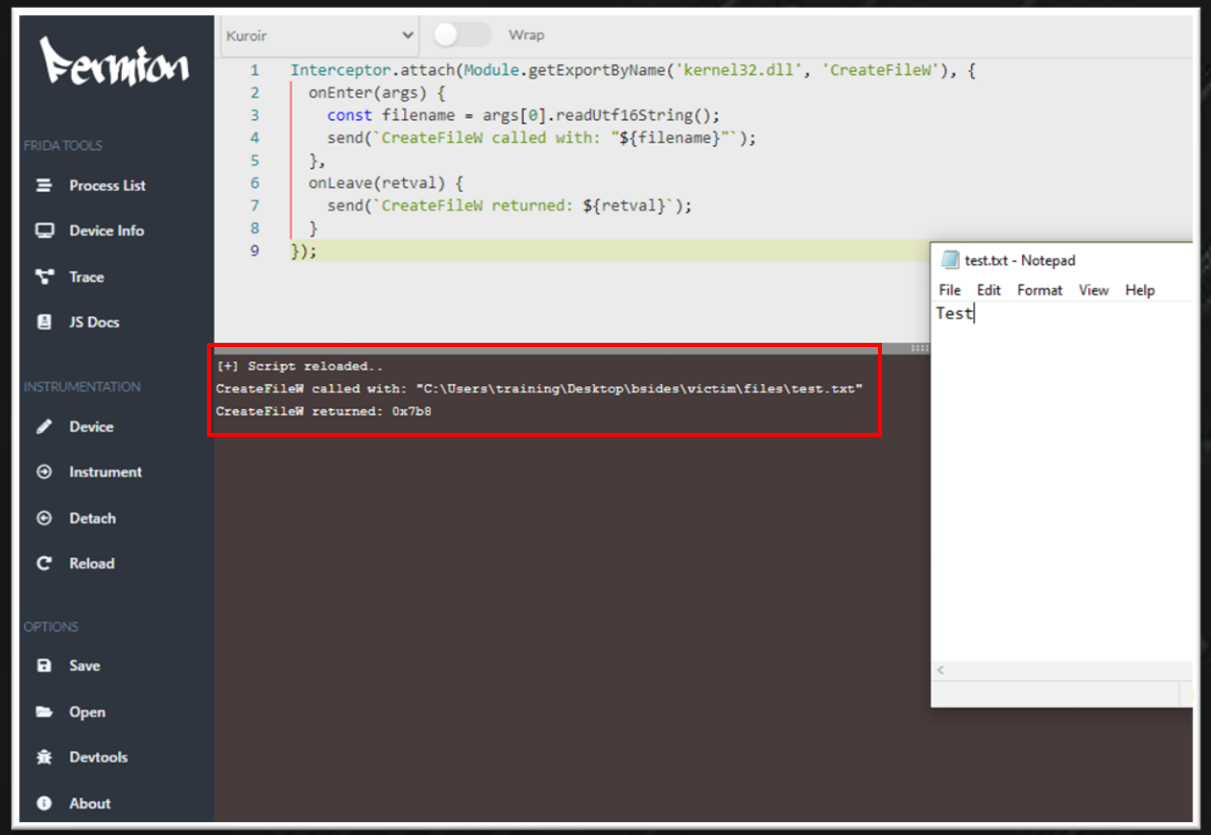
  - Uses send instead of console.log

https://github.com/FuzzySecurity/Fermion

Fermion Demo

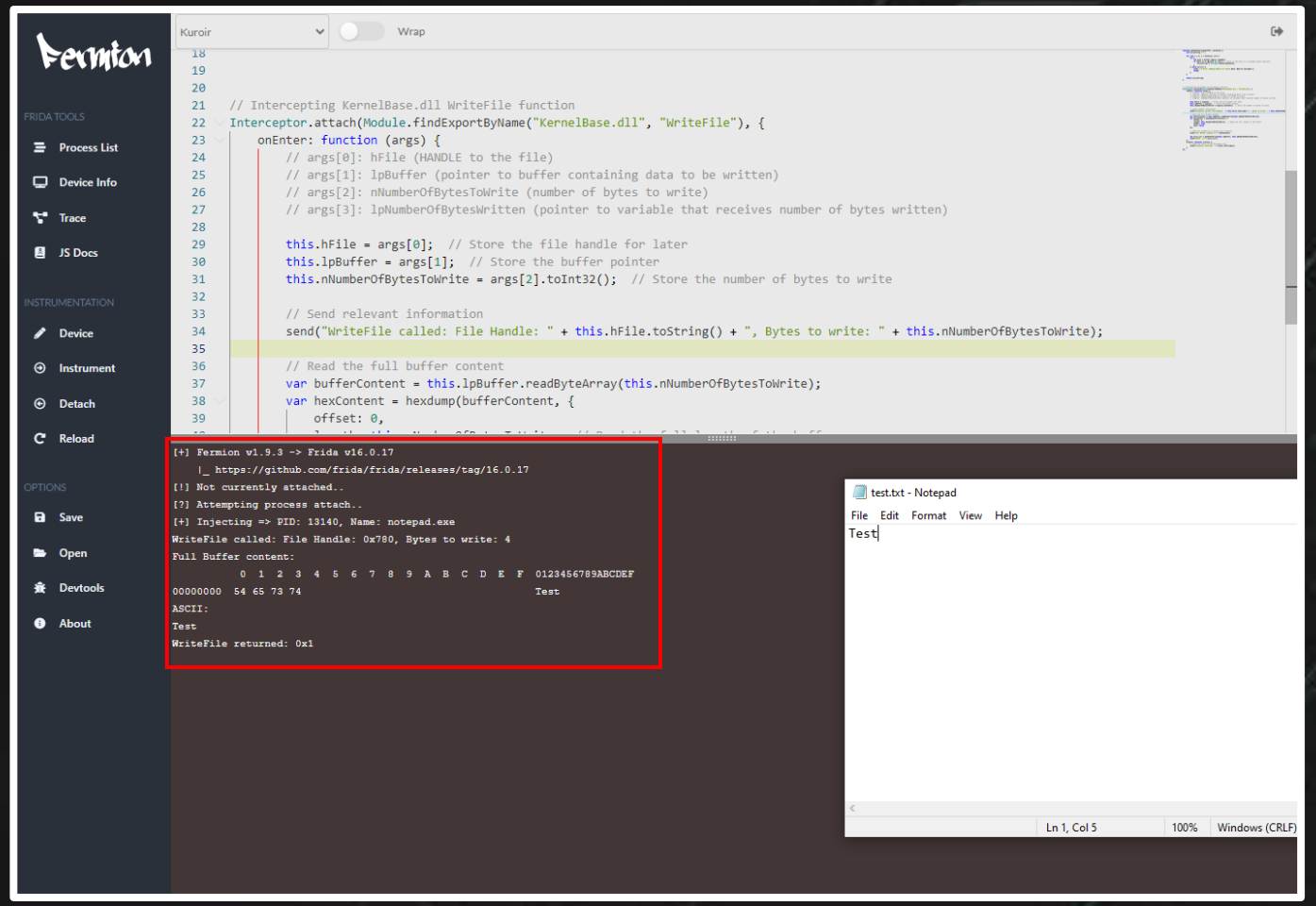
Fermion – Demo Review

- Hook
  - CreateFileW, Kernel32.dll
- Output
  - File handle info
  - Return Value



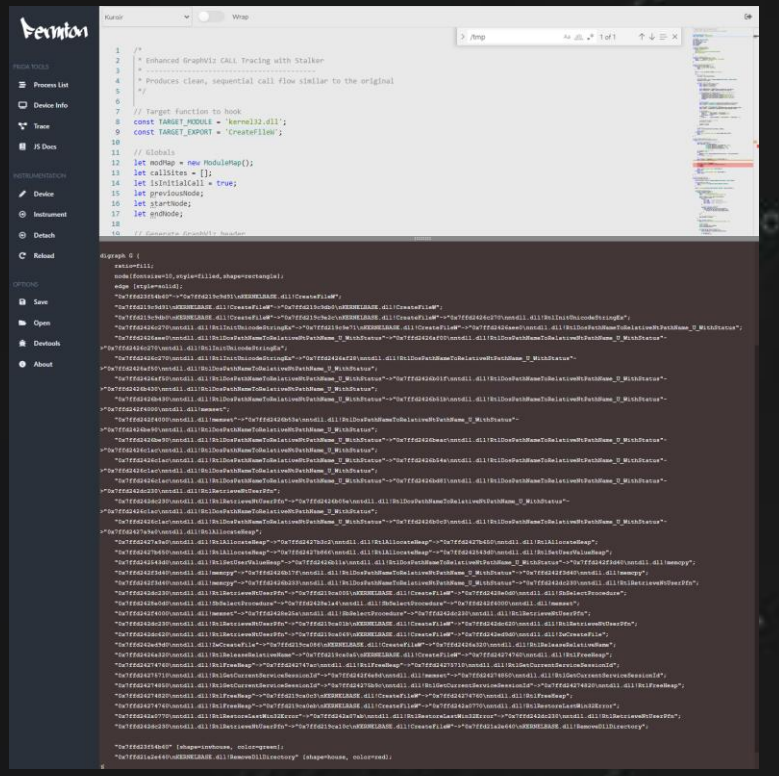
Fermion – Demo Review

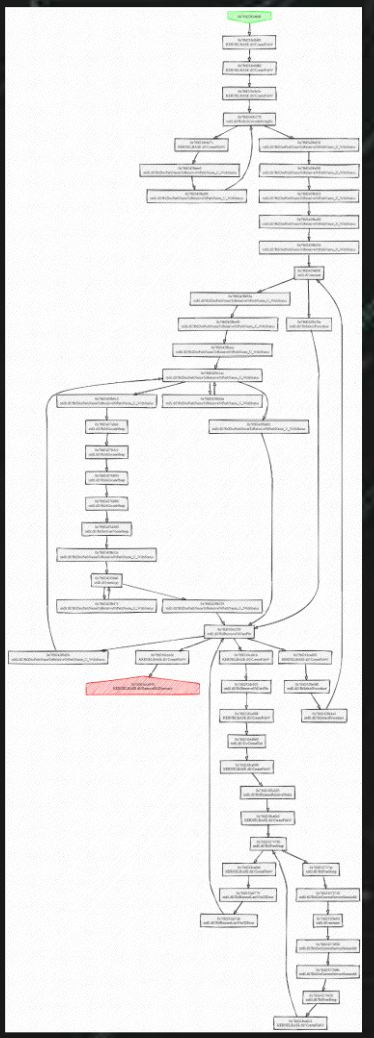
- Hook
  - WriteFile, KernelBase.dll
- Ouput
  - Handle Info
  - Full Buffer Content
  - ASCII Content
  - Return Value



Fermion – Demo Review

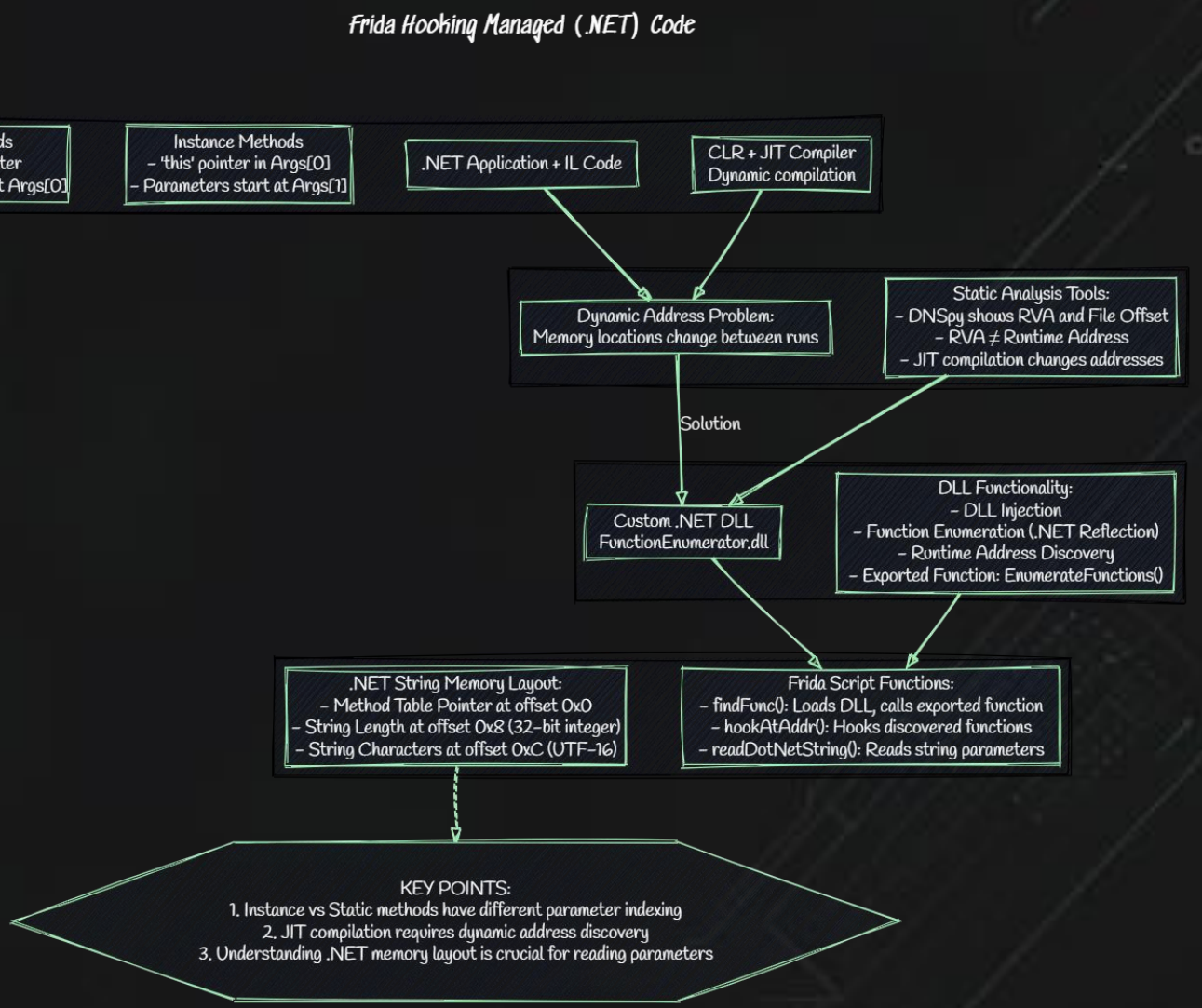
- Stalk API Calls
  - Kernel32.dll, CreateFileW
- Graph using graphviz syntax
- Render with Sketchviz.com





DirtyLittleDotNetHooker

DirtyLittleDotNetHooker



DirtyLittleDotNetHooker

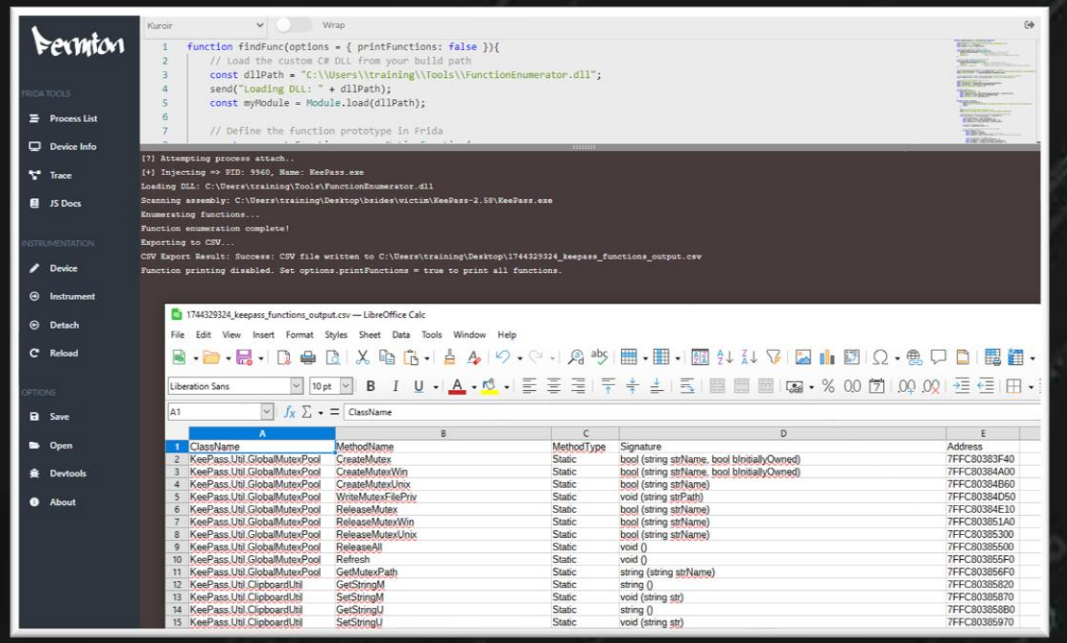
- GitHub:
  - github.com/watson0x90/DirtyLittleDotNetHooker
- Features:
  - .NET DLL
    - Discovering functions
  - Fermion Script:
    - Locate functions, hooking functions, and reading variables
- Updates:
  - .NET DLL
    - Improved function lookup
    - Export functions to CSV
  - Fermion Scripts:
    - Locate functions
    - Hooking functions and reading variables

https://github.com/watson0x90/DirtyLittleDotNetHooker

Let’s see it in action!

KeePass – Enumerate Functions

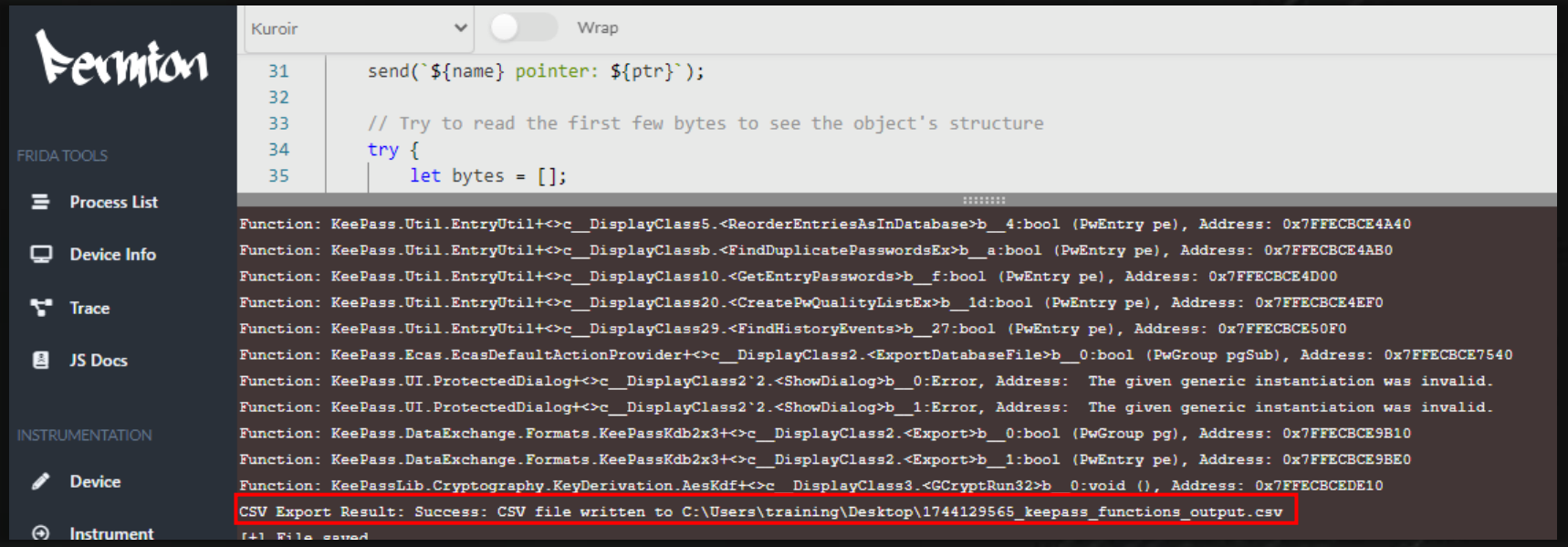
- Overview
  - Enumerate Functions using FunctionEnumerator.dll
  - Review CSV file
- Goal
  - Locate CopyAndMinimize



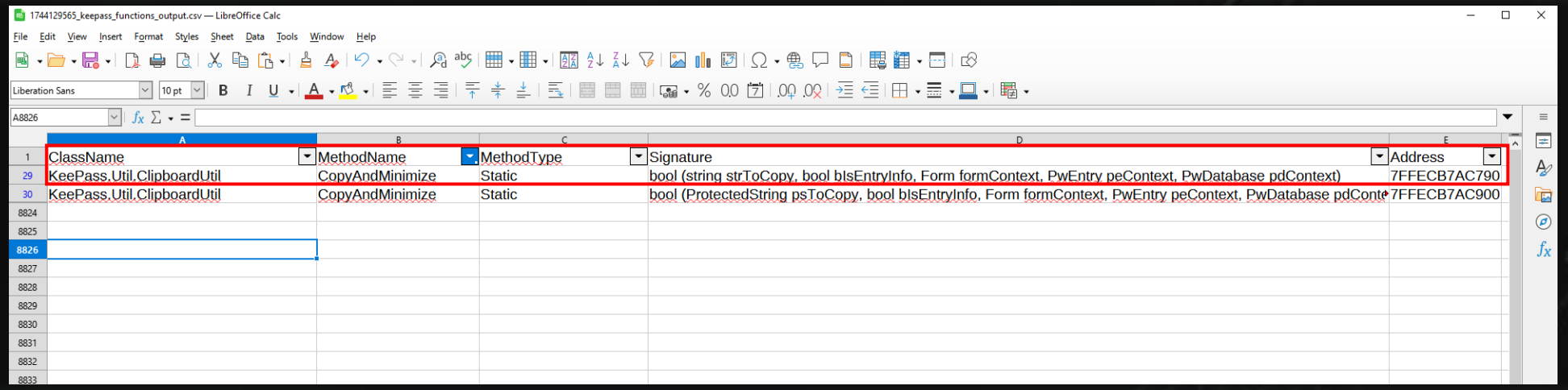
KeePass - CopyAndMinimize

- Overview
  - Review CopyAndMinimize using dnSpy
- Goal
  - Hook and read variables
    - Usernames & Passwords

KeePass - CopyAndMinimize

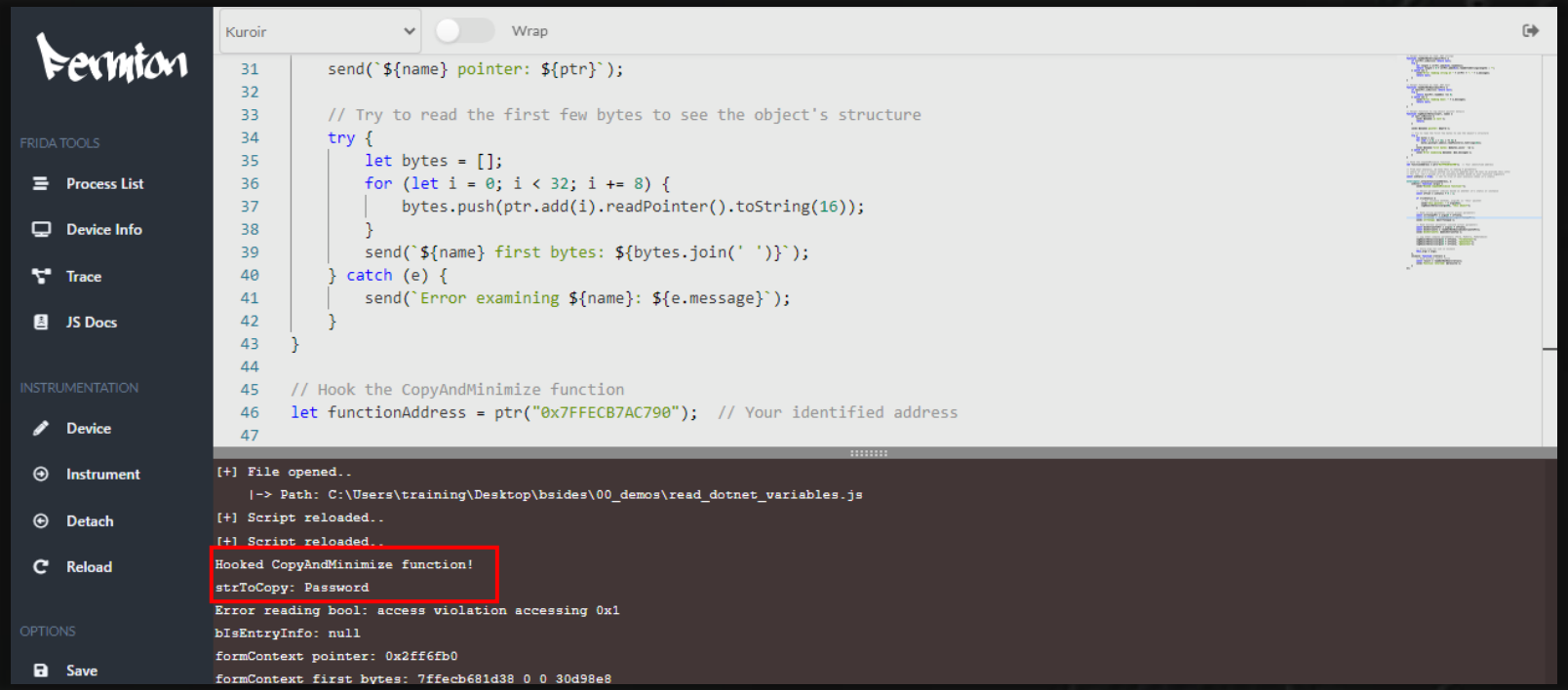


KeePass - CopyAndMinimize



	A	B	C	D	E
1	ClassName	MethodName	MethodType	Signature	Address
29	KeePass.Util.ClipboardUtil	CopyAndMinimize	Static	bool (string strToCopy, bool bIsEntryInfo, Form formContext, PwEntry peContext, PwDatabase pdContext)	7FFECB7AC790
30	KeePass.Util.ClipboardUtil	CopyAndMinimize	Static	bool (ProtectedString psToCopy, bool bIsEntryInfo, Form formContext, PwEntry peContext, PwDatabase pdConte	7FFECB7AC900
8824					
8825					
8826					
8827					
8828					
8829					
8830					
8831					
8832					
8833					

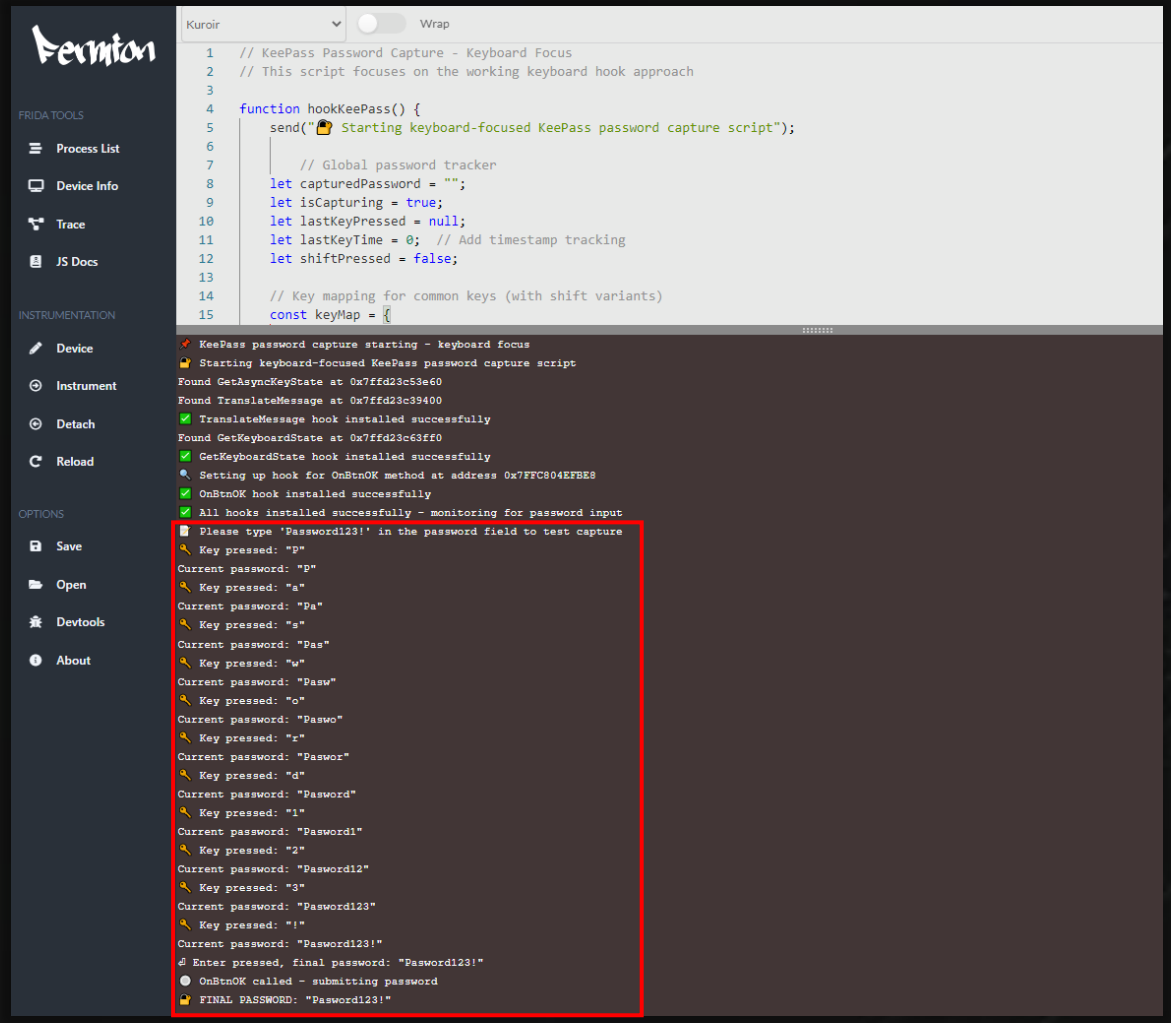
KeePass - CopyAndMinimize



KeePass – Master Key Login

- Overview
  - Why not hook the GUI?
  - Think outside the box
- Goal
  - Capture the text input
  - Win!

KeePass – Master Key Prompt



KeePass – Find interesting functions

- Overview
  - Look for functions we can abuse
  - Inspect them in dnSpy
- Goal
  - Call the function via memory address
  - Spoiler alert
    - RunBuildCommand

That was a lot…

Don’t worry, I forgot most of this too…

Things to remember

- Windows API Hooking
  - Understand application flow
- Unmanaged vs Managed Code
  - Unmanaged
    - More control, more responsibility
    - C, or C++
  - Managed
    - Runtime handles everything (JIT & CLR.dll)
    - C# or F#
- Tools
  - Frida
  - Fermion
  - DirtyLittleDotNetHooker
- Try Hard and build cool things!

Thank you!

Brought to you by your friendly neighborhood watson0x90!