txOne networks

# 神經級逆向技藝

## 從人機翻譯到全自動符號語義推理模型，打造仿生逆向聖杯

JrWei Huang, Sr. Threat Researcher
Threat Research, TXOne Networks Inc.
April 18, 2025 @CYBERSEC 2025

# Authors

## Sr. Threat Researcher, PSIRT and Threat Research at TXOne Networks Inc.

- Jr-Wei Huang (@in0de_16) is a Sr. threat researcher specializing in threat hunting, detection engineering, and malware analysis. He has hands-on experience in developing EDR product features and designing effective detection strategies.
- Jr-Wei Huang has spoken at major conferences such as HITCON, JASEC, and CYBERSEC Taiwan, covering topics including macOS and Windows security, blue team operations, and detection engineering. He has also delivered lectures and training sessions for universities and private companies across Taiwan.

## Team Lead, PSIRT and Threat Research at TXOne Networks Inc.

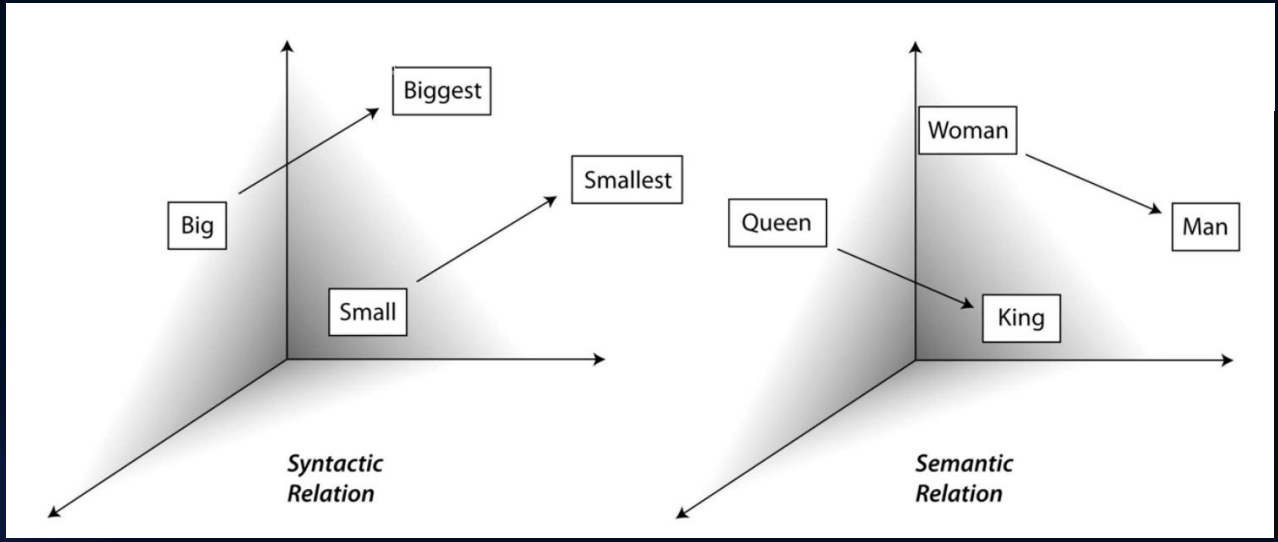
- Sheng-Hao Ma (@aaaddress1) is a team lead of TXOne Networks PSIRT and threat research team, responsible for coordinating product security and threat research. With over 15 years of expertise in reverse engineering, symbolic execution, malware analysis, and machine-learning, he is also part of CHROOT, a cybersecurity community in Taiwan.
- As a frequent speaker, trainer, and instructor, Sheng-Hao has contributed to numerous international conferences and organizations, including Black Hat USA, DEFCON, CODE BLUE, S4, SECTOR, HITB, VXCON, HITCON, and ROOTCON, as well as the Ministry of National Defense and the Ministry of Education. He is the author of "Windows APT Warfare: The Definitive Guide for Malware Researchers," a well-regarded cybersecurity book about reverse engineering of Windows.

# Contents

# The Origins of Symbol Translation

# 文字符號以空間向量表達語義

- 古埃及人應該覺得現代人蠻不禮貌的
  - 因為看不懂自己用的象形文字是什麼意思，就講那叫未知符號
- 但當年哥倫布發現新大陸時候，是如何破譯不同種族語言的？
  - 人一天要吃三餐，那不同語言文本一天出現三次、高機率是飲食行為
  - **傳統 NLP 問題：詞頻統計、以 co-occurrence matrix 投影出語義相似性**
- You shall know a word by the company it keeps (Firth, J. R. 1957:11)
  - **(Google) Word2Vec: Efficient Estimation of Word Representations in Vector Space**
  - (Stanford) GloVe: Global Vectors for Word Representation
  - 當總詞表數量大到一個數量級時，採用分散式離散數值表達而非詞頻方法解決工程問題



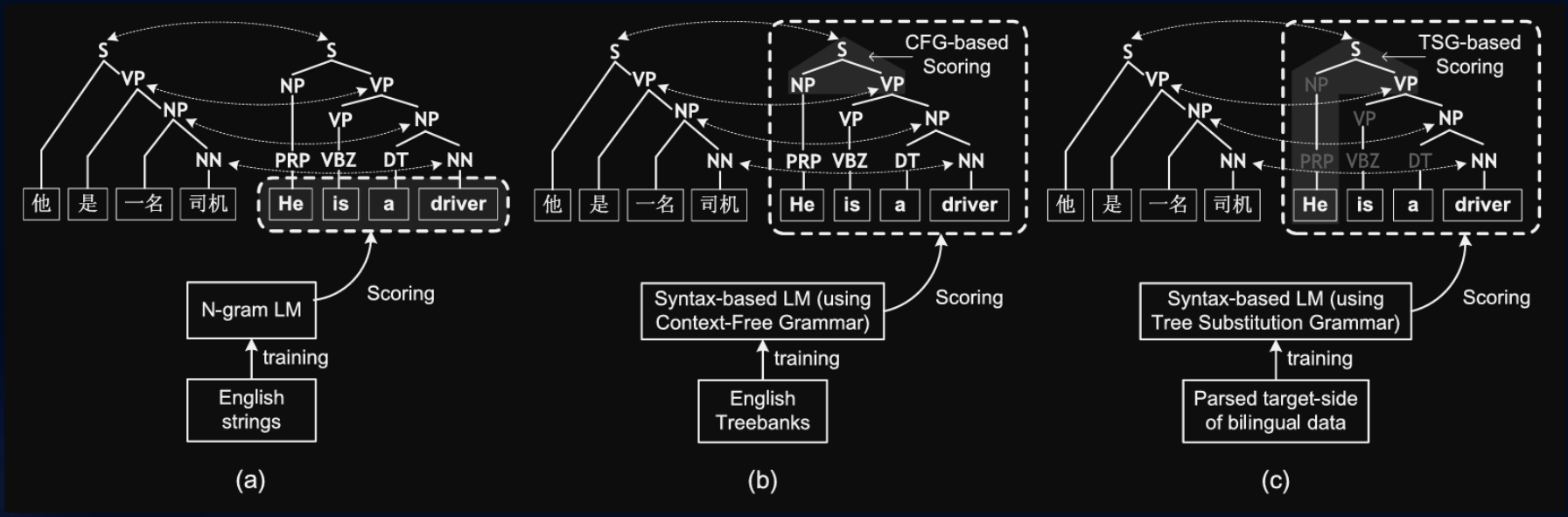


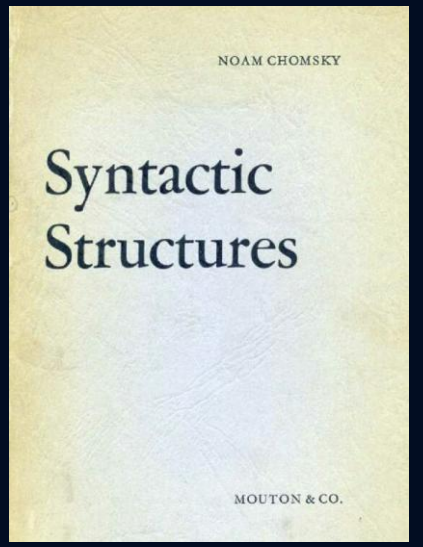
| | i | want | to | eat | chinese | food | lunch | spend |
|---|---|---|---|---|---|---|---|---|
| i | 6 | 828 | 1 | 10 | 1 | 1 | 1 | 3 |
| want | 3 | 1 | 609 | 2 | 7 | 7 | 6 | 2 |
| to | 3 | 1 | 5 | 687 | 3 | 1 | 7 | 212 |
| eat | 1 | 1 | 3 | 1 | 17 | 3 | 43 | 1 |
| chinese | 2 | 1 | 1 | 1 | 1 | 83 | 2 | 1 |
| food | 16 | 1 | 16 | 1 | 2 | 5 | 1 | 1 |
| lunch | 3 | 1 | 1 | 1 | 1 | 2 | 1 | 1 |
| spend | 2 | 1 | 2 | 1 | 1 | 1 | 1 | 1 |

**Figure 3.6** Add-one smoothed bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Previously-zero counts are in gray.

# 考慮兩語言轉換便是一種 A 符號系統轉換至 B 符號系統

- **Syntactic Structures by Chomsky**
  - 簡單的同義字詞取代（substitute）不能達成翻譯問題，例如英文、中文、日文語言系統有其自身句法結構排列方式。
  - **Modern languages follow the grammatical structure of linear B system, which is an old ancient Greek script, e.g. S + V + O**
    - Linear A system is an unknown grammatical structure system
- **Syntax-Based Machine Translation (SBMT)**
  - *Fig. 3. "Comparison of different language modeling methods. (a), (b), and (c) show the cases of traditional n-gram language modeling, CFG-based language modeling and TSG-based language modeling, respectively. The doted arrows represent the connections between the equivalent non-terminals in the synchronous trees."*



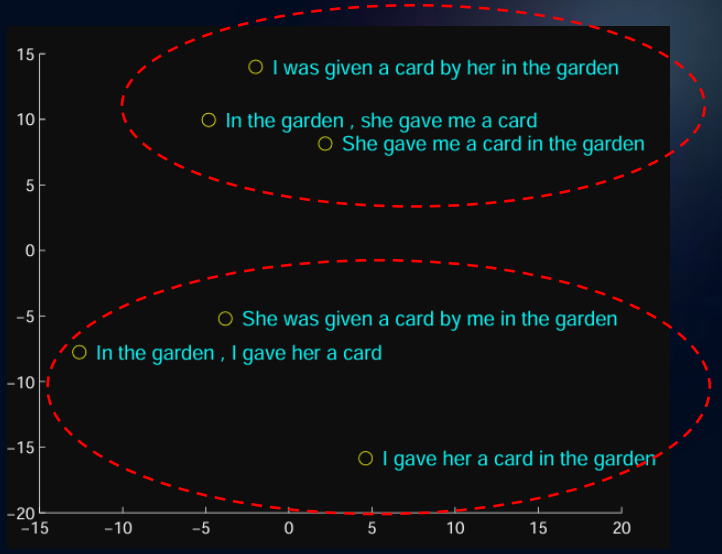



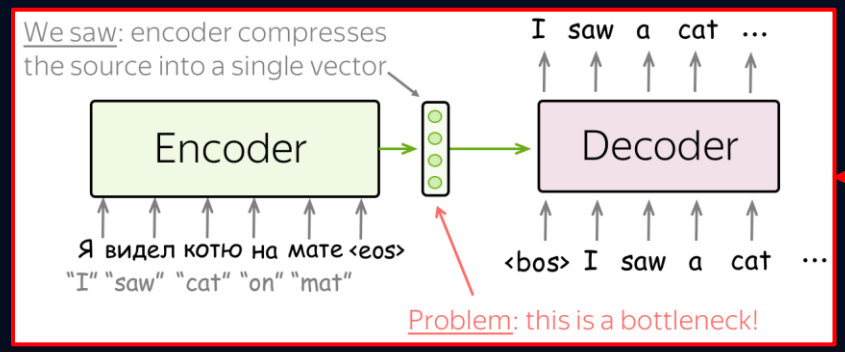



Xiao, Tong, Jingbo Zhu, and Muhua Zhu. "Language modeling for syntax-based machine translation using tree substitution grammars: A case study on chinese-english translation." ACM Transactions on Asian Language Information Processing (TALIP) 10.4 (2011): 1-29.

# Autoregressive modeling as a new trend of AI

- **Consider two language systems are both following the grammatical structures of linear B…**
  - Syntax-free Engineering – Cheap, Fast, but Low-Accuracy approach: human cognitive thinking on language is linear, so we can consider translation task as maximum probability of word-relationship in sequences.
  - So we just project one vectorized source sequence into target language as ease ;)
  - Autoregressive modeling **+ Self-Attention**



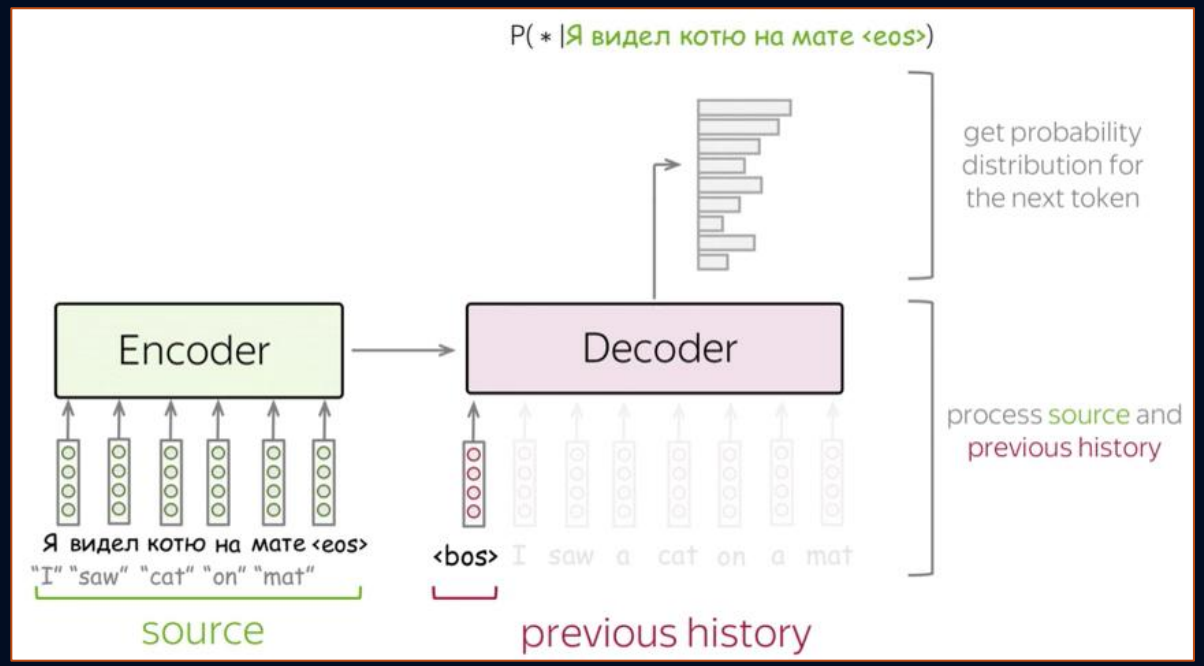




The goal of the LSTM is to estimate the conditional probability $p(y_1, \ldots, y_{T'} | x_1, \ldots, x_T)$ where $(x_1, \ldots, x_T)$ is an input sequence and $y_1, \ldots, y_{T'}$ is its corresponding output sequence whose length $T'$ may differ from $T$. The LSTM computes this conditional probability by first obtaining the fixed-dimensional representation $v$ of the input sequence $(x_1, \ldots, x_T)$ given by the last hidden state of the LSTM, and then computing the probability of $y_1, \ldots, y_{T'}$ with a standard LSTM-LM formulation whose initial hidden state is set to the representation $v$ of $x_1, \ldots, x_T$:

$$Language\ Models:\ P(y_1, y_2, \ldots, y_n) = \prod_{t=1}^{n} p(y_t | y_{<t})$$

$$Conditional\ Language\ Models:\ P(y_1, y_2, \ldots, y_n, | x) = \prod_{t=1}^{n} p(y_t | y_{<t}, x)$$

condition on source x



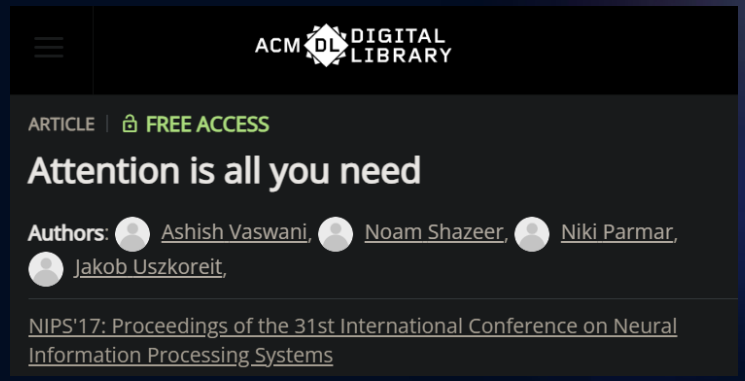

lena-voita.github.io/nlp_course/seq2seq_and_attention.html

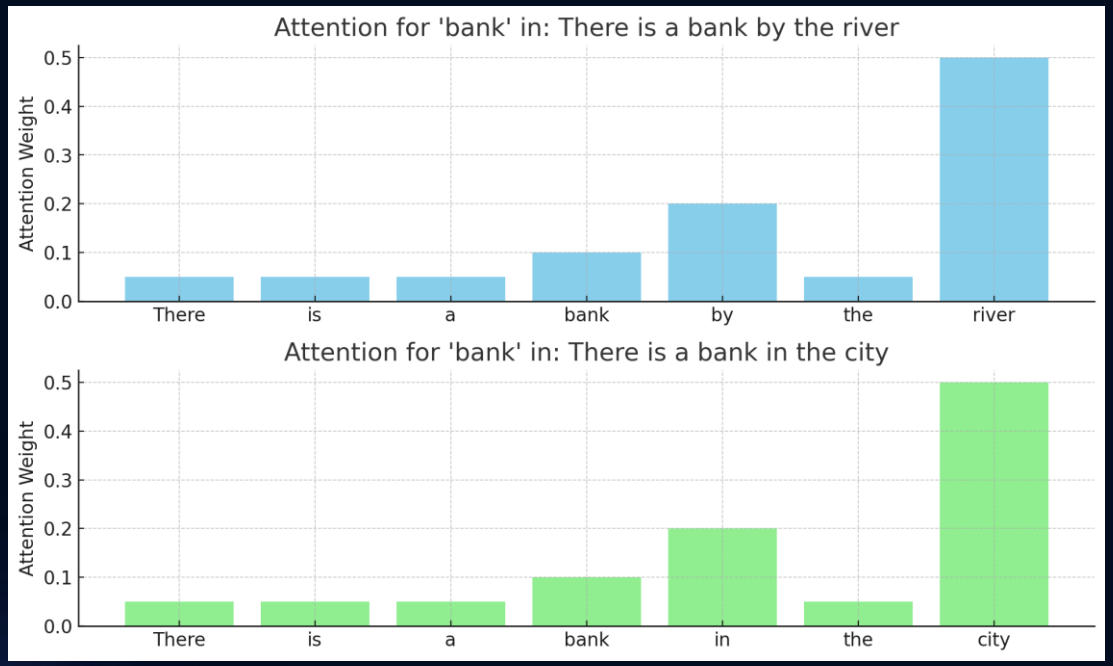
Keep the Operation Running

(Google 2014) Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." Advances in neural information processing systems 27 (2014). cited ~ 28497 times

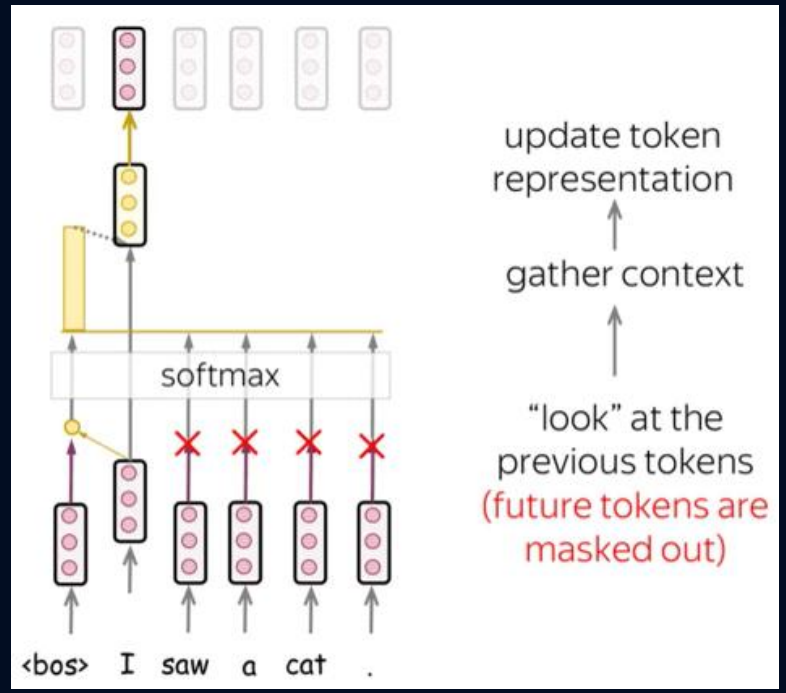
# Transformer – Attention Is All You Need

- **Google @ NeurIPS'17: Attention Is All You need**
  - No LSTM or CNN required, ONLY Attention ☺
  - Encoder: dynamically adjust attention based on the entire context
  - Decoder: from Cross-Attention to Self-Attention




Encoder




Decoder



Vaswani, AshisVaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.h, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

# Google's AI Lab: deciphering lost languages

- MIT + Google @ 57th Association for Computational Linguistics (ACL 2019)

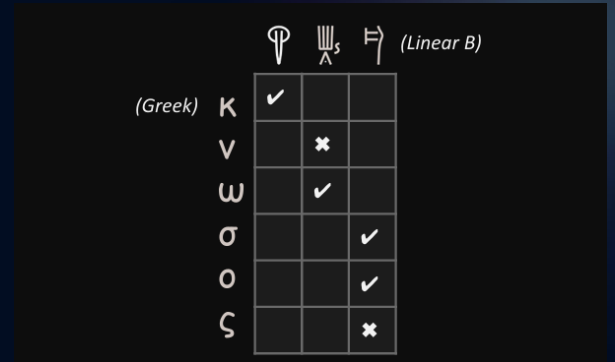


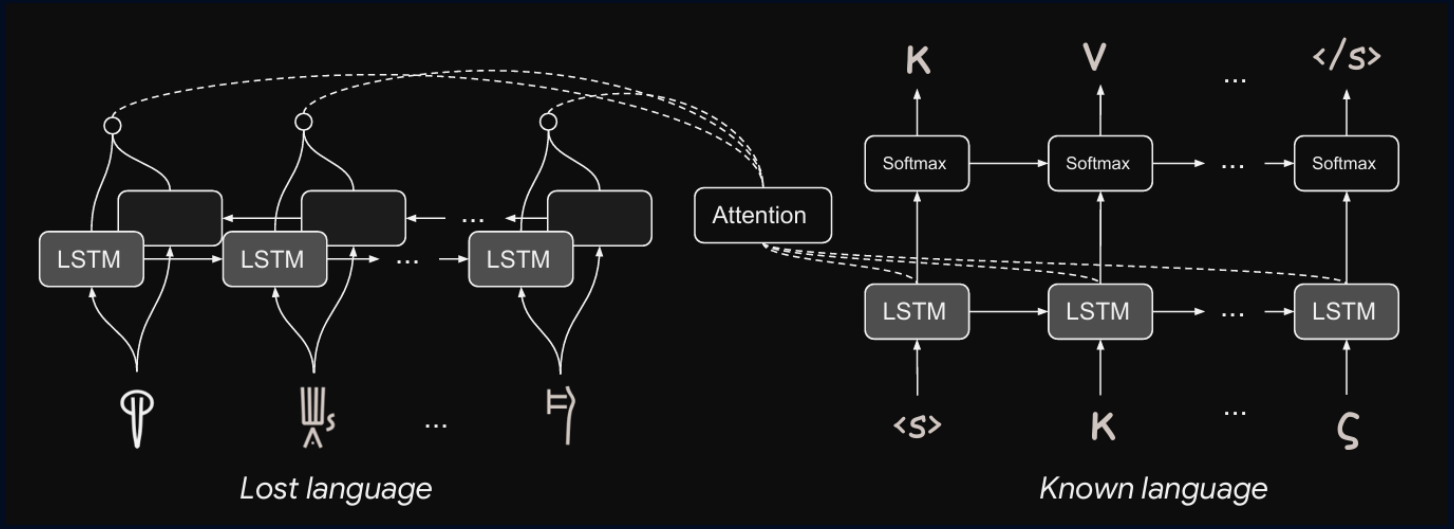

Figure 2: An example of alignment between a Linear B word and Greek word. ✔ and ✖ denote correct and wrong alignment positions respectively. The misalignment between 𐀏 and ν incurs a deletion error; 𐀞 and ζ incurs an insertion error.

[Submitted on 16 Jun 2019]

**Neural Decipherment via Minimum-Cost Flow: from Ugaritic to Linear B**

Jiaming Luo, Yuan Cao, Regina Barzilay

In this paper we propose a novel neural approach for automatic decipherment of lost languages. To compensate for the lack of strong supervision signal, our model design is informed by patterns in language change documented in historical linguistics. The model utilizes an expressive sequence-to-sequence model to capture character-level correspondences between cognates. To effectively train the model in an unsupervised




MIT Technology Review

**Machine learning has been used to automatically translate long-lost languages**

Some languages that have never been deciphered could be the next ones to get the machine translation treatment.

By Emerging Technology from the arXiv

July 1, 2019

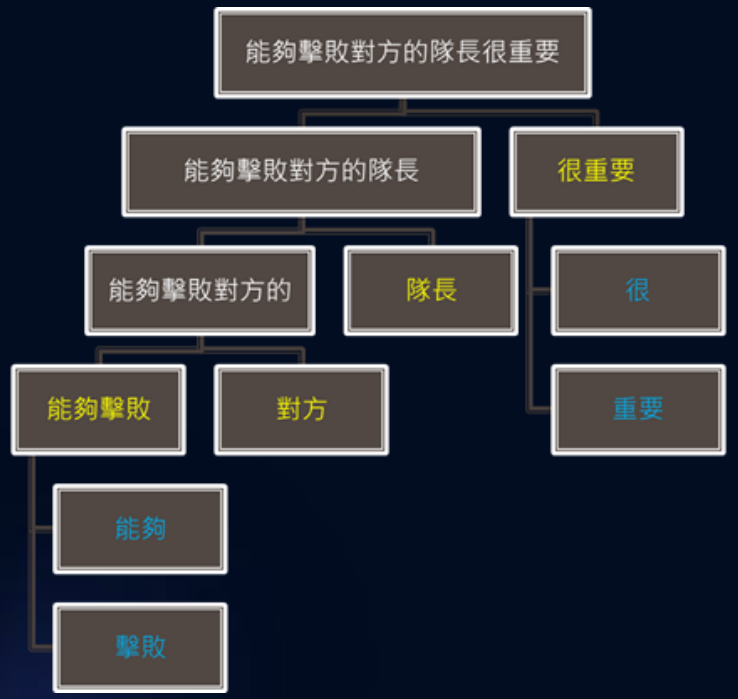
Ancient Greek inscribed in stone

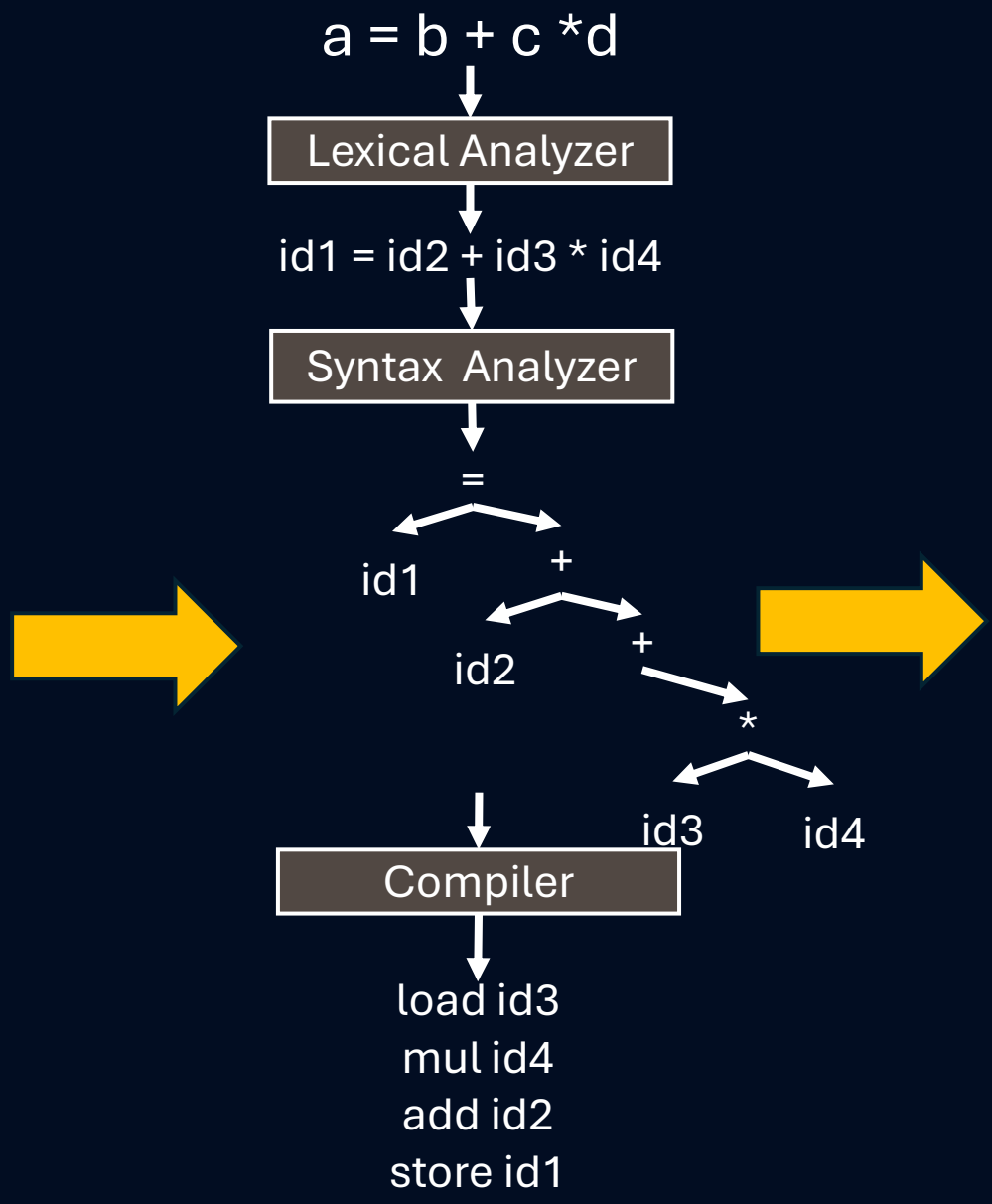


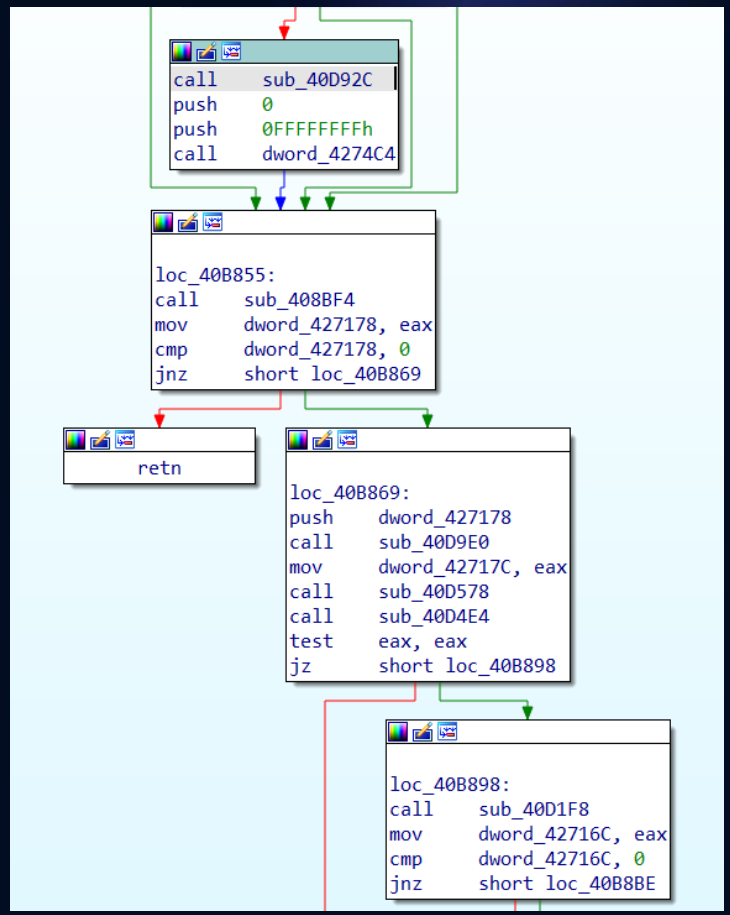
Jiaming Luo, Yuan Cao, and Regina Barzilay. 2019. Neural Decipherment via Minimum-Cost Flow: From Ugaritic to Linear B. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 3146–3155, Florence, Italy. Association for Computational Linguistics.

# Data-driven AI Reversing:
# From Translation to Neuro-Symbolic Reversing

# 句法結構 vs. 程式執行流程結構



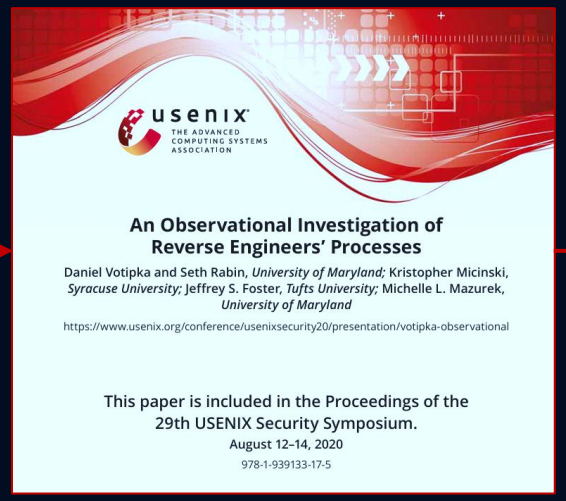



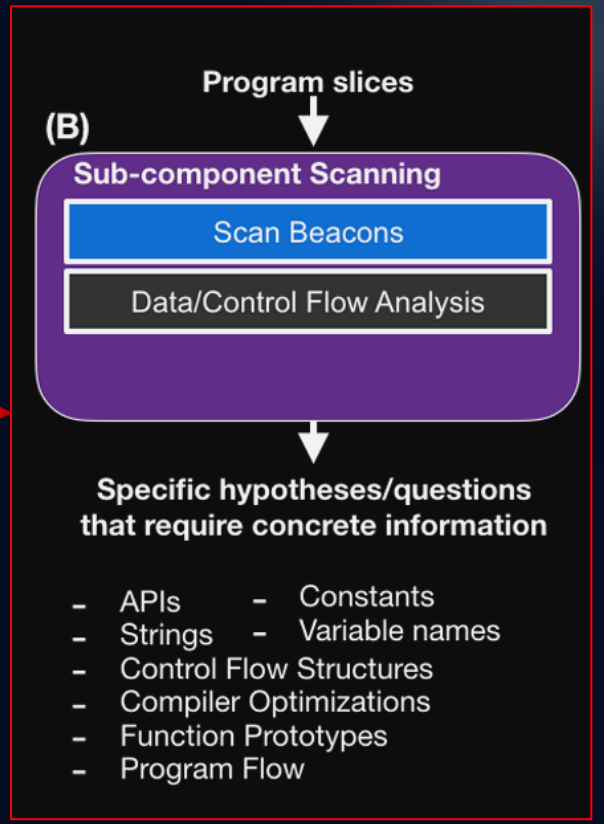



txOne networks

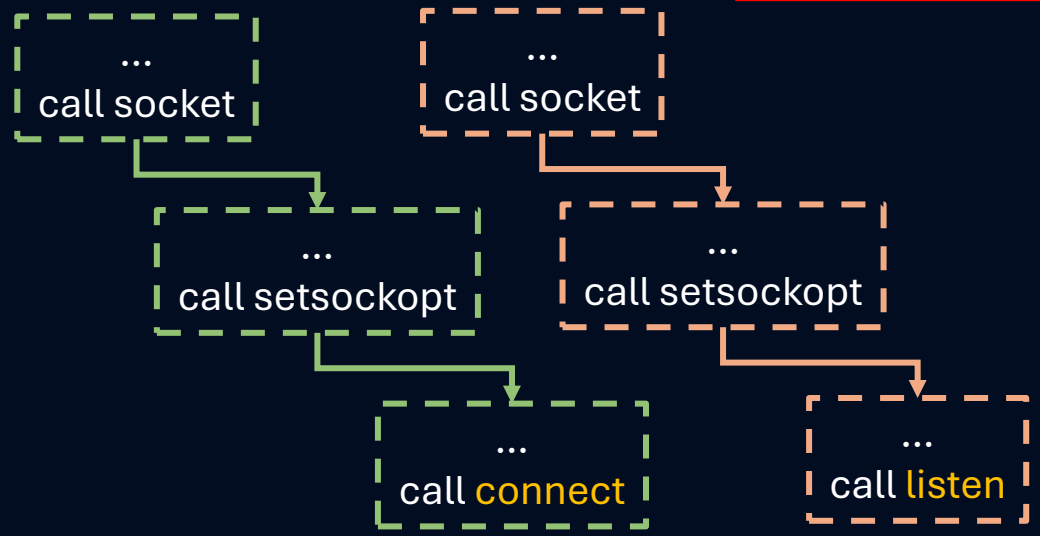
# AI driven future for reverse engineering

- **(USENIX'20) An Observational Investigation of Reverse Engineers' Processes**
  - Reverse engineering is fundamentally an inference process translating assembly into meaningful words and concepts
  - If we can infer how data is referenced and how functions interact based on the control flow graph, we can uncover the underlying semantics




- **(Israe Technion @ ACM'20) Neural Reverse Engineering of Stripped Binaries using Augmented Control Flow Graphs**
  - ➢ **Use Transformer to model the semantics of API usages from the binary functions**

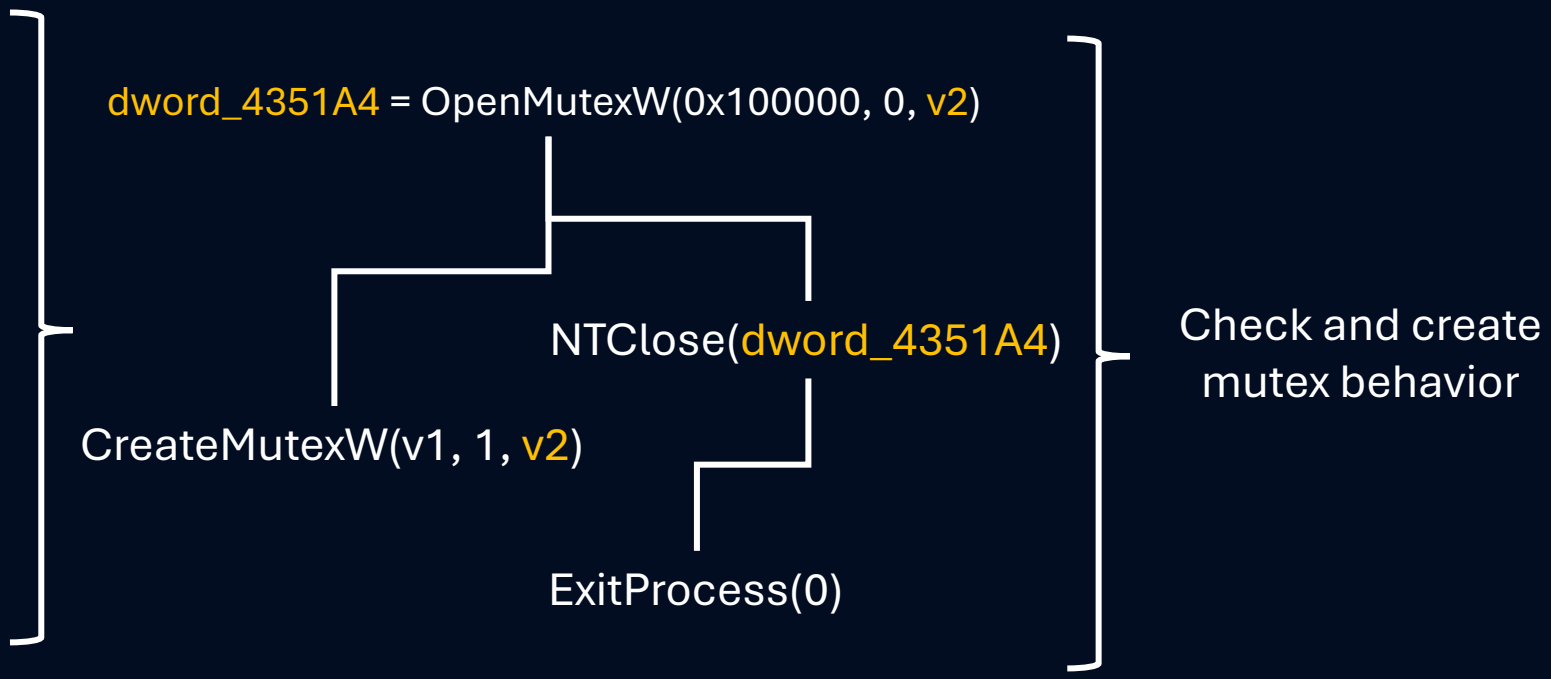



# AI driven future for reverse engineering

- **Case Study**
  - Human analysts observe what the function is doing through the extracted API sequences
  - By using API sequences, we can infer what operations the function is performing

```
void [redacted]()
{
  int *v0; // eax
  int v1[3]; // [esp+0h] [ebp-10h] BYREF
  int v2; // [esp+Ch] [ebp-4h]

  if ( byte_425129 )
  {
    v2 = sub_407FBC();
    dword_4251A4 = jump_kernel32_OpenMutexW(0x100000, 0, v2);
    if ( dword_4251A4 )
    {
      jump_ntdll_NtClose(dword_4251A4);
      if ( byte_42512E )
        sub_4104B4(0, 0, 0, byte_42512E);
      jump_kernel32_ExitProcess(0);
    }
    if ( GetWindowsVers_401564() < 0x3C )
      v0 = dword_409B60;
    else
      v0 = dword_409B00;
    v1[0] = 12;
    v1[1] = v0;
    v1[2] = 0;
    dword_4251A4 = jump_kernel32_CreateMutexW(v1, 1, v2);
    sub_4068EC(v2);
  }
}
```




txOne networks

# AI RE:

## from Decompile, Symbolic, to Neuro-RE

- Augmenting call sites concrete and abstract values
  - data flow between components is important
- Taint analysis helps us understand how the arguments are propagated through the program

ACM DIGITAL LIBRARY

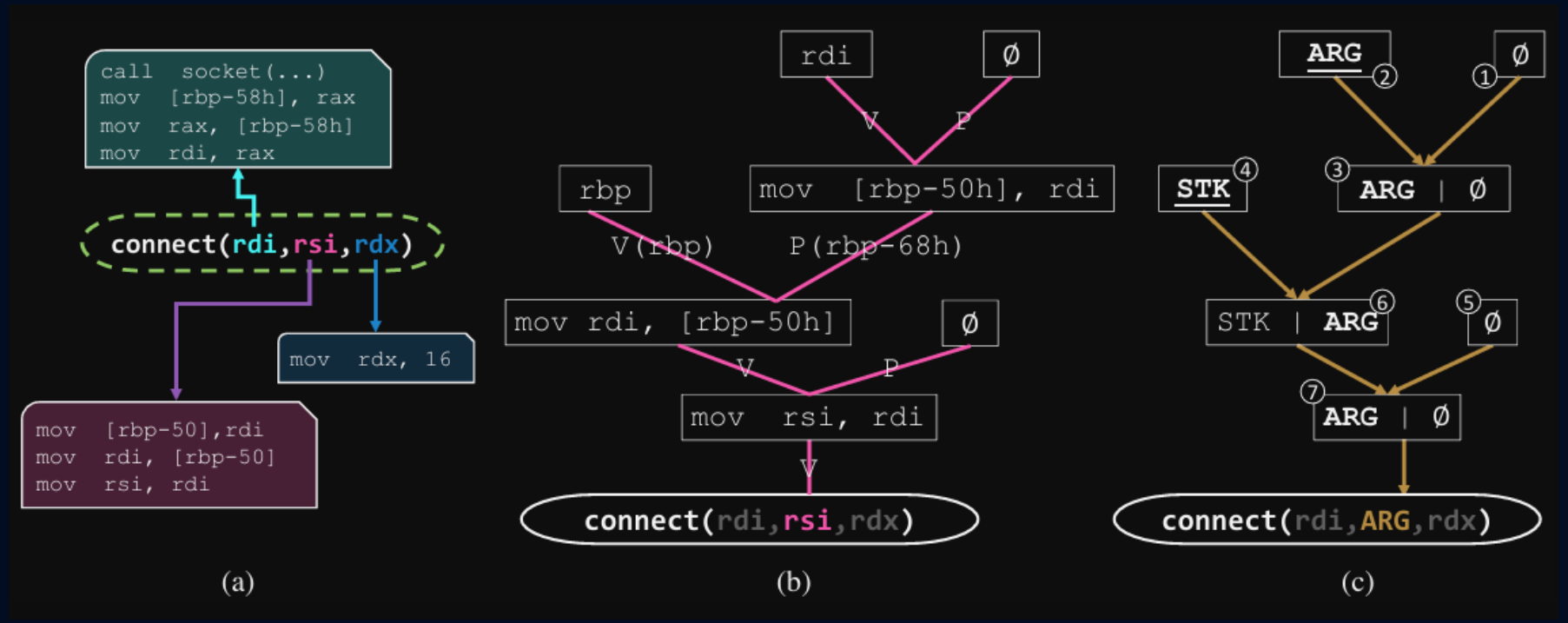
RESEARCH-ARTICLE | OPEN ACCESS

**Neural reverse engineering of stripped binaries using augmented control flow graphs**

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

Proceedings of the ACM on Programming Languages, Volume 4, Issue OOPSLA
Article No.: 225, Pages 1 - 28 • https://doi.org/10.1145/3428293

```
1: mov rsi, rdi
2: mov rdx, 16
3: mov rax, [rbp-58h]
4: mov rdi, rax
5: call connect
6: mov edx, eax
7: mov rax, [rbp-4h]
...
8: mov rax, [rbp-58h]
9: mov rdi, rax
10: mov r8, 4
11: mov rdx, 10
12: mov esi, 0
13: lea rcx, [rbp-88h]
14: call setsockopt
```






txOne networks

# AI RE:

## from Decompile, Symbolic, to Neuro-RE

- Control Flow Graph is too heavy to machine learning
  - Augmented Call Sites Graph
- Use a concrete value or an abstract value to replace the register name in the sink of the tree
  - concrete value, ARG, GLOBAL, RET, STK

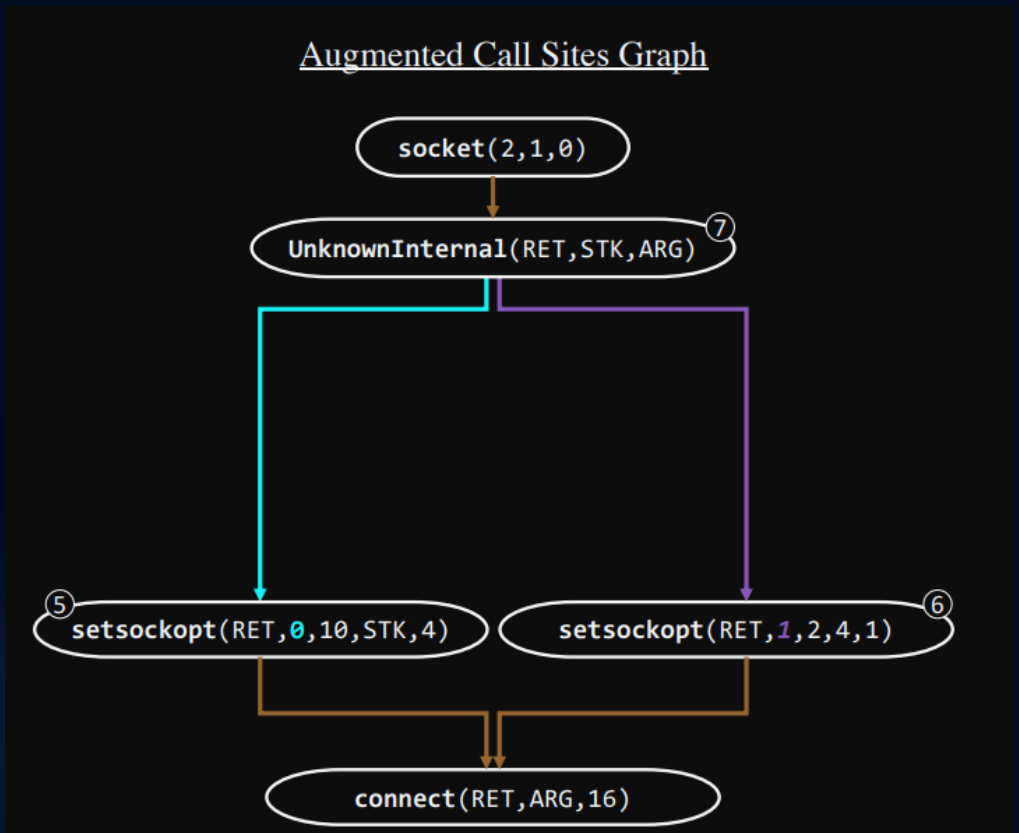
ACM DIGITAL LIBRARY

RESEARCH-ARTICLE | OPEN ACCESS

**Neural reverse engineering of stripped binaries using augmented control flow graphs**

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

Proceedings of the ACM on Programming Languages, Volume 4, Issue OOPSLA
Article No.: 225, Pages 1 - 28 • https://doi.org/10.1145/3428293




### 5.3 Encoding Call Site Sequences with Transformers

In the Transformer model, we encode each call site sequence separately using $N$ Transformer-encoder layers, followed by attention pooling to represent a call site sequence as a single vector:

$$z = \text{Transformer}_{encoder}\ (callsite_1, ..., callsite_l)$$

Finally, we represent the entire procedure as a set of its encoded call site sequences $\{z_1, z_2, ..., z_n\}$, which is passed to the Transformer-decoder as described in Section 3.3.

$$encode\_callsite\left(w_1 ... w_{k_s}, value_1, ..., value_{k_{args}}\right) = \left[\left(\sum_i^{k_s} E^{names}_{w_i}\right)\ ;\ E^{values}_{value_1}\ ;\ ...\ ;\ E^{values}_{value_{kargs}}\right]$$

| | Instruction | $V_{read}$ | $V_{write}$ | $P_{read}$ | $P_{write}$ |
|---|---|---|---|---|---|
| $inst_1$ | `mov rax, 5` | 5 | rax | ∅ | ∅ |
| $inst_2$ | `mov rax, [rbx+5]` | rbx | rax | rbx+5 | ∅ |
| $inst_3$ | `call rcx` | rcx | rax | rcx | ∅ |

# Results

- AI version of IDA 7.2+ features, Lumina & FLIRT
- **[As Forensics] It works, even better than zero-visibility!**
  - But accuracy rate still low ~39-49%
  - **how to make it better?**

| Error Type | Package | Ground Truth | Predicted Name |
|---|---|---|---|
| Programmers VS English Language | wget | i18n_initialize | i18n_init |
| | direvent | split_cfg_path | split_config_path |
| | gzip | add_env_opt | add_option |
| Data Structure Name Missing | gtypist | get_best_speed | get_list_item |
| | wget | ftp_parse_winnt_ls | parse_tree |
| | direvent | filename_pattern_free | free_buffer |
| | gzip | abort_gzip_signal | fatal_signal_handler |

| Model | Stripped | | | *Stripped & Obfuscated API calls* | | |
|---|---|---|---|---|---|---|
| | Precision | Recall | F1 | Precision | Recall | F1 |
| LSTM-text | 22.32 | 21.16 | 21.72 | 15.46 | 14.00 | 14.70 |
| Transformer-text | 25.45 | 15.97 | 19.64 | 18.41 | 12.24 | 14.70 |
| Debin [He et al. 2018] | 34.86 | 32.54 | 33.66 | 32.10 | 28.76 | 30.09 |
| DIRE [Lacomis et al. 2019] | 38.02 | 33.33 | 35.52 | 23.14 | 25.88 | 24.43 |
| Nero-LSTM | 39.94 | 38.89 | 39.40 | 39.12 | 31.40 | 34.83 |
| Nero-Transformer | 41.54 | 38.64 | 40.04 | 36.50 | 32.25 | 34.24 |
| Nero-GNN | **48.61** | **42.82** | **45.53** | **40.53** | **37.26** | **38.83** |

```
558BEC
      0EFF7604.........59595DC3558BEC0EFF7604.........59595DC3 _registerbgidriver
      1E
        078A66048A460E8B5E108B4E0AD1E9D1E980E1C0024E0C8A6E0A8A76 _biosdisk
        B4
          1AC55604CD211F5DC3                                     _setdta
          2FCD210653B41A8B5606CD21B44E8B4E088B5604CD219C5993B41A _findfirst
```

TclKit_SetKitPath

```
1  int __cdecl                    (const char *a1)
2  {
3    unsigned int v1; // kr04_4
4    unsigned int v2; // ebx
5
6    if ( a1 )
7    {
8      v1 = strlen(a1) + 1;
9      v2 = v1 - 1;
10     if ( dword_68B02C )
11       sub_4EE49C(dword_68B02C);
12     dword_68B02C = sub_4EE344(v1);
13     qmemcpy(dword_68B02C, a1, v2);
14     *(dword_68B02C + v2) = 0;
15   }
16   return dword_68B02C;
17 }
```

Keep the Operation Running

David, Yaniv, Uri Alon, and Eran Yahav. "Neural reverse engineering of stripped binaries using augmented control flow graphs." Proceedings of the ACM on Programming Languages 4.OOPSLA (2020): 1-28.

# Reversing Challenge of Variable Type and Naming

# Reversing Challenge of Variable Type

- (24h2) KernelBase ! K32GetModuleFileNameExW()
  - Read specific filepath of targeted module in the other process
  - Fetch PEB.Ldr (PEB_LDR_DATA) of selected module to get the path from LDR_DATA_TABLE_ENTRY.FullDllName.Buffer
  - [note] this code snippets enable Stack Slot Reuse (Stack Frame Optimization)

```
typedef struct _PROCESS_BASIC_INFORMATION {
    NTSTATUS    ExitStatus;                    // 0x00 (4 bytes)
    ULONG       Reserved0;                     // 0x04 (4 bytes, padding/alignment)
    PPEB        PebBaseAddress;                // 0x08 (8 bytes)
    ULONG_PTR   AffinityMask;                  // 0x10 (8 bytes)
    LONG        BasePriority;                  // 0x18 (4 bytes)
    ULONG       Reserved1;                     // 0x1C (4 bytes, padding/alignment)
    ULONG_PTR   UniqueProcessId;               // 0x20 (8 bytes)
    ULONG_PTR   InheritedFromUniqueProcessId; // 0x28 (8 bytes)
} PROCESS_BASIC_INFORMATION, *PPROCESS_BASIC_INFORMATION;
```

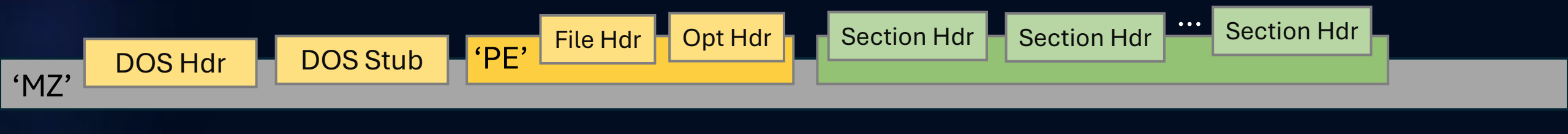
```
1  DWORD __stdcall K32GetModuleFileNameExW(
2    HANDLE hProcess, HMODULE hModule, LPWSTR lpFilename, DWORD nSize)
3  {
4    __int64 Buffer; // [rsp+48h] [rbp-B8h] BYREF
5    ULONG_PTR NumberOfBytesRead; // [rsp+60h] [rbp-A8h] BYREF
6    ULONG_PTR ProcessInformation; // [rsp+68h] [rbp-A0h] BYREF
7    _QWORD procInfo[5]; // [rsp+70h] [rbp-98h] BYREF
8    PVOID BaseAddress; // [rsp+F8h] [rbp-10h]
9
10   if ( hModule )
11   {
12     memset_0(v33, 0, 0x88ui64);
13     *(_QWORD *)ReturnLength = hModule;
14     ...
15     memset(procInfo, 0, sizeof(procInfo));
16     v8 = NtQueryInformationProcess(
17       hProcess, ProcessBasicInformation, procInfo, 0x30u, 0i64
18     );
19     ...
20     v9 = NtReadVirtualMemory(
21       hProcess, (PVOID)(procInfo[1] + 24i64), &Buffer, 8ui64, &NumberOfBytesRead
22     );
23 LABEL_16:
24 ...
25       v10 = Buffer + 32;
26       v9 = NtReadVirtualMemory(hProcess, (PVOID)(Buffer + 32), &v26, 8ui64, &Proc
27
28
```

```
1  DWORD __stdcall K32GetModuleFileNameExW(
2    HANDLE hProcess, HMODULE hModule, LPWSTR lpFilename, DWORD nSize)
3  {
4    LDR_DATA_TABLE_ENTRY *Buffer; // [rsp+48h] [rbp-B8h] BYREF
5    ULONG_PTR NumberOfBytesRead; // [rsp+60h] [rbp-A8h] BYREF
6    DWORD ProcessInformation; // [rsp+68h] [rbp-A0h] OVERLAPPED BYREF
7    PROCESS_BASIC_INFORMATION procInfo; // [rsp+70h] [rbp-98h] BYREF
8    PVOID BaseAddress; // [rsp+F8h] [rbp-10h]
9
10   if ( hModule )
11   {
12     memset_0(v32, 0, 0x88ui64);
13     *(_QWORD *)ReturnLength = hModule;
14     ...
15     memset(&procInfo, 0, sizeof(procInfo));
16     v8 = NtQueryInformationProcess(
17       hProcess, ProcessBasicInformation, &procInfo, 0x30u, 0i64
18     );
19     ...
20     v9 = NtReadVirtualMemory(
21       hProcess, &procInfo.PebBaseAddress->Ldr, &Buffer, 8ui64, &NumberOfBytesRead
22     );
23 LABEL_16:
24 ...
25       p_InMemoryOrderModuleList = &Buffer->InMemoryOrderModuleList;
26       v9 = NtReadVirtualMemory(hProcess, &Buffer->InMemoryOrderModuleList, &v26, 8ui6
27
28
```

# Threat Example: Reflective Loader

```
if ( *(_WORD *)a1 == 23117 )
{
  v2 = *(int *)(a1 + 60);
  if ( (int)v2 <= 1024 && *(_DWORD *)(v2 + a1) == 17744 )
  {
    v3 = *(unsigned int *)(v2 + a1 + 144);
    if ( (_DWORD)v3 )
    {
      v20 = *(unsigned int *)(v2 + a1 + 148);
      v19 = 0i64;
      if ( !*(_DWORD *)(v2 + a1 + 148) )
        return 1;
      for ( i = (unsigned int *)(v3 + a1); ; i += 5 )
      {
        if ( !*i && !i[4] )
          return 1;
        v5 = (const char *)(a1 + i[3]);
        printf_1("    [+] Import DLL: %s\n", v5);
        v6 = i[4];
        v7 = v6;
        v8 = v6 + a1;
        if ( *i )
          v7 = *i;
        v9 = (FARPROC *)(v6 + a1);
        while ( 1 )
        {
          v10 = (char *)v9 + v7 - v8;
          if ( (*(_DWORD *)(*i + a1) & 0x80000000) != 0 )
          {
```

```
if ( a1 )
{
  if ( a1->e_magic == 'ZM' )
  {
    e_lfanew = a1->e_lfanew;
    if ( e_lfanew <= 1024 && *(&a1->e_magic + e_lfanew) == 'EP' )
    {
      v3 = *(&a1[2].e_sp + e_lfanew);
      if ( v3.VirtualAddress )
      {
        v20 = *(&a1[2].e_ip + e_lfanew);
        v19 = 0i64;
        if ( !*(&a1[2].e_ip + e_lfanew) )
          return 1;
        for ( i = (a1 + *&v3); ; ++i )
        {
          if ( !i->Characteristics && !i->FirstThunk )
            return 1;
          v5 = a1 + i->Name;
          printf_1("    [+] Import DLL: %s\n", v5);
          FirstThunk = i->FirstThunk;
          Characteristics = FirstThunk;
          v8 = a1 + FirstThunk;
          if ( i->Characteristics )
            Characteristics = i->Characteristics;
          v9 = a1 + FirstThunk;
          while ( 1 )
```



# Reversing Challenge of Variable Type

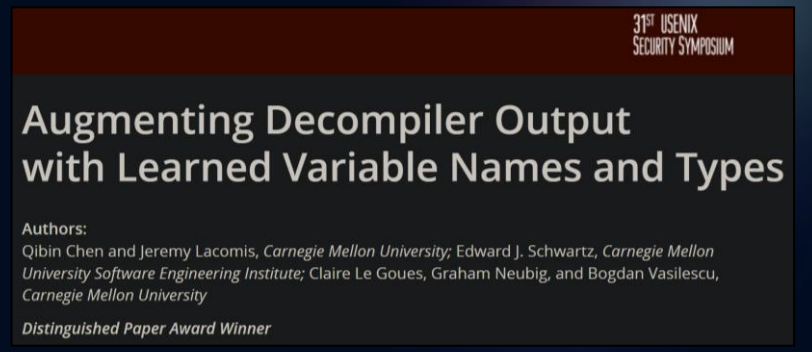
- This technique abuses the CMSTPLUA COM object to execute arbitrary commands
  - CoGetObject, when used with the CLSID and IID of CMSTPLUA, returns an object of type ICMLuaUtil
- How did a human expert predict?
  - Image the memory layout of an unknown variables
  - Regard to the strategy of taint-analysis, and analyze the used offset of the same begin of memory
  - Note - Stack Frame Optimization still a big problem.

```
decrypt_401240(clsid, 34);                    // CLSID: Elevation:Administrator!new:{3E5FC7F9-9A51-4367-9063-A120244FBEC7}
jump_ntdll_memset(v4, 0, 36);
v4[0] = 36;
v4[5] = 4;
return jump_ole32_CoGetObject(clsid, v4, IID_ICMLuaUtil, a1);// IID: {6EDD6D74-C007-4E75-B76A-E5740995E24C}
```

```
1  int sub_40BABC()
2  {
3    ICMLuaUtil *ComObj; // [esp+8h] [ebp-8h] BYREF
4
5    result = jump_ole32_CoInitialize(0);
6    if ( result != 0x54F )
7    {
8      GetObject_40B944(&ComObj);                  // Get Com object (CMSTP->ICMLuaUtil)
9      if ( ComObj )
10     {
11       cmd = ParseCmd_409740(Commandline, ImageName);// Parses a Unicode command line string and
12       arg = jump_shell32_CommandLineToArgvW(cmd, &arg_count);
13       ...
14       if ( !(ComObj->lpVtbl->ShellExec)(ComObj, *_arg, _cmd, 0, 0, 0) )// Create process using ICMLuaUtil
15         (ComObj->lpVtbl->Release)(ComObj);
16       ...
17     }
18     return jump_combase_CoUninitialize();
19   }
20   return result;
21 }
```

```
13        ...
14        if ( !(*(*v8 + 36))(v8, *v3, v4, 0, 0, 0) )
15          (*(*v8 + 8))(v8);
16        ...
```

# Challenge of Data Type/Naming Prediction




- Reverse engineering often encounters Decompiler errors in variable type prediction and meaningless variable names, requiring analysts to spend significant time manually interpreting the code
- Carnegie Mellon University - Software Engineering Institute @ USENIX'22
  - Supporting 48,888 types and generalizing to unseen functions or code, it covered the most of known C/C++ types!
  - Use tainted offsets of several properties of an income memory layout via Hex-Rays decompile, to predict the max probability known data-types

```
void fun() {
  // stack layout:
  // [xxx][p][yyyy]
  char x[3];
  int y;
  // ...
}
```

(a) Original code

```
void fun() {
  // stack layout:
  // [xxxx][yyyy]
  char x[4];
  int y;
  // ...
}
```

(b) Decompiled `fun`

```
typedef struct point {
  float x;
  float y;
} pnt;
```

```
void fun() {
  pnt p1, p2;
  p1.x = 1.5;
  p1.y = 2.3;
  // ...
  use_pts(&p1, &p2);
}
```

(a) Original code

```
void fun() {
  float v1[2], v2[2];
  v1[0] = 1.5;
  v1[1] = 2.3;
  // ...
  use_pts(v1, v2);
}
```

(b) Decompiled `fun`

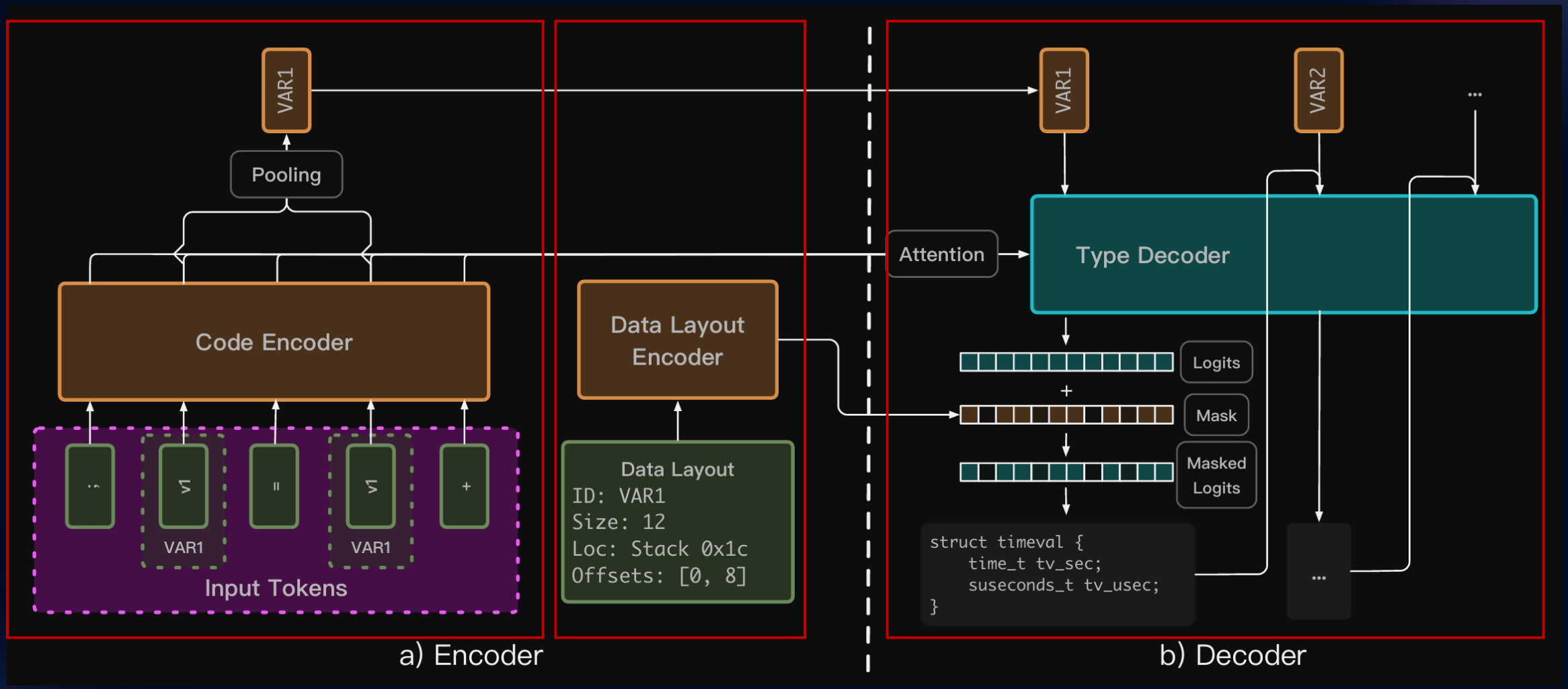
```
typedef struct {
  int x;
  int y;
} point;

double func(point *a1, point *a2) {
  double v1, v2;
  v1 = pow((a1->x - a2->x), 2);
  v2 = pow((a1->y - a2->y), 2);
  return sqrt(v1 + v2);
}
```

```
double func(int *a1, int *a2) {
  double v1, v2;
  v1 = pow((*a1 - *a2), 2);
  v2 = pow((a1[1] - a2[1]), 2);
  return sqrt(v1 + v2);
}
```

Q. Chen, J. Lacomis, E. J. Schwartz, C. Le Goues, G. Neubig, and B. Vasilescu, "Augmenting decompiler output with learned variable names and types," in 31st USENIX Security Symposium (USENIX Security 22), 2022, pp. 4327–4343.

# Challenge of Data Type/Naming Prediction

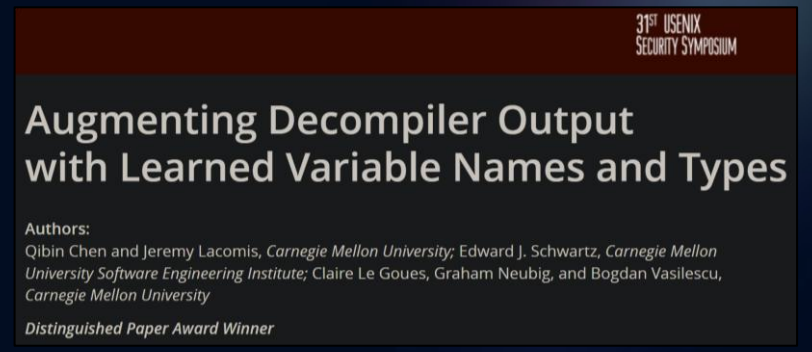





Q. Chen, J. Lacomis, E. J. Schwartz, C. Le Goues, G. Neubig, and B. Vasilescu, "Augmenting decompiler output with learned variable names and types," in 31st USENIX Security Symposium (USENIX Security 22), 2022, pp. 4327–4343.

# Challenge of Data Type/Naming Prediction

- Carnegie Mellon University - Software Engineering Institute @ USENIX'22
- Use tainted offsets of several properties of an income memory layout via Hex-Rays decompile, to predict the max probability known data-types



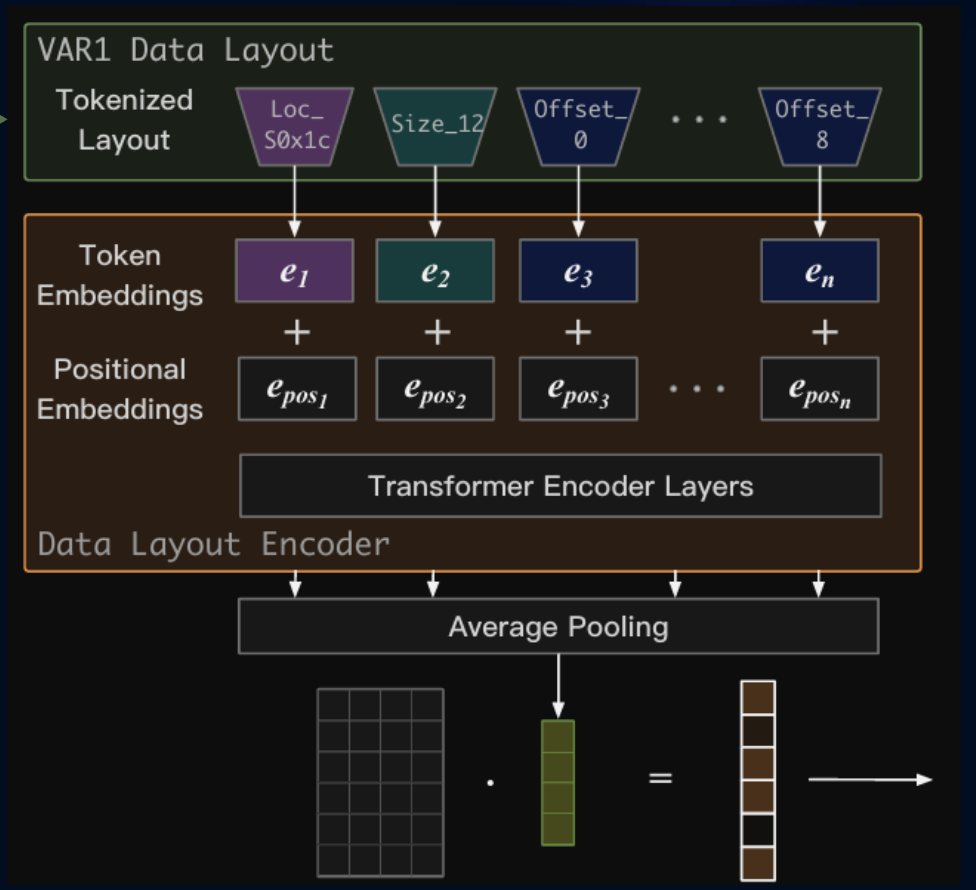

```
Data Layout
ID: VAR1
Size: 12
Loc: Stack 0x1c
Offsets: [0, 8]
```

Data Layout of Struct Generated by Hex-Rays

**Location:** A variable can be located either in registers (tokenized as `[Loc_<Register Name>]`) or on the stack (`[Loc_S<Offset>]`). E.g., a variable stored 28 bytes below the stack pointer is tokenized as `[Loc_S0x1c]`.

**Size:** Measured in bytes and tokenized as `[Size_<Size>]`.

**Internal Offsets:** The offsets of members of the type (either array elements or struct fields), in bytes. E.g., the type `int[2]` would have the offsets $\{0, 4\}$, while a `struct` with two `char` fields would have the offsets $\{0, 1\}$. These are tokenized as a sequence of `[Offset_<Offset>]`. For consistency, we also use `[Offset_0]` for types without substructure (i.e., scalar types like `int`).



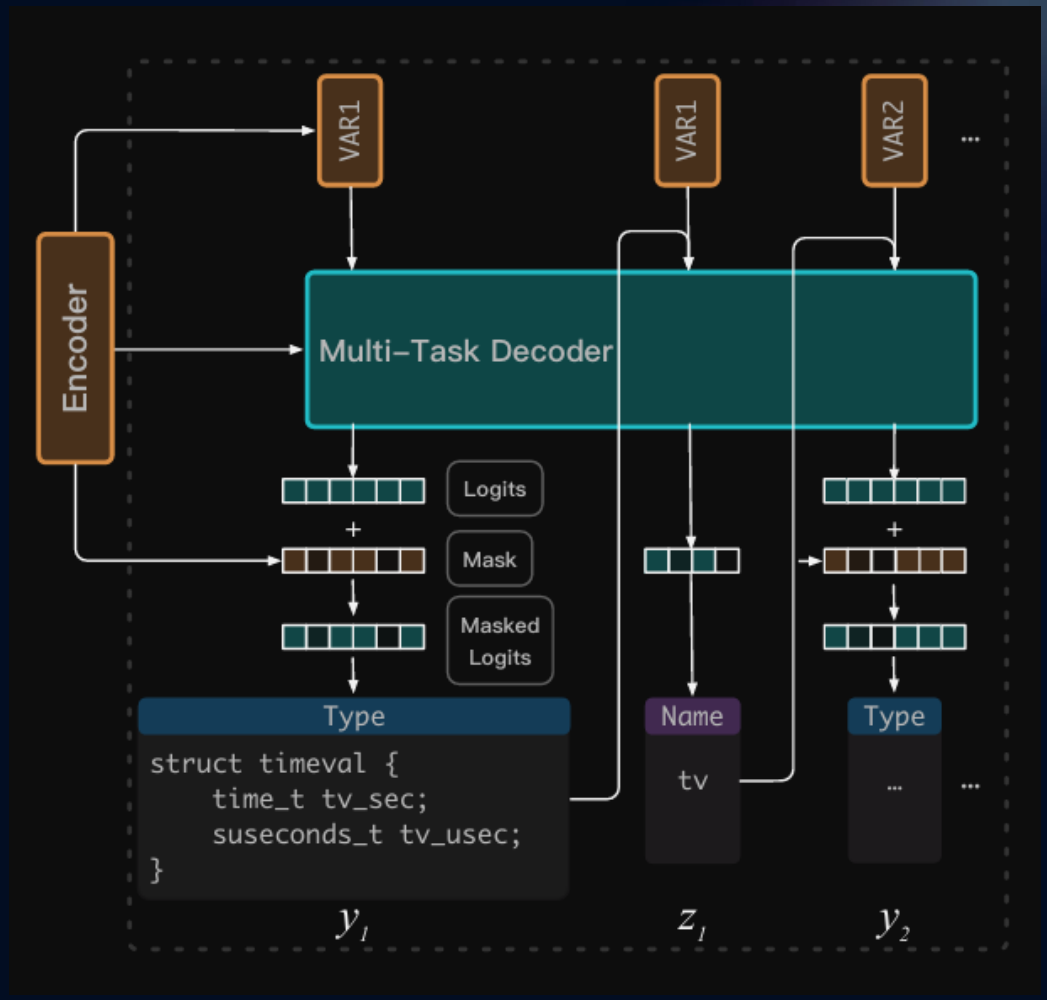



Q. Chen, J. Lacomis, E. J. Schwartz, C. Le Goues, G. Neubig, and B. Vasilescu, "Augmenting decompiler output with learned variable names and types," in 31st USENIX Security Symposium (USENIX Security 22), 2022, pp. 4327–4343.

# Results

1. Encode an data layout of income structure
2. Decoder predict the data type according to the layout
3. Then again, Decoder predict the variable name by the usage of this data type in such usage inside a binary function

```
int find_unused_picture(int a1, int a2, int a3) {
    int i, j, v1;
    if (a3) {
        for (i = <Num>;; ++i) {
        if (i > <Num>)
            goto LABEL_13;
        if (!*(*(<Num> * i + a2) + <Num>))
            break;
        }
        v1 = i;
    } else {
        for (j = <Num>;; ++j) {
        if (j > <Num>) {
        LABEL_13:
            av_log(a1, <Num>, <Str>);
            abort();
        }
        if (pic_is_unused(<Num> * j + a2))
            break;
        }
        v1 = j;
    }
    return v1;
}
```




| ID | Developer | DIRTY |
|---|---|---|
| a1 | AVCodecContext_0 *avctx | MpegEncContext_0 *s |
| a2 | Picture_0 *picture | Picture_0 *pic |
| a3 | int shared | int shared |
| v1 | int result | int result |

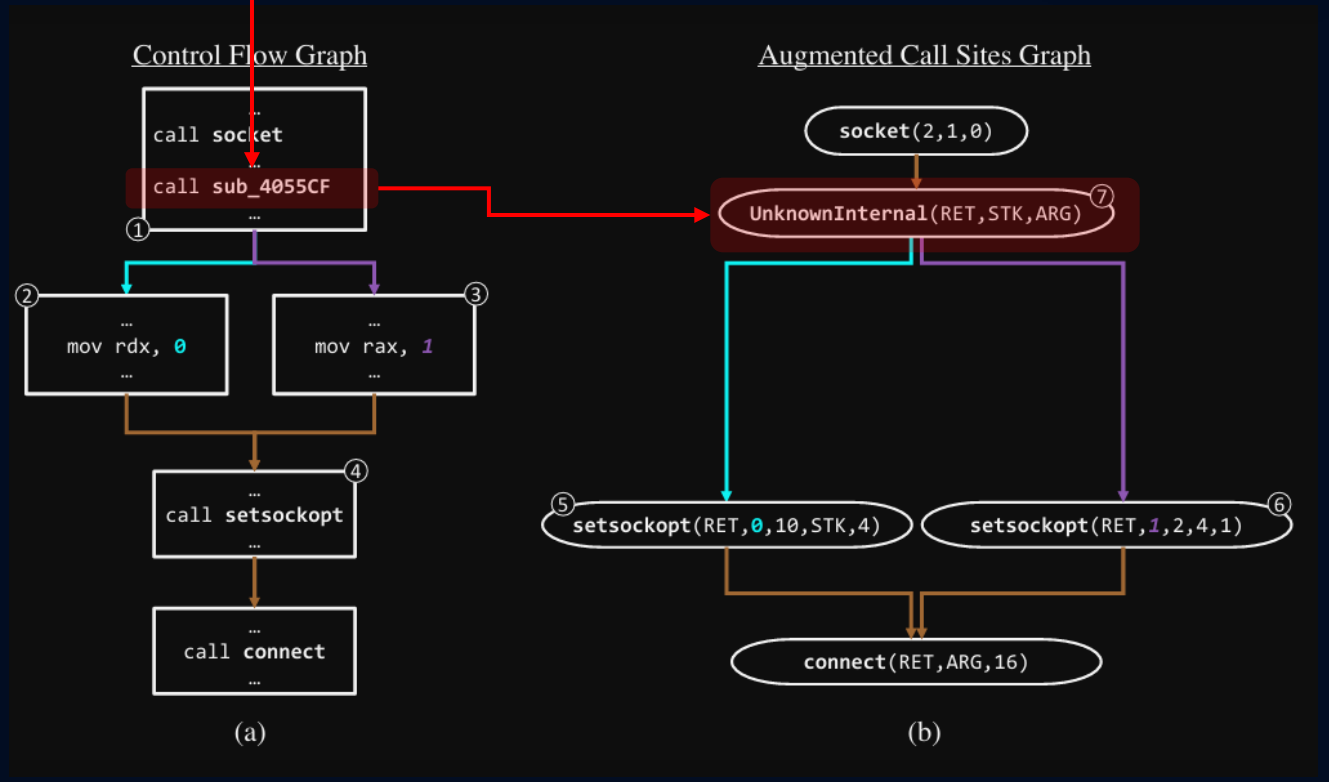


Q. Chen, J. Lacomis, E. J. Schwartz, C. Le Goues, G. Neubig, and B. Vasilescu, “Augmenting decompiler output with learned variable names and types,” in 31st USENIX Security Symposium (USENIX Security 22), 2022, pp. 4327–4343.

# More Work in Integrating Neural Networks into Cybersecurity

# Extending Research to Combat API Obfuscation

- Inspired by the works mentioned, our team presented a new concept at Black Hat USA 2024:
  - Inferring Windows APIs through Function Parameters
    - Exploits the fixed patterns of API parameters
    - **Strengthens obfuscated function reversing analysis**


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

# Takeaway

- From Linguistic Theory to Neural Machine Translation
  - Foundational linguistic theories: Chomsky (syntax), Firth (context)
  - For Transformer models, natural language translation has become nearly perfect

- Neural Reverse Engineering
  - Automatically infer function names, variable types, and variable names to significantly reduce manual reverse engineering efforts
  - Enhance the model's robustness in translating decompiled code to enable more effective automated reverse engineering
  - Summarizing function behavior can serve as a new technique for detecting malicious activity


txOne
networks