



— 0000001000111 —
PROJECT
LUPINE
— 0000000001111 —

**Harnessing Large Language Models for
Enhanced Malware Reverse Engineering**

Introduction

Who am I?

Jeremy (Dyngnosis) Richards

Started reversing microsoft patches for a company called nCircle in the mid 2000s, came up through BBS and IRC scene before that.

20 years of reversing/vuln research / exploit dev

5 years firmware / embedded research

5 years malware research (mobile / desktop)

Lead Malware Reverse Engineer at Arctic Wolf Labs.

👁 Intersection of AI/ML and Security 👁



Follow

dyngnosis ✓

@dyngnosis

machine learning, reversing engineering, malware, vulns, exploits, hardware, firmware, League of Legends.



Agenda

- 01** Introduce Problem & Goals
- 02** The Project
- 03** Results
- 04** Future work & Q/A



Introduction: The Problem

The Need for Automation in Reverse Engineering

Reverse engineering...

- Is labor-intensive and time-consuming task.
- requires a deep understanding of the programming languages, libraries, and frameworks used in the software
- requires the ability to recognize patterns and structures within the code.
- analyzing malicious code is a specialized task (obfuscation, anti-analysis, etc.)

Rise of the LLMs

Large Language Models have showcased their prowess in processing and generating text that mirrors human-like understanding, but can it understand decompiled code?



Introduction: Goals

Why are we here?

Project Objectives

The aim of this project is to enhance the reverse engineering experience, and improve the velocity and quality of analyst output by providing tools to automate the boring and illuminate the interesting..

Presentation Objectives

Encourage others to adopt the usage of Lupine and contribute the model

Show my method, mistakes, and successes when fine tuning a model

Provide the background necessary to start your own project focused on fine-tuning a local model to increase performance on a custom tasks



Project Objective

Project Goals

Fine-tune a local model to improve performance on the following three reverse engineering tasks:

1. Explain `<code>` step by step
2. Provide a summary for the following `<code>`
3. Supply a descriptive new name that describes the following `<code>`

Providing automation around these high level reversing tasks allows an analyst to identify interesting functions by name and summary.



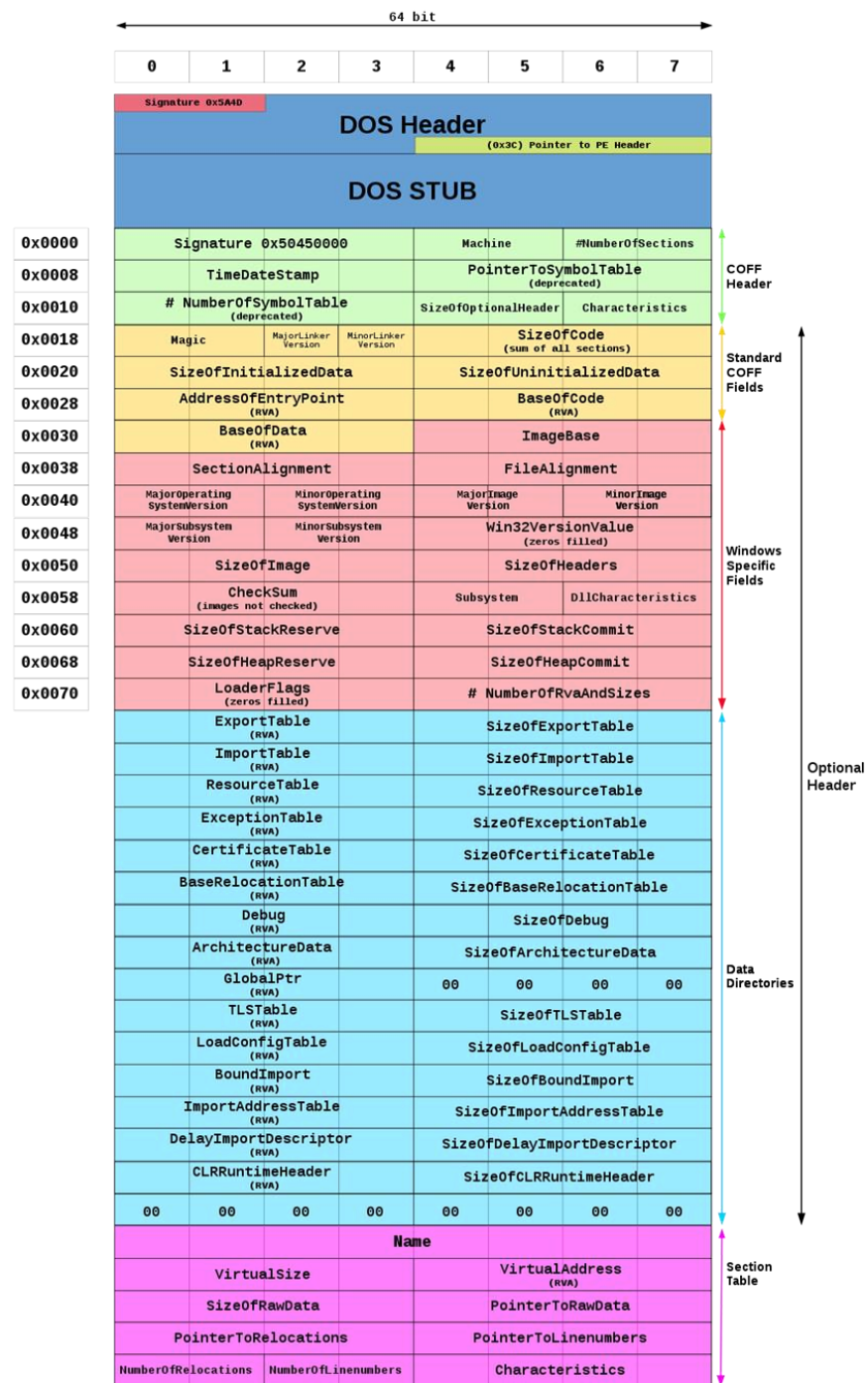
Introduction: Reverse Engineering

Reversing an unknown binary

When approaching the task of reverse engineering a binary, the analyst usually gets a little context but assuming there is none we:

1. Identify the file type & architecture (**PE32**, ELF, MACHO/x86, x64, arm)
2. PE Specific file parsing (headers, resources, imports, exports)
3. xref interesting API calls **to find interesting functions**
4. xref interesting strings **to find interesting functions**
5. reverse / trace execution from entry point **to find interesting functions**





Sections

File Edit Analysis Graph Navigation Search Select Tools Window Help

Program Trees

- c373d37b5a9427a18dbf93d519968d9fda04f2a262f424d0611830764c8cc69c
 - Headers
 - .flat
 - .text
 - .rdata
 - .data

Program Tree X

Symbol Tree

- Imports
- Exports
- Functions
- Labels
- Classes
- Namespaces

Filter:

Data Type Manager

- Data Types
 - BuiltInTypes
 - c373d37b5a9427a18dbf9
 - windows_vs12_32

Filter:

00417f3c	e8 4f d7	CALL	FUN_00415690
	ff ff		
00417f41	83 c4 10	ADD	ESP,0x10
00417f44	a3 b8 40	MOV	[DAT_004340b8],EAX
	43 00		
00417f49	5f	POP	EDI
00417f4a	5e	POP	ESI
00417f4b	c3	RET	
00417f4c	cc	??	CCh
00417f4d	cc	??	CCh
00417f4e	cc	??	CCh
00417f4f	cc	??	CCh

```
*****
*                                     *
*                               FUNCTION                               *
*                               *                                     *
*****
undefined __stdcall entry(void)
    assume FS_OFFSET = 0xffdf000
    AL:1    <RETURN>
entry
XREF[2]:   Entry Point(*), 00400140(*)

00417f50 81 ec 70    SUB     ESP,0x370
           03 00 00
00417f56 53          PUSH    EBX
00417f57 56          PUSH    ESI
00417f58 57          PUSH    EDI
```

Decompile: entry - (c373d37b5a9427a18dbf93d519968d9fda04f2a262f424d0611830764c8cc69c)

```
1
2 /* WARNING: Globals starting with '_' overlap smaller symbols at the same offset */
3
4 void entry(void)
5
6 {
7     int *piVar1;
8     code *pcVar2;
9     LPCWSTR pWVar3;
10    bool bVar4;
11    int iVar5;
12    LPWSTR pWVar6;
13    DWORD DVar7;
14    BOOL BVar8;
15    ushort *puVar9;
16    undefined3 extraout_var;
17    undefined4 uVar10;
18    HANDLE hHeap;
19    LPSTR lpMultiByteStr;
20    undefined4 extraout_ECX;
21    undefined4 extraout_ECX_00;
22    undefined4 extraout_ECX_01;
23    undefined4 extraout_ECX_02;
24    undefined4 extraout_ECX_03;
25    undefined4 extraout_ECX_04;
```

Console - Scripting

Function Call Trees: entry - (c373d37b5a9427a18dbf93d519968d9fda04f2a262f424d0611830764c8cc69c)

Incoming Calls

- Incoming References - entry

Filter:

Outgoing Calls

- Outgoing References - entry
 - memset
 - GetVersionExW
 - FUN_004010a4
 - FUN_00418d90
 - lstrcmplW

Filter:

00417f50

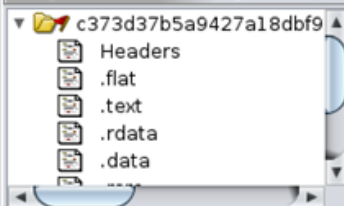
entry

SUB ESP,0x370

File Edit Analysis Graph Navigation Search Select Tools Window Help

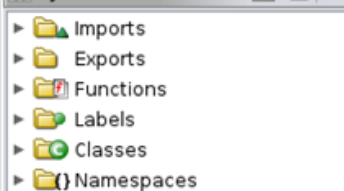


Program Trees



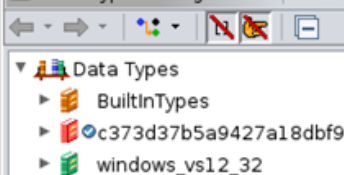
Program Tree X

Symbol Tree



Filter:

Data Type Manager



Filter:

Listing: c373d37b5a9427a18dbf93d519968d9fda04f2a262f424d0611830764c8cc69c

```
00417f3c e8 4f d7 CALL FUN_00415690
          ff ff
00417f41 83 c4 10 ADD ESP,0x10
00417f44 a3 b8 40 MOV [DAT_004340b8],EAX
          43 00
00417f49 5f POP EDI
00417f4a 5e POP ESI
00417f4b c3 RET
00417f4c cc ?? CCh
00417f4d cc ?? CCh
00417f4e cc ?? CCh
00417f4f cc ?? CCh
```

Assembly

```
*****
*                               *
*                               *
*****
```

```
undefined __stdcall entry(void)
          assume FS_OFFSET = 0xffdf000
          AL:1 <RETURN>
```

undefined

entry

XREF[2]: Entry Point(*), 00400140(*)

```
00417f50 81 ec 70 SUB ESP,0x370
          03 00 00
00417f56 53 PUSH EBX
00417f57 56 PUSH ESI
00417f58 57 PUSH EDI
```

Decompile: entry - (c373d37b5a9427a18dbf93d519968d9fda04f2a262f424d0611830764c8cc69c)

```
1
2 /* WARNING: Globals starting with '_' overlap smaller symbols at the same memory location */
3
4 void entry(void)
5
6 {
7     int *piVar1;
8     code *pcVar2;
9     LPCWSTR pWVar3;
10    bool bVar4;
11    int iVar5;
12    LPWSTR pWVar6;
13    DWORD DVar7;
14    BOOL BVar8;
15    ushort *puVar9;
16    undefined3 extraout_var;
17    undefined4 uVar10;
18    HANDLE hHeap;
19    LPSTR lpMultiByteStr;
20    undefined4 extraout_ECX;
21    undefined4 extraout_ECX_00;
22    undefined4 extraout_ECX_01;
23    undefined4 extraout_ECX_02;
24    undefined4 extraout_ECX_03;
25    undefined4 extraout_ECX_04;
```

Console - Scripting



Function Call Trees: entry - (c373d37b5a9427a18dbf93d519968d9fda04f2a262f424d0611830764c8cc69c)

Incoming Calls

f Incoming References - entry

Filter:

Outgoing Calls

```
f Outgoing References - entry
  f memset
  f GetVersionExW
  f FUN_004010a4
  f FUN_00418d90
  f lstrcmplW
```

Filter:

00417f50

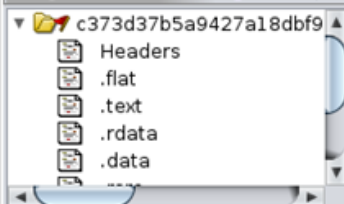
entry

SUB ESP,0x370

File Edit Analysis Graph Navigation Search Select Tools Window Help

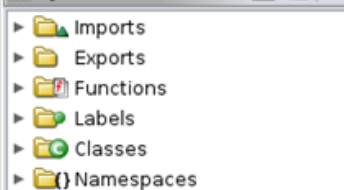


Program Trees



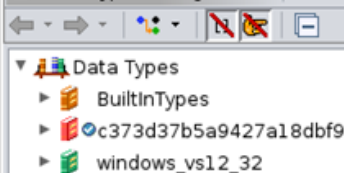
Program Tree X

Symbol Tree



Filter:

Data Type Manager



Filter:

Listing: c373d37b5a9427a18dbf93d519968d9fda04f2a262f424d0611830764c8cc69c

```
00417f3c e8 4f d7 CALL FUN_00415690
          ff ff
00417f41 83 c4 10 ADD ESP,0x10
00417f44 a3 b8 40 MOV [DAT_004340b8],EAX
          43 00
00417f49 5f POP EDI
00417f4a 5e POP ESI
00417f4b c3 RET
00417f4c cc ?? CCh
00417f4d cc ?? CCh
00417f4e cc ?? CCh
00417f4f cc ?? CCh
```

```
*****
*                               *
*                               *
*****
```

```
undefined __stdcall entry(void)
          assume FS_OFFSET = 0xffdf000
          AL:1 <RETURN>
```

undefined

entry

XREF[2]: Entry Point(*), 00400140(*)

```
00417f50 81 ec 70 SUB ESP,0x370
          03 00 00
00417f56 53 PUSH EBX
00417f57 56 PUSH ESI
00417f58 57 PUSH EDI
```

undefined FUN_00415690()

= ??

Decompiled Code

Decompile: entry - (c373d37b5a9427a18dbf93d519968d9fda04f2a262f424d0611830764c8cc69c)

```
1
2 /* WARNING: Globals starting with '_' overlap smaller symbols at the same offset */
3
4 void entry(void)
5 {
6
```

```
13
14 B00
15 uVar9;
16 uVar9;
17 uVar10;
18 hHeap;
19 lpMultiByteStr;
20 extraout_ECX;
21 extraout_ECX_00;
22 extraout_ECX_01;
23 extraout_ECX_02;
24 extraout_ECX_03;
25 extraout_ECX_04;
```

Console - Scripting



Function Call Trees: entry - (c373d37b5a9427a18dbf93d519968d9fda04f2a262f424d0611830764c8cc69c)

Incoming Calls

f Incoming References - entry

Filter:

Outgoing Calls

f Outgoing References - entry

```
f memset
f GetVersionExW
f FUN_004010a4
f FUN_00418d90
f lstrcmplW
```

Filter:

00417f50

entry

SUB ESP,0x370

File Edit Analysis Graph Navigation Search Select Tools Window Help

Program Trees

- c373d37b5a9427a18dbf9
 - Headers
 - .flat
 - .text
 - .rdata
 - .data

Progr

Sym

- Imports
- Exports
- Functions
- Labels
- Classes
- Namespaces

Filter:

Data Type Manager

- Data Types

- BuiltInTypes
- c373d37b5a9427a18dbf9
- windows_vs12_32

Filter:

Listing: c373d37b5a9427a18dbf93d519968d9fda04f2a262f424d0611830764c8cc69c

00417f3c	e8 4f d7	CALL	FUN_00415690
	ff ff		
00417f41	83 c4 10	ADD	ESP,0x10
00417f44	a3 b8 40	MOV	[DAT_004340b8],EAX
	43 00		
00417f49	5f	POP	EDI
00417f4a	5e	POP	ESI
00417f4b	c3	RET	
00417f4c	cc	??	CCh
00417f4d	cc	??	CCh
00417f4e	cc	??	CCh
00417f4f	cc	??	CCh

undefined

00417f50	81 ec 70	SUB	ESP,0x370
	03 00 00		
00417f56	53	PUSH	EBX
00417f57	56	PUSH	ESI
00417f58	57	PUSH	EDI

```
*****  
*                               *  
*               FUNCTION       *  
*****
```

```
undefined __stdcall entry(void)  
    assume FS_OFFSET = 0xffdf000  
    AL:1    <RETURN>
```

entry

XREF[2]: Entry Point(*), 00400140(*)

Console - Scripting

Function Call Trees: entry - (c373d37b5a9427a18dbf93d519968d9fda04f2a262f424d0611830764c8cc69c)

Incoming Calls

f Incoming References - entry

Filter:

Outgoing Calls

f Outgoing References - entry

- f memset
- f GetVersionExW
- f FUN_004010a4
- f FUN_00418d90
- f lstrcmplW

Filter:

Decompile: entry - (c373d37b5a9427a18dbf93d519968d9fda04f2a262f424d0611830764c8cc69c)

```
1  
2 /* WARNING: Globals starting with '_' overlap smaller symbols at the same offset */  
3  
4 void entry(void)  
5  
6 {  
7     int *piVar1;  
8     code *pcVar2;  
9     LPCWSTR pWVar3;  
10    bool bVar4;  
11    int iVar5;  
12    LPWSTR pWVar6;  
13    DWORD DVar7;  
14    BOOL BVar8;  
15    ushort *puVar9;  
16    undefined3 extraout_var;  
17    undefined4 uVar10;  
18    HANDLE hHeap;  
19    LPSTR lpMultiByteStr;  
20    undefined4 extraout_ECX;  
21    undefined4 extraout_ECX_00;  
22    undefined4 extraout_ECX_01;  
23    undefined4 extraout_ECX_02;  
24    undefined4 extraout_ECX_03;  
25    undefined4 extraout_ECX_04;
```

00417f50

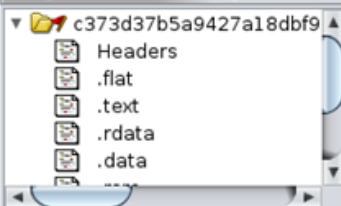
entry

SUB ESP,0x370

File Edit Analysis Graph Navigation Search Select Tools Window Help

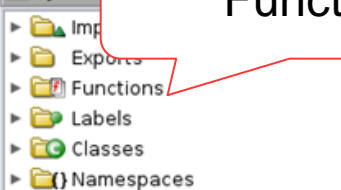


Program Trees



Program Tree X

Symbol

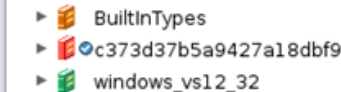


Filter:

Data Type Manager



Data Types



Filter:

Listing: c373d37b5a9427a18dbf93d519968d9fda04f2a262f424d0611830764c8cc69c

00417f3c	e8 4f d7	CALL	FUN_00415690	undefined FUN_00415690()
	ff ff			
00417f41	83 c4 10	ADD	ESP,0x10	
00417f44	a3 b8 40	MOV	[DAT_004340b8],EAX	= ??
	43 00			
00417f49	5f	POP	EDI	
00417f4a	5e	POP	ESI	
00417f4b	c3	RET		
00417f4c	cc	??	CCh	
00417f4d	cc	??	CCh	
00417f4e	cc	??	CCh	
00417f4f	cc	??	CCh	
***** * FUNCTION * *****				
		undefined __stdcall entry(void)		
		assume FS_OFFSET = 0xffdf000		
	AL:1	<RETURN>		
		entry		XREF[2]: Entry Point(*), 00400140(*)
00417f50	81 ec 70	SUB	ESP,0x370	
	03 00 00			
00417f56	53	PUSH	EBX	
00417f57	56	PUSH	ESI	
00417f58	57	PUSH	EDI	

Decompile: entry - (c373d37b5a9427a18dbf93d519968d9fda04f2a262f424d0611830764c8cc69c)

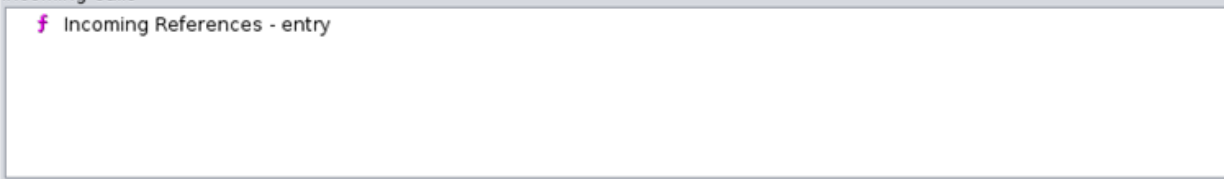
```
1
2 /* WARNING: Globals starting with '_' overlap smaller symbols at the same memory location */
3
4 void entry(void)
5 {
6     int *piVar1;
7     code *pcVar2;
8     LPCWSTR pWVar3;
9     bool bVar4;
10    int iVar5;
11    LPWSTR pWVar6;
12    DWORD DVar7;
13    BOOL BVar8;
14    ushort *puVar9;
15    undefined3 extraout_var;
16    undefined4 uVar10;
17    HANDLE hHeap;
18    LPSTR lpMultiByteStr;
19    undefined4 extraout_ECX;
20    undefined4 extraout_ECX_00;
21    undefined4 extraout_ECX_01;
22    undefined4 extraout_ECX_02;
23    undefined4 extraout_ECX_03;
24    undefined4 extraout_ECX_04;
25 }
```

Console - Scripting



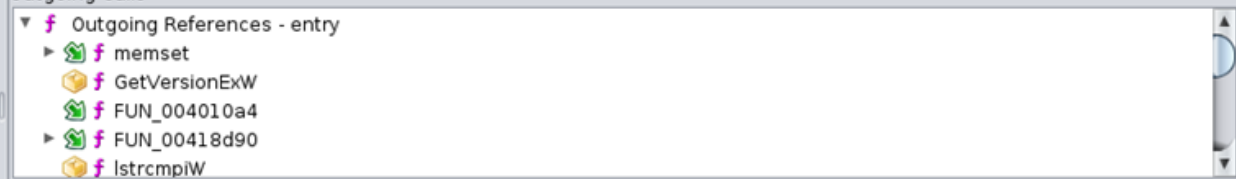
Function Call Trees: entry - (c373d37b5a9427a18dbf93d519968d9fda04f2a262f424d0611830764c8cc69c)

Incoming Calls



Filter:

Outgoing Calls



Filter:

00417f50

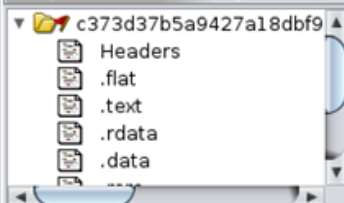
entry

SUB ESP,0x370

File Edit Analysis Graph Navigation Search Select Tools Window Help

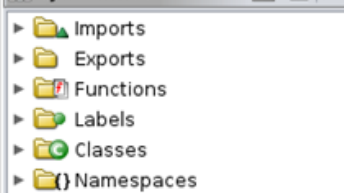


Program Trees



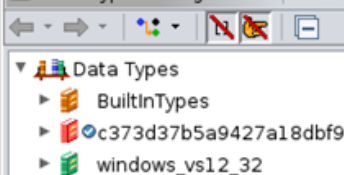
Program Tree X

Symbol Tree



Filter:

Data Type Manager



Filter:

Listing: c373d37b5a9427a18dbf93d519968d9fda04f2a262f424d0611830764c8cc69c

```
00417f3c e8 4f d7 CALL FUN_00415690
          ff ff
00417f41 83 c4 10 ADD ESP,0x10
00417f44 a3 b8 40 MOV [DAT_004340b8],EAX
          43 00
00417f49 5f POP EDI
00417f4a 5e POP ESI
00417f4b c3 RET
00417f4c cc ?? CCh
00417f4d cc ?? CCh
00417f4e cc ?? CCh
00417f4f cc ?? CCh
```

```
*****
*                               *
*                               *
*****
```

```
undefined __stdcall entry(void)
          assume FS_OFFSET = 0xffdff000
          AL:1 <RETURN>
```

undefined

entry

XREF[2]: Entry Point(*), 00400140(*)

```
00417f50 81 ec 70 SUB ESP,0x370
          03 00 00
00417f56 53 PUSH EBX
00417f57 56 PUSH ESI
00417f58 57 PUSH EDI
```

undefined FUN_00415690(u

= ??

Decompile: entry - (c373d37b5a9427a18dbf93d519968d9fda04f2a262f424d0611830764c8cc69c)

```
1
2 /* WARNING: Globals starting with '_' overlap smaller symbols at the same memory location */
3
4 void entry(void)
5
6 {
7     int *piVar1;
8     code *pcVar2;
9     LPCWSTR pWVar3;
10    bool bVar4;
11    int iVar5;
12    LPWSTR pWVar6;
13    DWORD DVar7;
14    BOOL BVar8;
15    ushort *puVar9;
16    undefined3 extraout_var;
17    undefined4 uVar10;
18    HANDLE hHeap;
19    LPSTR lpMultiByteStr;
20    undefined4 extraout_ECX;
21    undefined4 extraout_ECX_00;
22    undefined4 extraout_ECX_01;
23    undefined4 extraout_ECX_02;
24    undefined4 extraout_ECX_03;
25    undefined4 extraout_ECX_04;
```

Console - Scripting



Function Call Trees: entry - (c373d37b5a9427a18dbf93d519968d9fda04f2a262f424d0611830764c8cc69c)

Incoming Calls

f Incoming References - entry

Filter:

Calling functions

Outgoing Calls

```
f Outgoing References - entry
  f memset
  f GetVersionExW
  f FUN_004010a4
  f FUN_00418d90
  f lstrcmplW
```

Filter:

00417f50

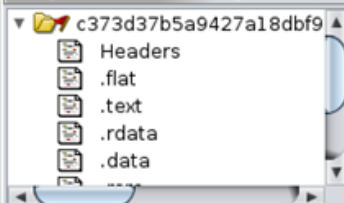
entry

SUB ESP,0x370

File Edit Analysis Graph Navigation Search Select Tools Window Help

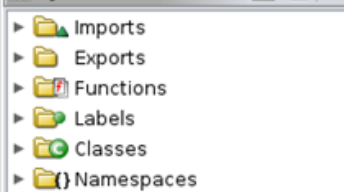


Program Trees



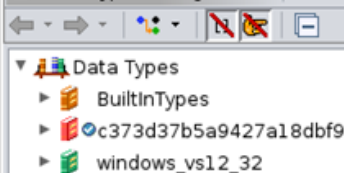
Program Tree X

Symbol Tree



Filter:

Data Type Manager



Filter:

Listing: c373d37b5a9427a18dbf93d519968d9fda04f2a262f424d0611830764c8cc69c

```
00417f3c e8 4f d7 CALL FUN_00415690
          ff ff
00417f41 83 c4 10 ADD ESP,0x10
00417f44 a3 b8 40 MOV [DAT_004340b8],EAX
          43 00
00417f49 5f POP EDI
00417f4a 5e POP ESI
00417f4b c3 RET
00417f4c cc ?? CCh
00417f4d cc ?? CCh
00417f4e cc ?? CCh
00417f4f cc ?? CCh
```

```
*****
*                               *
*                               *
*****
```

```
undefined __stdcall entry(void)
          assume FS_OFFSET = 0xffdf000
          AL:1 <RETURN>
```

undefined

entry

XREF[2]: Entry Point(*), 00400140(*)

```
00417f50 81 ec 70 SUB ESP,0x370
          03 00 00
00417f56 53 PUSH EBX
00417f57 56 PUSH ESI
00417f58 57 PUSH EDI
```

undefined FUN_00415690(u

= ??

Decompile: entry - (c373d37b5a9427a18dbf93d519968d9fda04f2a262f424d0611830764c8cc69c)

```
1
2 /* WARNING: Globals starting with '_' overlap smaller symbols at the same memory location */
3
4 void entry(void)
5
6 {
7     int *piVar1;
8     code *pcVar2;
9     LPCWSTR pWVar3;
10    bool bVar4;
11    int iVar5;
12    LPWSTR pWVar6;
13    DWORD DVar7;
14    BOOL BVar8;
15    ushort *puVar9;
16    undefined3 extraout_var;
17    undefined4 uVar10;
18    HANDLE hHeap;
19    LPSTR lpMultiByteStr;
20    undefined4 extraout_ECX;
21    undefined4 extraout_ECX_00;
22    undefined4 extraout_ECX_01;
23    undefined4 extraout_ECX_02;
24    undefined4 extraout_ECX_03;
25    undefined4 extraout_ECX_04;
```

Console - Scripting



Function Call Trees: entry - (c373d37b5a9427a18dbf93d519968d9fda04f2a262f424d0611830764c8cc69c)

Incoming Calls

f Incoming References - entry

Filter:

Outgoing Calls

```
f Outgoing References - entry
  f memset
  f GetVersionExW
  f FUN_004010a4
  f FUN_00418d90
  f lstrcmplW
```

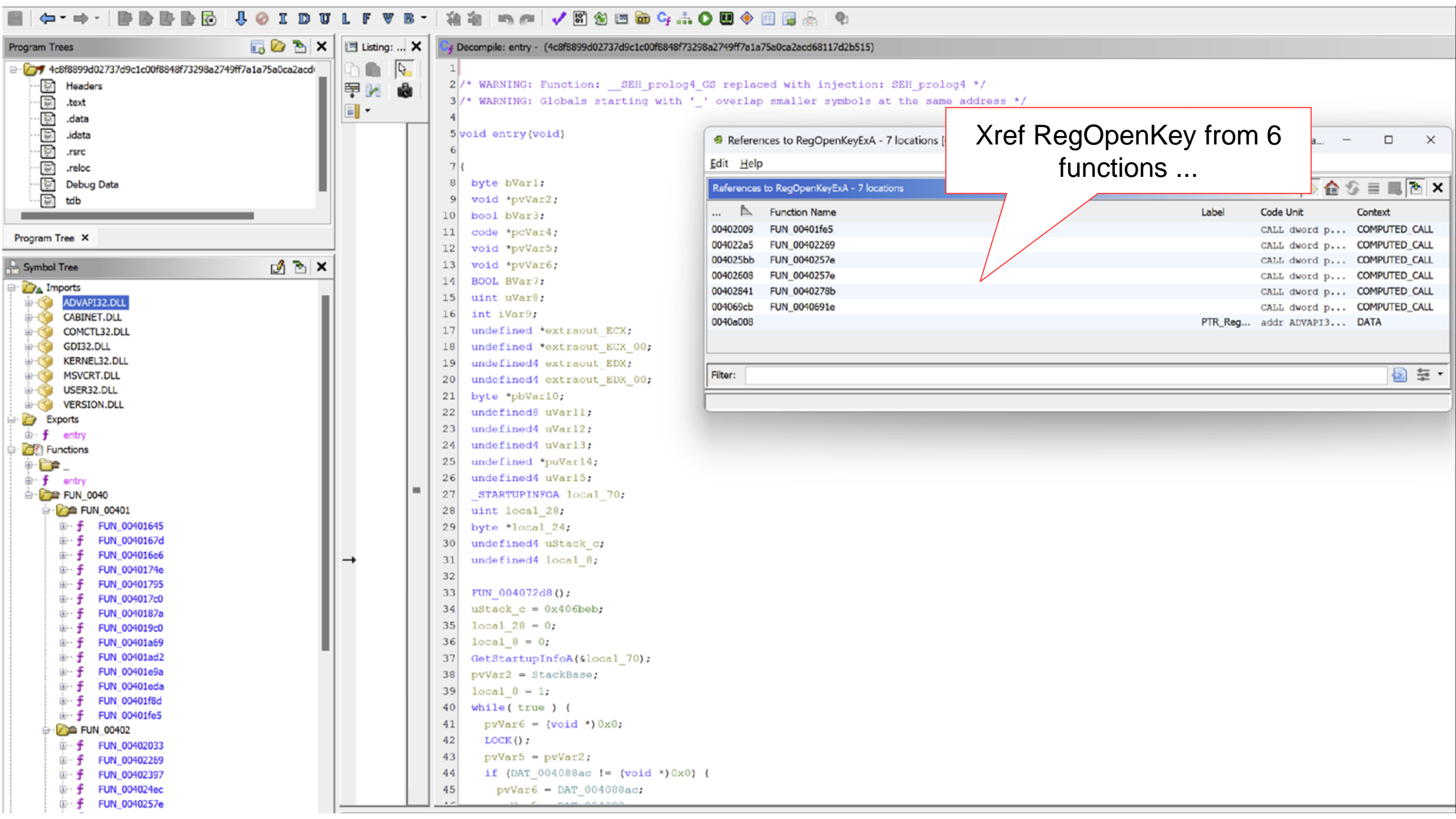
Filter:

Functions called

00417f50

entry

SUB ESP,0x370



```
1
2 /* WARNING: Globals starting with '_' overlap smaller symbols at the same address */
3
4 void entry(void)
5 {
6     int *piVar1;
7     code *pcVar2;
8     LPCTSTR pWVar3;
9     bool bVar4;
10    int iVar5;
11    LPWSTR pWVar6;
12    DWORD DVar7;
13    BOOL BVar8;
14    ushort *puVar9;
15    undefined3 extraout_var;
16    undefined4 uVar10;
17    HANDLE hHeap;
18    LPSTR lpMultiByteStr;
19    undefined4 extraout_ECX;
20    undefined4 extraout_ECX_00;
21    undefined4 extraout_ECX_01;
22    undefined4 extraout_ECX_02;
23    undefined4 extraout_ECX_03;
24    undefined4 extraout_ECX_04;
25    undefined4 extraout_ECX_05;
26    undefined4 extraout_ECX_06;
27    undefined4 extraout_ECX_07;
28    undefined4 extraout_ECX_08;
29    undefined4 extraout_ECX_09;
30    undefined4 extraout_ECX_10;
31    undefined4 extraout_ECX_11;
32    undefined4 extraout_ECX_12;
33    undefined4 extraout_ECX_13;
34    int iVar11;
35    undefined4 extraout_ECX_14;
36    undefined4 extraout_ECX_15;
37    undefined4 extraout_ECX_16;
38    undefined4 extraout_ECX_17;
39    undefined4 extraout_ECX_18;
40    undefined4 extraout_ECX_19;
41    undefined4 extraout_EDX;
42    uint uVar12;
43    undefined4 extraout_EDX_00;
44    undefined4 extraout_EDX_01;
45    undefined4 extraout_EDX_02;
46    char *pcVar13;
47    uint uVar14;
48    int *piVar15;
49    undefined8 uVar16;
50    SIZE_T dwBytes;
51    undefined *puVar17;
52    undefined *puStack_370;
53    undefined auStack_36c [4];
54    undefined auStack_368 [16];
55    undefined auStack_358 [16];
56    void *pvStack_348;
57    WCHAR aWStack_334 [8];
58    undefined auStack_324 [20];
59    WCHAR aWStack_310 [2];
60    undefined auStack_30c [2];
61    undefined auStack_30a [518];
62    undefined auStack_104 [248];
63    undefined4 uStack_c;
64
65    memset(&DAT_004340e4,0,0x118);
66    _DAT_004340e0 = 0x11c;
67    GetVersionExW((LPOSVERSIONINFO) &DAT_004340e0);
68    uVar16 = FUN_004010a4(extraout_ECX,extraout_EDX,0x42f0fc,0xd,(undefined *) &DAT_0042f47c,3);
69    uVar16 = FUN_004010a4(extraout_ECX_00,(int)((ulonglong)uVar16 >> 0x20),0x42f14c,9,
70        (undefined *) &DAT_0042f47c,3);
71    _DAT_00436f80 = DAT_0042f510;
72    DAT_0042f510 = DAT_0042f514;
73    _DAT_0042f518 = 0;
74    DAT_0042f514 = 0;
75    DAT_0042f514 = 0;
76    DAT_00436920 = 0;
77    uVar16 = FUN_004010a4(extraout_ECX_01,(int)((ulonglong)uVar16 >> 0x20),0x42f138,0xd,
78        (undefined *) &DAT_0042f47c,3);
79    _DAT_00436f80 = DAT_0042f510;
80    DAT_0042f510 = DAT_0042f514;
81    _DAT_0042f518 = 0;
82    DAT_0042f514 = 0;
83    uVar16 = FUN_004010a4(extraout_ECX_02,(int)((ulonglong)uVar16 >> 0x20),0x42f418,0xd,
84        (undefined *) &DAT_0042f47c,3);
85    _DAT_00436f80 = DAT_0042f510;
86    DAT_0042f510 = DAT_0042f514;
87    _DAT_0042f518 = 0;
```

```
87    _DAT_0042f518 = 0;
88    DAT_0042f514 = 0;
89    DAT_00436920 = 0;
90    uVar16 = FUN_004010a4(extraout_ECX_03,(int)((ulonglong)uVar16 >> 0x20),0x42f110,0xd,
91        (undefined *) &DAT_0042f47c,3);
92    uVar12 = (uint)((ulonglong)uVar16 >> 0x20);
93    for (uVar14 = DAT_00436920; 9 < uVar14; uVar14 = uVar14 % 10) {
94        uVar12 = uVar14 / 10;
95    }
96    DAT_00436920 = uVar14;
97    FUN_004010a4(extraout_ECX_04,uVar12,0x42f468,0xb,(undefined *) &DAT_0042f47c,3);
98    FUN_00418d90(extraout_ECX_05,uVar14);
99    _DAT_00436f80 = DAT_0042f510;
100   DAT_0042f510 = DAT_0042f514;
101   _DAT_0042f518 = 0;
102   DAT_0042f514 = 0;
103   DAT_00436920 = 0;
104   uVar16 = FUN_004010a4(extraout_ECX_06,extraout_EDX_00,0x42f124,0xd,(undefined *) &DAT_0042f47c,3);
105   uVar16 = FUN_004010a4(extraout_ECX_07,(int)((ulonglong)uVar16 >> 0x20),0x42f010,0xc,
106       (undefined *) &DAT_0042f47c,3);
107   uVar16 = FUN_004010a4(extraout_ECX_08,(int)((ulonglong)uVar16 >> 0x20),0x42f42c,0xc,
108       (undefined *) &DAT_0042f47c,3);
109   FUN_004010a4(extraout_ECX_09,(int)((ulonglong)uVar16 >> 0x20),0x42f024,0xb,
110       (undefined *) &DAT_0042f47c,3);
111   FUN_00418d90(extraout_ECX_10,3);
112   _DAT_00436f80 = 0;
113   DAT_0042f510 = 0;
114   _DAT_0042f518 = 0;
115   DAT_0042f514 = 0;
116   DAT_00436920 = 0;
117   FUN_004010a4(extraout_ECX_11,extraout_EDX_01,0x42f440,0xc,(undefined *) &DAT_0042f47c,3);
118   FUN_00418d90(extraout_ECX_12,0xf);
119   FUN_004010a4(extraout_ECX_13,extraout_EDX_02,0x42f0e8,0xc,(undefined *) &DAT_0042f47c,3);
120   puStack_370 = DAT_0042f47c;
121   DAT_0042fee0 = FUN_00415690(0x2ba92ec1,&DAT_0042f14c,0xc);
122   uVar12 = 0;
123   uVar14 = uVar12;
124   do {
125       for (uVar12 < 5; uVar12 = uVar12 + 1) {
126           if (uVar14 != uVar12) {
127               auStack_36c = (undefined [4])0x0;
128           }
129       }
130       uVar12 = uVar14 + 1;
131       uVar14 = uVar12;
132   } while (uVar12 < 5);
133   piVar1 = *(int **)((int *)((int)ProcessEnvironmentBlock + 0xc) + 0x10);
134   piVar15 = piVar1;
135   do {
136       piVar15 = (int *)piVar15;
137       if ((piVar15[6] != 0) && (iVar5 = lstrcpw((LPCTSTR)piVar15[0xc],L"kernel32.dll"), iVar5 == 0))
138           break;
139   } while (piVar1 != piVar15);
140   _DAT_00436f0c = (code *)FUN_00415700(piVar15[6],0x1f3ffddb,0xd,0x7d0e76b);
141   iVar5 = (*_DAT_00436f0c)(&DAT_0042f0fc);
142   _DAT_00434254 = (code *)FUN_00415700(iVar5,0x4dd730b7,0xf,0x36ca1fd);
143   DAT_0042f550 = (*_DAT_00434254)();
144   iVar5 = 5;
145   do {
146       puStack_370 = (undefined *)((uint)puStack_370 ^ 0x4a108947);
147       iVar5 = iVar5 + -1;
148   } while (iVar5 != 0);
149   piVar1 = *(int **)((int *)((int)ProcessEnvironmentBlock + 0xc) + 0x10);
150   piVar15 = piVar1;
151   do {
152       piVar15 = (int *)piVar15;
153       if ((piVar15[6] != 0) && (iVar5 = lstrcpw((LPCTSTR)piVar15[0xc],L"kernel32.dll"), iVar5 == 0))
154           break;
155   } while (piVar1 != piVar15);
156   _DAT_00436f0c = (code *)FUN_00415700(piVar15[6],0x1f3ffddb,0xd,0x7d0e76b);
157   iVar5 = (*_DAT_00436f0c)(&DAT_0042f14c);
158   DAT_00434214 = FUN_00415700(iVar5,0xc9a2a74c,0x10,0x5bfa915f);
159   piVar6 = GetCommandLine();
160   DAT_0042fee4 = (code *)FUN_00415690(0x9b4c2d0,&DAT_0042f0e8,0x13);
161   (*_DAT_0042fee4)(pWVar6,auStack_368);
162   if (((S < _DAT_004340e4) && ((int)puStack_370 < 3)) && (iVar5 = FUN_00417540()), iVar5 != 0) &&
163       (iVar5 = FUN_00419b00(), iVar5 != 0)) {
164       DAT_00434214 = (code *)FUN_00415690(0x2a13bb3,&DAT_0042f418,0xe);
165       (*DAT_00434214)(0x80000001,L"Software\\Microsoft\\Windows\\CurrentVersion\\Run",0,0xf003f,
166           &puStack_370);
167       DVar7 = GetModuleFileNameW((HMODULE)0x0,aWStack_310,0x104);
168       if (DVar7 == 0) {
169           FUN_0041b980(0);
170       }
171   } else {
172       iVar5 = 0;
173       iVar11 = 0;
```

```
191    DAT_00434224 = (code *)FUN_00415690(0x2a5ebfb7,&DAT_0042f418,0xc);
192    (*_DAT_00434224)(puVar17);
193    BVar8 = ShellExecuteExW((SHELLEXECUTEINFO) &auStack_36c);
194    if (BVar8 != 0) {
195        /* WARNING: Subroutine does not return */
196        ExitProcess(0x65);
197    }
198    }
199    }
200    DAT_00436928 = (LPCTSTR)FUN_00415610(0x80);
201    DAT_0042fef4 = (code *)FUN_00415690(0xaff20ccb,&DAT_0042f0fc,9);
202    (*DAT_0042fef4)(auStack_104,&DAT_00425170);
203    puVar9 = FUN_0041c220();
204    uVar12 = 0;
205    do {
206        wprintfw(aWStack_334,L"%x",(uint)puVar9[uVar12]);
207        uVar12 = uVar12 + 1;
208    } while (uVar12 < 5);
209    if (5 < _DAT_004340e4) {
210        FUN_0041e130();
211    }
212    pWVar3 = DAT_00436928;
213    _DAT_004342a8 = (code *)FUN_00415690(0x76a4fa0b,&DAT_0042f0fc,0xb);
214    iVar5 = (*_DAT_004342a8)(0x1f0001,0,pWVar3);
215    pWVar3 = DAT_00436928;
216    if (iVar5 != 0) {
217        /* WARNING: Subroutine does not return */
218        ExitProcess(0);
219    }
220    _DAT_00434240 = (code *)FUN_00415690(0xed3e7aaa,&DAT_0042f0fc,0xd);
221    (*_DAT_00434240)(0,0,pWVar3);
222    FUN_00402030();
223    FUN_00415690(0xa38e7de7,&DAT_0042f0fc,0x1b);
224    FUN_00415690(0x97622898,&DAT_0042f0fc,10);
225    bVar4 = FUN_004192e0();
226    iVar5 = CONCAT31(extraout_var,bVar4);
227    if (iVar5 == 0) {
228        return;
229    }
230    _DAT_0042f880 = (code *)FUN_00415690(0x48b64e68,&DAT_0042f0fc,0x12);
231    uVar10 = (*_DAT_0042f880)();
232    _DAT_004342b0 = (code *)FUN_00415690(0xf0df01cc,&DAT_0042f0fc,0x11);
233    (*_DAT_004342b0)(uVar10,0x80);
234    PTR_DAT_0042f038 = (undefined *)FUN_00415650(0x100);
235    wprintfw((LPWSTR)PTR_DAT_0042f038,L"%s.%s",_DAT_00436928,L"anobtanium@tutanota.com",L".pandora"
236        );
237    iVar11 = FUN_00419b00();
238    if (iVar11 != 0) {
239        FUN_004198a0(&DAT_0042f160,0x2b5);
240    }
241    iVar11 = lstrlen((LPCTSTR)PTR_DAT_0042f038);
242    DAT_00436930 = iVar11 * 2 + 2;
243    DAT_00434200 = FUN_00415690(0xac74699e,&DAT_0042f0fc,0xc);
244    DAT_0043692c = (LPCTSTR)FUN_00415650(0x1fa0);
245    FUN_00415610(0x1000);
246    dwBytes = 0x100;
247    DVar7 = 8;
248    hHeap = GetProcessHeap();
249    lpMultiByteStr = (LPSTR)HeapAlloc(hHeap,DVar7,dwBytes);
250    if (iVar5 == 1) {
251        FUN_0041bc60();
252        FUN_0041be00(extraout_ECX_15,&DAT_00436720);
253        pcVar13 = "<br><strong><p></div></body>";
254        uVar10 = extraout_ECX_16;
255    }
256    else {
257        uVar10 = extraout_ECX_14;
258        if (iVar5 != 3) goto LAB_004186f9;
259        DVar7 = 0;
260        WideCharToMultiByte(0,0,DAT_00436928,-1,lpMultiByteStr,0x80,(LPCSTR)0x0,(LPBOOL)0x0);
261        FUN_0041bc60();
262        FUN_0041be00(extraout_ECX_17,lpMultiByteStr);
263        pcVar13 = "<br><strong><p></div></body>";
264        uVar10 = extraout_ECX_18;
265    }
266    FUN_0041be00(uVar10,pcVar13);
267    uVar10 = extraout_ECX_19;
268    LAB_004186f9:
269    FUN_0041c0b0(uVar10,uStack_c,DVar7);
270    FUN_00417d90();
271    DAT_00436934 = lstrlenA(DAT_0043692c);
272    FUN_0041e450();
273    pcVar2 = (code *)swi(3);
274    (*pcVar2)();
275    return;
276 }
277
```

Introduction:

Potential of LLMs in Reverse Engineering Malware

Solution:

We hypothesize that with appropriate base model, fine-tuning, and a quality dataset, LLMs can significantly contribute to malware reverse engineering. They might automate the recognition of malicious code patterns, extraction of embedded constants, identification of malware signatures, and even suggest potential detection measures. This accelerates the malware analysis process and lowers the entry barrier for individuals venturing into this critical cybersecurity domain.

Today, I'll unfold my journey of constructing a dataset and a model, developing a plugin, and a plugin server, and amalgamating these elements to probe the potential of Large Language Models in the realm of malware reverse engineering.



Literature Review (tools)

G-3PO: A Protocol Droid for Ghidra: An AI assistant developed by Olivia Lucca Fraser at Tenable for analysing and annotating decompiled code in Ghidra, which queries OpenAI and/or Anthropic's language models. See this writeup on the [Tenable tech blog](#) for details.

Gepetto: An IDA Pro plugin that queries GPT models for explanatory comments and meaningful variable names (like G-3PO for IDA Pro). Developed by Ivan Kwiatkowski.

ai for Pwndbg: Your trusty AI debugging sidekick, developed by Olivia Lucca Fraser at Tenable as a Pwndbg command.

ai for GEF: Same as above, but implemented as a GEF command. Developed by Olivia Lucca Fraser at Tenable.

GPT-WPRE: Whole-program Reverse Engineering with GPT-3. This is a little toy prototype of a tool that attempts to summarize a whole binary using GPT-3 (specifically the text-davinci-003 model), based on decompiled code provided by Ghidra. Developed by Brendan Dolan-Gavitt.

IATelligence: IATelligence is a Python script that extracts the Import Address Table (IAT) from a PE file and uses OpenAI's GPT-3 model to provide details about each Windows API imported by the file. The script also searches for related MITRE ATT&CK techniques and explains how the API could potentially be used by attackers. Developed by Thomas Roccia.



Literature Review (models)

Code Llama: Code Llama is a family of large language models for code based on Llama 2 providing state-of-the-art performance among open models, infilling capabilities, support for large input contexts, and zero-shot instruction following ability for programming tasks. (7B, 13, 34B)

Starcoder: The StarCoder models are 15.5B parameter models trained on 80+ programming languages

CodeAlpaca: The Code Alpaca models are fine-tuned from a 7B and 13B LLaMA model on 20K instruction-following data

replit-code-v1.5: Replit Code v1.5 is a 3.3B parameter Causal Language Model focused on Code Completion.



Literature Review (papers)

[LmPa](#): Improving Decompilation by Synergy of Large Language Model and Program Analysis

DIRTY [Usenix 2022 Conference Video](#): Augmenting Decompiler Output with Learned Variable Names and Types

[Code](#) | [Paper PDF](#) | [Demo](#)

```
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

Large Language models

How do they work?

1. **Input Embeddings:** Text sequences are converted into vectors. These vectors hold the meaning and context of words.

2. **Attention Mechanism:** Determines which parts of the input are relevant.

Weights importance of words based on context.

Allows for capturing long-range dependencies in text.

3. **Encoder-Decoder Stacks:** Processes and transforms input embeddings.

Multiple layers of encoders capture the context.

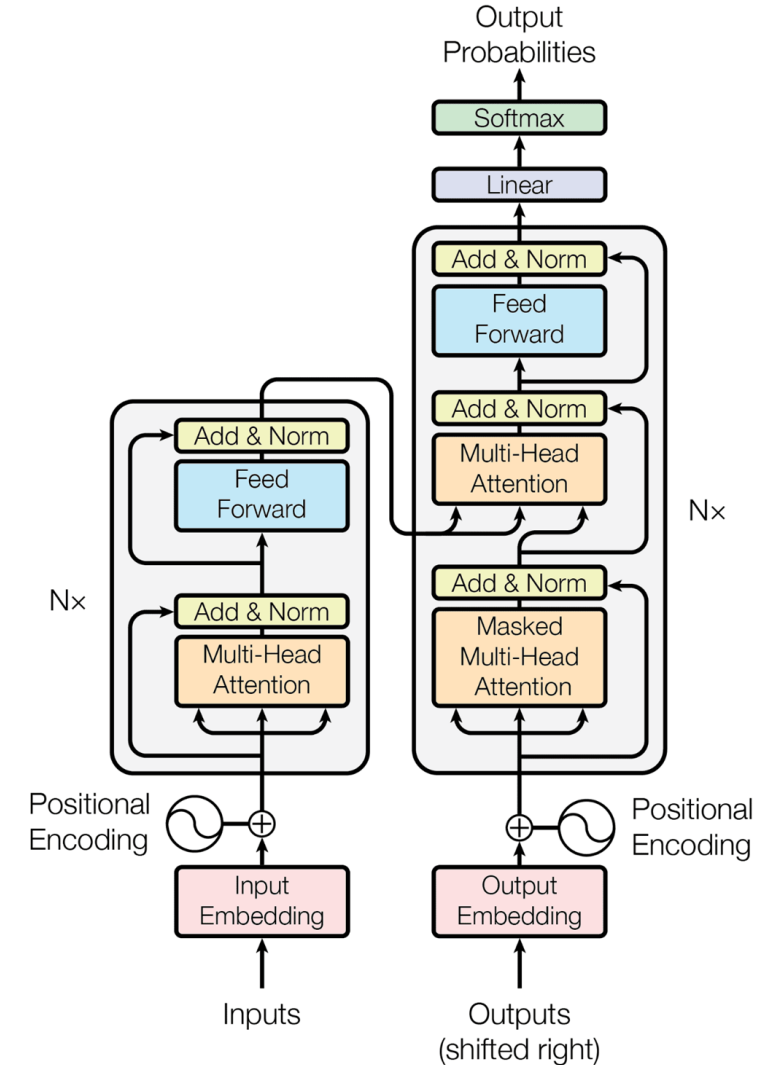
Decoders generate the output, be it text translation, summarization, or other tasks.

4. **Positional Encoding:** Ensures sequence order is maintained.

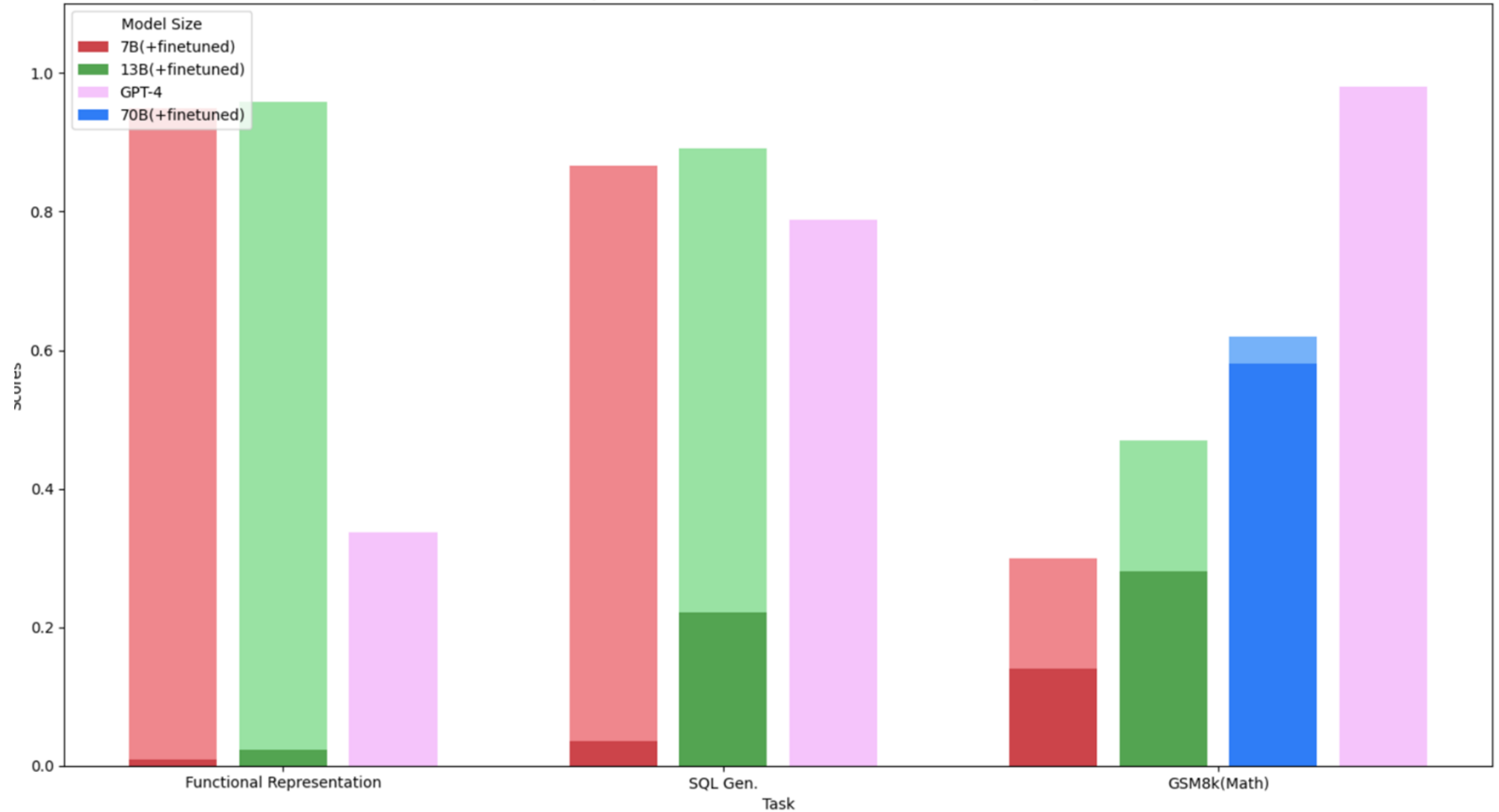
Adds information about the position of each word in the sequence.

[Transformers United 2023: Introduction to Transformers w/ Andrej Karpathy](#)

[Neural Networks: Zero to Hero](#)



Scores by task and model size before and after fine-tuning



Methodology

How it's done

- (1) Generate a dataset of decompiled functions paired with the answers to the questions we will ask.
- (2) Develop a set of prompts (questions) whose answers generate the data we require.
- (3) Select and fine tune an existing model.
- (4) Deploy webserver that loads the model and provides inference.
- (5) Provide IDE integration to send & receive code comments and function names for analysis & update IDE
- (6) Provide feedback mechanism for Analysts to make suggestions
- (7) Retrain / Test / Announce / Release



Methodology

(1) Generate a dataset of decompiled functions that includes the following:

0. decompiled code
1. descriptive name
2. short summary
3. step-by-step description.



Methodology

Develop a set of prompts

This shows an example function that builds a training example out of the dataset.

The **input** is the decompiled code and the **output** is the function name being learned .

* **Context should be treated as a budget like in [LmPa](#)**

```
1  def tokenize(prompt):
2      result = tokenizer(
3          prompt,
4          truncation=True,
5          max_length=4096,
6          padding=False,
7          return_tensors=None,
8      )
9
10     # "self-supervised learning" means the labels are also the inputs:
11     result["labels"] = result["input_ids"].copy()
12
13     return result
14
15     def generate_and_tokenize_prompt(data_point):
16         full_prompt = f"""You are an advanced malware reverse engineer capable \
17 of understanding decompiled C code and identifying malicious \
18 functionality please create a ### New Function Name from the \
19 decompiled code in the ### Context.
20
21 You must output a descriptive ### Function name for the decompiled code
22 provided in ### Context.
23
24 ### Context:
25 {data_point["input"]}
26
27 ### New Function Name:
28 {data_point["output"]}
29
30 """"
31     return tokenize(full_prompt)
```

Methodology

Generate a dataset

A synthetic malware dataset is generated with `ttp_code_finder.py`

- (1) recursively generate instruction prompts to request the implementation of a specific TTP using a specific API call
- (2) compile response and execute response
- (3) keep successful binaries
- (4) analyze / decompile the binaries

With this workflow we have the function description/summary/function name (instruction prompt) AND the decompiled code for the interesting function.

*NOTE: Prompt template for `ttp_code_finder.py` to include print/debug statements and lots of comments when generating source of truth.

*NOTE: Remove print statements before training or you end up relying on them instead of APIs & Arguments




```
def get_ttp_code_for_api(api, action):  
    # Step 1: send the conversation and available functions to GPT  
  
    ip_address = "172.105.16.201"  
    port = "9000"  
    domain_name = "apt.watch"  
  
    sp_content = "You are a C programming assistant designed to create working C code that "  
    sp_content += "produces specific functionality using a specific windows API. You must always "  
    sp_content += "test your code with a compiler and it must compile with x86_64-w64-mingw32-gcc. "  
    sp_content += "Comment the code and print debug messages to help the user understand how "  
    sp_content += "the code works. Try to make the code as unique and different as possible. "  
    sp_content += "When generating code that demonstrates connection to a server use the "  
    sp_content += f"IP Address: {ip_address} and Port: {port}. When contacting a domain name use "  
    sp_content += f"Domain name: {domain_name}. When sending data to a server send *.log files. "
```



```
messages=[
    {"role": "system", "content": f"{sp_content}"},
    {"role": "user", "content": f"Please provide example code for how an attacker might abuse the api \"{api}\" \
to provide \"{action}\" functionality. and test it with a compiler. You will need to pass compiler flags like \
-lpsapi -lwininet -lws2_32 in some cases"}
]

functions = [
    {
        "name": "compile_ttp_code",
        "description": "This function will test to make sure code provided compiles.",
        "parameters": {
            "type": "object",
            "properties": {
                "code": {
                    "type": "string",
                    "description": "C source code to compile",
                },
                "action": {
                    "type": "string",
                    "description": "functionality provided",
                },
                "api": {
                    "type": "string",
                    "description": "the Function name being abused",
                },
                "compiler_options": {
                    "type": "string",
                    "description": "arguments to pass the compiler",
                }
            },
        },
        "required": ["code", "action", "api", "compiler_options"],
    }
]
```



```
































































enumeration = ["CreateToolhelp32Snapshot", "EnumDeviceDrivers", "EnumProcesses", "EnumProce
injection = ["CreateFileMappingA", "CreateProcessA", "CreateRemoteThread", "CreateRemoteTh
evasion = ["CreateFileMappingA", "DeleteFileA", "GetModuleHandleA", "GetProcAddress", "Loadl
spying = ["AttachThreadInput", "CallNextHookEx", "GetAsyncKeyState", "GetClipboardData", "Ge
internet = ["WinExec", "FtpPutFileA", "HttpOpenRequestA", "HttpSendRequestA", "HttpSendReque
anti_debugging = ["CreateToolhelp32Snapshot", "GetLogicalProcessorInformation", "GetLogica
ransomware = ["CryptAcquireContextA", "EncryptFileA", "CryptEncrypt", "CryptDecrypt", "CryptC
helper = ["ConnectNamedPipe", "CopyFileA", "CreateFileA", "CreateMutexA", "CreateMutexExA",

```

```

messages=[
    {"role": "system", "content": f"{sp_content}"},
    {"role": "user", "content": f"Please provide example code for how an attacker might abuse the api \"{api}\" \
to provide \"{action}\" functionality. and test it with a compiler. You will need to pass compiler flags like \
-lpsapi -lwininet -lws2_32 in some cases"}
]

```


se_Connect_2b94c14445d6434861a07cfba...	 abuse_Connect_4a736b99562a4543476f39301...	 abuse_Connect_5fe816f4721b7c820decf22159...	 abuse_Connect_c313a0b26c8e14e5b25775560...
se_Connect_f1b2d3c712c4bcb32e3081211...	 abuse_Connect_f2926b9053c7374f4bd062fb56...	 abuse_Connect_fe408739100bd5a33f66df69f9...	 abuse_ConnectNamedPipe_02906d69cb98c3f...
se_ControlService_69217efc11564d5a4e48...	 abuse_CopyFileExA_81ae8edc599fe3fd7a62b8...	 abuse_CopyFileExA_99efe425cb40c5b4ea821b...	 abuse_CopyFileExA_f3bcdef4641152201a29d4...
se_CreateFile2_953cb2b364e8721e673821...	 abuse_CreateFileA_24c73653a547191f26248a4...	 abuse_CreateFileA_b01e51e463e30f7bee4674...	 Abuse_CreateFileA_e18d5e040076391c49d79c...
se_CreateMutexA_45dc0150489d4ea27559...	 abuse_CreateMutexExA_29ad0fd133eb2d7d6d...	 abuse_CreatePipe_5ba5a9983ae339d9b28c39...	 abuse_CreatePipe_5baf19f261a80cb635afc087...
se_CreateToolhelp32Snapshot_b7b61530f...	 abuse_DeviceIoControl_b0911d0caa283a1bf3f...	 abuse_FindClose_4969a78b7fcef0675cb53a94f...	 abuse_FindNextUrlCacheEntryA_4b35e49eb6d...
se_FindResourceA_6c0dcf4aa0c1d86d2f2e...	 abuse_FindResourceA_7ad3d16f66fa5e6d78d7...	 abuse_FindResourceA_783e79c97db579be8ed...	 abuse_FindResourceExA_796bfe4b681b6a61b...
se_FindResourceExA_cc6209a64876bd405...	 abuse_GetDesktopWindow_186fd76dea42835...	 abuse_GetDriveTypeA_abd75edc10fd2fbb31d...	 abuse_Gethostbyname_4e561917e8994fa2bd0...
se_Gethostbyname_08b1f5f3a563741edb6...	 abuse_Gethostbyname_b5b8ec7670d5a774f4a...	 abuse_Gethostname_2c6e7e89606a54358890b...	 abuse_GetLogicalDrives_1ddbb3272f5021b7e...
se_GetLogicalDrives_aacce21b9411541ef9...	 abuse_GetModuleBaseNameA_b1223443bfb8...	 Abuse_GetModuleFileNameA_3ee3b2327cf93...	 abuse_GetProcAddress_85f6ce89f070a1e0974f...
se_GetProcessHeap_de2b06d945ba9f6bec...	 abuse_GetTempFileNameA_0a637ebff756b61...	 abuse_GetTempFileNameA_218b640825d401e...	 abuse_GetTempFileNameA_d51f243a61ccb41...
se_GetTempPathA_5ec7f371d7766fc271ef...	 abuse_GetTempPathA_28ccd015af36b94b6796...	 abuse_GetTempPathA_200fe38580990c891474...	 abuse_GetThreadId_62261d1723041d683e372...
se_GetTickCount_dbcfad40b372cc548bd9...	 abuse_GetTickCount_e7ccf8d324ce0e9f3563e0...	 abuse_HttpAddRequestHeaders_68e55dd25fa...	 abuse_HttpOpenRequestA_6d4a0c3a7836e2d...
se_HttpSendRequestExA_1a92a5cca95982...	 abuse_HttpSendRequestExA_6d2d115f6744e1...	 abuse_HttpSendRequestExA_6d199a44f7016c...	 abuse_HttpSendRequestExA_cfb0a6d0a59e99...
se_Inet_addr_3afa27f9480dc13d773b80ffe...	 abuse_Inet_addr_9a47bb22dfe1ccae8ba90577...	 abuse_Inet_addr_12e01483b427356d222a4fd9...	 abuse_Inet_addr_d3a3c60d846ee55ed66accf6...
se_InternetOpenA_6facdb9a898e2c2de858...	 abuse_InternetOpenA_54fe06fe8adc3cf960...	 Abuse_InternetOpenA_824d51a3cde035699cb...	 abuse_InternetOpenA_eb53aee87bac0ce4259...
se_InternetOpenUrlA_7a3874a3fa1616034f...	 abuse_InternetOpenUrlA_8063131149e9c7647...	 abuse_InternetOpenUrlA_cb22410c7482c4aa6...	 abuse_InternetReadFileExA_0708ea4eda66d6e...
se_InternetReadFileExA_698449738c542c7...	 abuse_InternetReadFileExA_e776a10490f1335...	 abuse_InternetSetOptionA_8a8cdf8aa3bac59e...	 abuse_InternetSetOptionA_42e0ee06c2e6bec0...
se_IsDebuggerPresent_1c17d09acb028176...	 abuse_Listen_2b0fefe8a5a456d70a773be12d1...	 abuse_Listen_65ac8b64edfbb0780f3e90b1ce5...	 abuse_Listen_a3830a2a8d8dc84211478fb9acc...
se_LoadLibraryExA_3c02259a9fb5b38fa3b...	 abuse_LockResource_5f09752376ae5c140396a...	 abuse_LockResource_7d065117ebbc42250861...	 abuse_LockResource_141e237c4f305d82f09e4...
se_lstrcatA_96c4e57d0f1dd53585c06c51c0...	 abuse_lstrcatA_abc05c345b85536f3bcfd53096...	 abuse_lstrcatA_d9c2f0fb4ef1bd3c5989a791c8...	 abuse_MoveFileA_44ae480f6fe2610c470816af...

Methodology

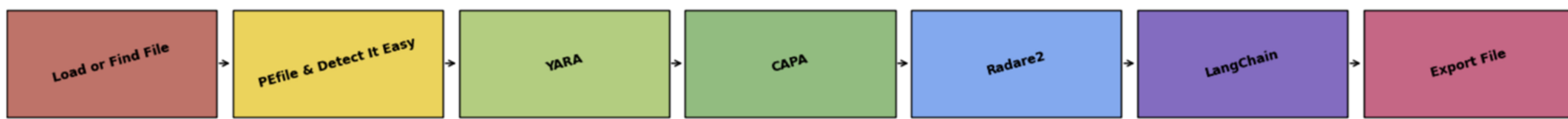
Generate a dataset: Malware Processing Pipeline

Synthetic data is a great start, but need to get more real world malware samples in front of the model for training. We need to be able to recognize anti-dissassembly / anti-debugging malware code.

We leverage an existing set of tools to generate candidate functions for analysis and store them for review.



AiAnalyzer.py Workflow



1. PEfile and Detect It Easy

The Python library `pefile` and the CLI tool `Detect It Easy` (DIE) are potent utilities for analyzing PE (Portable Executable) files. `pefile` facilitates the inspection of a PE file's structure and attributes, including headers, sections, imports, exports, and resources, aiding in the understanding of its behavior. Additionally, `Detect It Easy` offers extended functionality, including the identification of file types, entropy analysis, and detection of specific packers.

2. YARA

YARA is known for its pattern matching capabilities, making it a valuable tool for identifying and classifying malware based on textual or binary patterns. It can be easily extended

3. Capa

Capa is utilized for capability hunting which is crucial for understanding what a particular piece of code or software is capable of doing.

4. Radare2

This is a comprehensive framework for analyzing binaries, dealing with the analysis of imports, exports, strings and decompilation, which are critical for reverse engineering purposes.

5. Langchain

Interaction with LLM, generate requests and store responses

6. Ghidra llm.py

The output from AiAnalyzer is then loaded by a plugin in Ghidra, which is a software reverse engineering (SRE) framework. This plugin is designed to process and display the intelligence extracted by the LLM.





Execute CAPA

Finding interesting functions

We execute CAPA to detect interesting regions of code and end up with a bunch of memory addresses (offsets).

[Github](#)

```
1 rule:
2   meta:
3     name: log keystrokes
4     namespace: collection/keylog
5     authors:
6       - moritz.raabe@mandiant.com
7     scope: function
8     att&ck:
9       - Collection::Input Capture::Keylogging [T1056.001]
10    examples:
11      - C91887D861D9BD4A5872249B641BC9F9:0x4015FD
12    features:
13      - or:
14        - and:
15          - api: SetWindowsHookEx
16          - api: GetKeyState
17        - and:
18          - api: RegisterHotKey
19          - api: user32.keyboard_event
20          - api: UnregisterHotKey
21        - and:
22          - api: CallNextHookEx
23          - api: user32.GetKeyNameText
24          - api: user32.GetAsyncKeyState
25          - api: user32.GetForegroundWindow
26        - api: user32.AttachThreadInput
27        - api: user32.MapVirtualKey
```




Execute CAPA

Finding interesting functions

We execute CAPA to detect interesting regions of code and end up with a bunch of memory addresses (offsets).

[Github](#)

```
1 rule:
2   meta:
3     name: capture public ip
4     namespace: collection/network
5     authors:
6       - "@_re_fox"
7     scope: function
8     att&ck:
9       - Discovery::System Network Configuration Discovery [T1016]
10    examples:
11      - 84f1b049fa8962b215a77f51af6714b3:0x100061e5
12    features:
13      - and:
14        - api: InternetOpen
15        - api: InternetOpenUrl
16        - api: InternetReadFile
17      - or:
18        - substring: "bot.whatismyipaddress.com"
19        - substring: "ipinfo.io/ip"
20        - substring: "checkip.dyndns.org"
21        - substring: "ifconfig.me"
22        - substring: "ipecho.net/plain"
23        - substring: "api.ipify.org"
24        - substring: "checkip.amazonaws.com"
25        - substring: "icanhazip.com"
26        - substring: "wtfismyip.com/text"
27        - substring: "api.myip.com"
28        - substring: "ip-api.com/line"
```



Execute CAPA

Finding interesting functions

We execute CAPA to detect interesting regions of code and end up with a list of memory addresses (offsets).

[Github](#)

```
import capa

def capa_match(self, file_path):
    format_ = "auto"
    backend = "vivisect"
    _os = "windows"
    #get the files in self.sig path and make a list of them, include full path
    flie_list = []
    for root, dirs, files in os.walk(self.sigpath):
        for file in files:
            if file.endswith(".sig"):
                flie_list.append(os.path.join(root, file))
    sigpaths = flie_list #this makes a list with a single string in it, I was having trouble with the sigpath being a str
    rules = capa.main.get_rules([f"{self.caparulepath}"])

    try:
        extractor = capa.main.get_extractor(file_path, format_, _os, backend, sigpaths)
        capabilities, _ = capa.main.find_capabilities(rules, extractor)
        output_data = {}
        for rule_name, matches in capabilities.items():
            for match in matches:
                memory_address = match[0]
                if str(memory_address) not in output_data:
                    output_data[str(memory_address)] = []
                output_data[str(memory_address)].append(rule_name)
        self.analysis_results["capa"] = output_data # Store analysis results in capa key
    #TODO: Implement better error handling
    except capa.exceptions.UnsupportedFormatError:
        #TODO This causes a parsing bug down the line when a dict is expected but a str is found.
        self.analysis_results["capa"] = f"Unsupported format: {file_path}"
        print("Unsupported format: ", file_path)
    except ValueError:
        #TODO This causes a parsing bug down the line when a dict is expected but a str is found.
        self.analysis_results["capa"] = f"ValueError: {file_path}"
        print("ValueError: ", file_path)
```



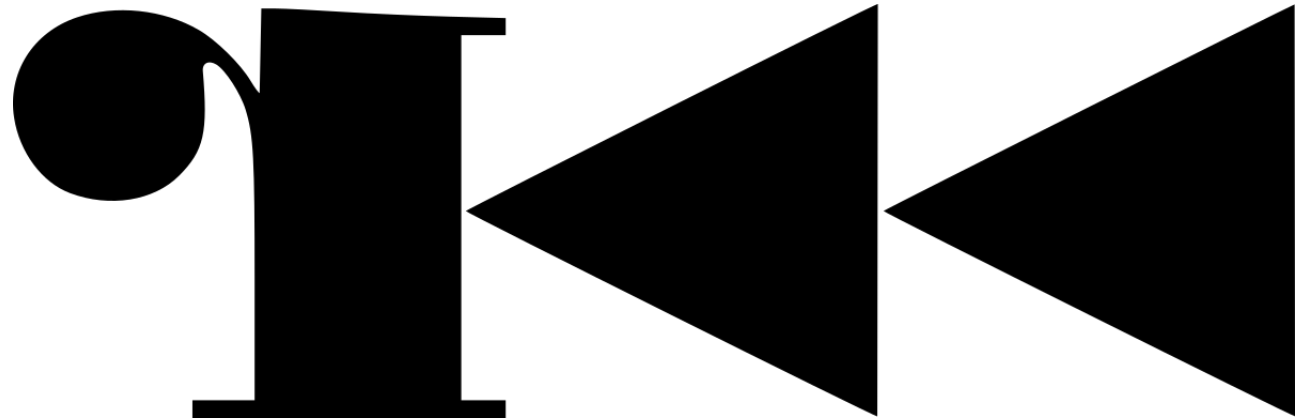
Decompile Function

Decompiling interesting functions

With the interesting function offsets above we use radare2 to decompile the function and store the results in the analysis_results dict.

[Website](#)

[Github](#)



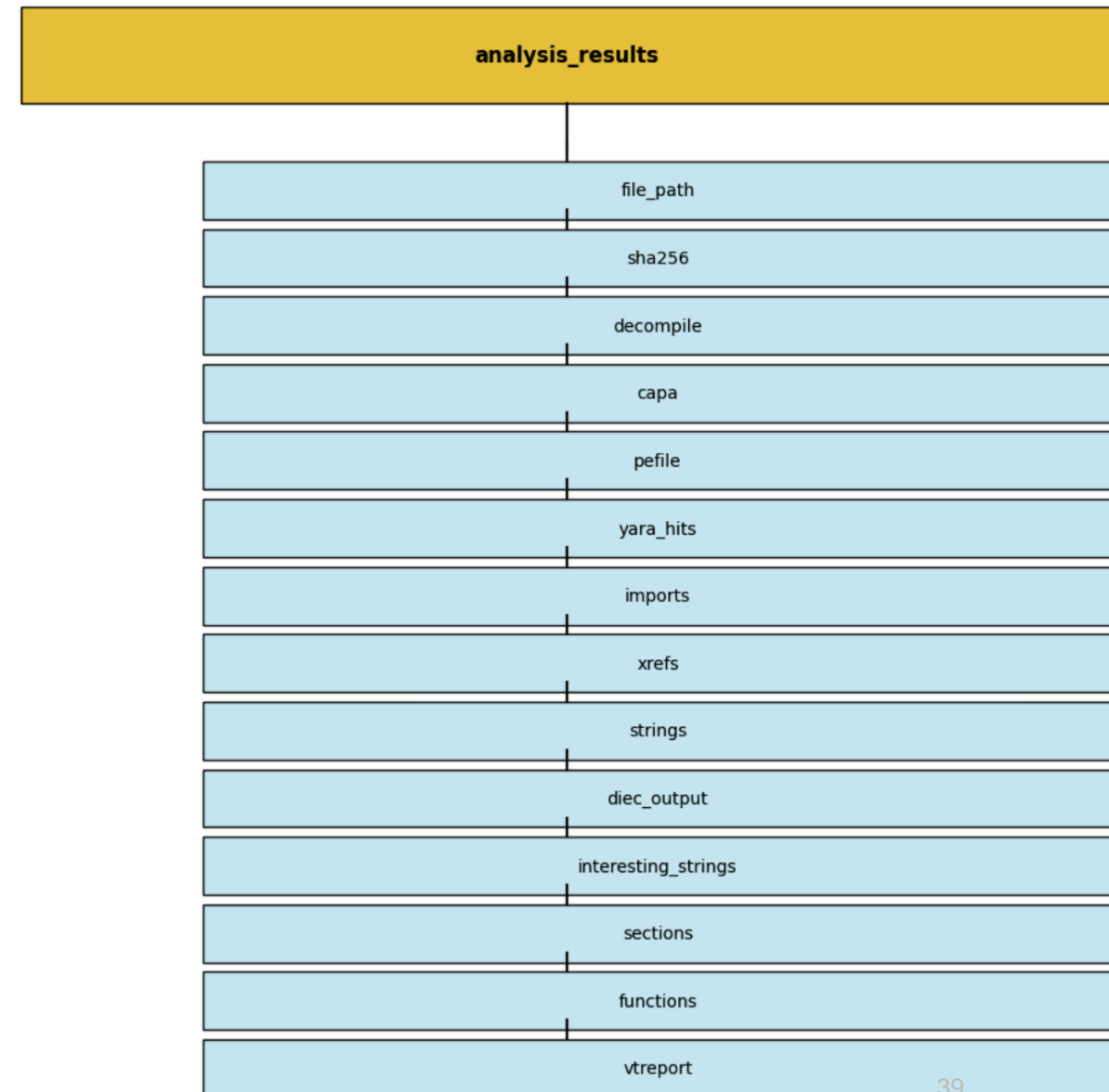


Analysis Results Data

Decompiling interesting functions

`analysis_results` is a dictionary of dictionaries containing the results of the entire analysis pipeline.

We're interested in the capa dictionary which store the function offsets for matched rules and the functions dictionary that contains the decompiled code.



7: gpu@gpu: ~/code/AiAnalyzer

```
llm name: execute_function_chain_004066e0
llm name: allocate_memory_004067e0
llm name: get_module_handle_and_set_pointers_00406710
llm name: get_module_handle_and_set_pointers_00406710
llm name: execute_process_heap_functions_004066d0
llm name: execute_function_chain_004066e0
llm name: convert_float_to_integer_00407610
llm name: execute_function_chain_004066e0
llm name: convert_float_to_integer_00407610
llm name: get_module_handle_and_set_pointers_00406710
llm name: free_memory_00407990
llm name: get_module_handle_and_set_pointers_00406710
llm name: free_memory_00407990
llm name: convert_float_to_integer_00407610
llm name: handle_bad_instruction_data_00425069
llm name: free_memory_00407990
llm name: convert_float_to_integer_00407610
llm name: handle_bad_instruction_data_00425069
llm name: get_process_type_00407c20
llm name: free_memory_00407990
llm name: handle_bad_instruction_data_00425069
llm name: get_process_type_00407c20
llm name: handle_bad_instruction_data_00425069
llm name: get_process_type_00407c20
llm name: get_process_type_00407c20
```

8: gpu@gpu: ~/code/AiAnalyzer

```
llm name: process_data_0ede0b05
llm name: process_data_0ede0bd4
llm name: execute_process_with_params_0ede0ca0
llm name: execute_process_0ede0d4e
llm name: process_and_execute_0ede0dea
llm name: process_critical_section_0ede0ea2
llm name: process_critical_section_0ede1011
llm name: process_function_0ede11a9
llm name: process_data_0ede1320
llm name: check_and_call_function_0ede1c71
llm name: process_function_0ede3680
llm name: handle_window_message_0ede4367
llm name: calculate_offset_0ede4fbc
llm name: free_memory_0ede52c4
llm name: free_and_reset_param_1_0ede55ac
llm name: process_cleanup_0ede5d26
llm name: process_critical_section_0ede612b
llm name: process_critical_section_0ede6585
llm name: copy_variants_0ede6d44
llm name: cleanup_process_data_0ede6bf2
llm name: validate_and_process_data_0ede7101
llm name: delete_and_cleanup_process_0ede6492
llm name: create_new_process_with_param_0ede75ab
llm name: initialize_process_0ede7f4e
llm name: process_data_0ede8eb5
```

4: gpu@gpu: ~/code/AiAnalyzer

```
llm name: execute_process_56dc4f47
llm name: initialize_process_56d37f50
llm name: process_param_56d380fc
llm name: process_function_56d381a8
llm name: get_cpuid_info_56d3840c
llm name: initialize_param_1_56d38448
llm name: execute_process_56d3852c
llm name: initialize_memory_56d38628
llm name: process_param_2_56d3e948
llm name: initialize_process_pointer_56d38728
llm name: check_and_execute_callback_56d3890c
llm name: find_and_call_function_56d38b38
llm name: call_function_with_param_56d38c20
llm name: execute_function_56d38c78
llm name: process_execution_56d38cd8
llm name: check_and_execute_task_56d38d50
llm name: get_logical_processor_information_56d38d94
llm name: reset_and_call_another_function_56d38f08
llm name: execute_process_56d38f78
llm name: check_and_acquire_lock_56d38ff8
llm name: update_process_status_56d3927c
llm name: update_param_2_pointer_56d39368
llm name: remove_element_from_list_56d393d8
llm name: execute_process_56d394f8
llm name: update_process_status_56d39b14
```

5: gpu@gpu: ~/code/AiAnalyzer

```
llm name: leave_critical_section_100d5ed1
llm name: initialize_process_100e2de7
llm name: call_LCMapStringEx_100e2e01
llm name: convert_locale_name_to_lcid_100e2e1b
llm name: enter_critical_section_100e4a8f
llm name: leave_critical_section_100e4ad7
llm name: calculate_polynomial_approximation_100e6ade
llm name: convert_to_float10_100e70ab
llm name: process_status_update_100e961e
llm name: process_data_100e965a
llm name: call_fcn_with_reversed_params_100eaa7e
llm name: call_fcn_100d4a66_100d2e0b
llm name: wait_for_single_object_and_decrement_counter_1003bbd0
llm name: wait_and_close_handle_10049e00
llm name: initialize_process_10001df3
llm name: calculate_weights_10004ea0
llm name: validate_string_10017090
llm name: compare_memory_regions_100170a0
llm name: get_unique_value_100200b0
llm name: find_index_of_value_in_array_100307f0
llm name: find_first_process_with_flag_10036690
llm name: find_process_by_param_10036cf0
llm name: get_value_from_array_1005af00
llm name: execute_process_with_callback_10055e10
llm name: terminate_and_exit_process_1001f410
```

ATIONS



Review Dataset Item

AllFilter

Unreviewed Items Remaining: 1

Suggested name:

LLM Suggested name:

all_retrieveOperatingSystemAndComputerInfo

LLM Summary:

Retrieves the operating system name, computer name, and a constant string

Modify Short Summary:

Retrieves the operating system name, computer name, and a constant string

Modified by step Summary:

1. Opens the registry key 'SOFTWARE\Microsoft\Windows NT\CurrentVersion' using RegOpenKeyExW function.
2. Queries the value of 'ProductName' from the opened registry key using RegQueryValueExW function.
3. Closes the opened registry key using RegCloseKey function.
4. If the return value of RegCloseKey is not 0, sets the 'local_210' variable to '???' using lstrcpyW function.
5. Retrieves the fully qualified domain name of the computer using GetComputerNameExW function.
6. Formats the retrieved operating system name and computer name into a JSON string and appends it to 'DAT_0043c8e4' using lstrcatW function.
7. Appends a constant string 'venus' to 'DAT_0043c8e4' using lstrcatW function.

1. Opens the registry key 'SOFTWARE\Microsoft\Windows NT\CurrentVersion' using RegOpenKeyExW function.
2. Queries the value of 'ProductName' from the opened registry key using RegQueryValueExW function.
3. Closes the opened registry key using RegCloseKey function.
4. If the return value of RegCloseKey is not 0, sets the 'local_210' variable to '???' using lstrcpyW function.
5. Retrieves the fully qualified domain name of the computer using GetComputerNameExW function.
6. Formats the retrieved operating system name and computer name into a JSON string and appends it to 'DAT_0043c8e4' using lstrcatW function.
7. Appends a constant string 'venus' to 'DAT_0043c8e4' using lstrcatW function.

Save Changes

Code:

```
/*
This code retrieves the operating system name, computer name, and a constant string.
The code performs the following steps:

1. Opens the registry key 'SOFTWARE\Microsoft\Windows NT\CurrentVersion' using RegOpenKeyExW
function.
2. Queries the value of 'ProductName' from the opened registry key using RegQueryValueExW
function.
3. Closes the opened registry key using RegCloseKey function.
4. If the return value of RegCloseKey is not 0, sets the 'local_210' variable to '???' using
lstrcpyW function.
5. Retrieves the fully qualified domain name of the computer using GetComputerNameExW function.
6. Formats the retrieved operating system name and computer name into a JSON string and appends
it to 'DAT_0043c8e4' using lstrcatW function.
7. Appends a constant string 'venus' to 'DAT_0043c8e4' using lstrcatW function. */

void retrieveOperatingSystemAndComputerInfo(void)
{
    LSTATUS LVar1;
    WCHAR local_210 [256];
    DWORD local_10;
    DWORD local_c;
    HKEY local_8;

    local_8 = (HKEY)0x0;
    local_c = 0x100;
    RegOpenKeyEx((HKEY)0x80000002,L"SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion",0,1,&local_8);
    RegQueryValueExW(local_8,L"ProductName",(LPDWORD)0x0,(LPDWORD)0x0,(LPBYTE)local_210,&local_c);
    LVar1 = RegCloseKey(local_8);
    if (LVar1 != 0) {
        lstrcpyW(local_210,L"???");
    }
    local_10 = 0x100;
    wsprintfW(DAT_0043ab08,L"\\OS\\: \\%s\\",local_210);
    lstrcatW(DAT_0043c8e4,DAT_0043ab08);
    GetComputerNameExW(ComputerNamePhysicalDnsFullyQualified,local_210,&local_10);
    wsprintfW(DAT_0043ab08,L"\\CompName\\: \\%s\\",local_210);
    lstrcatW(DAT_0043c8e4,DAT_0043ab08);
    wsprintfW(DAT_0043ab08,L"\\ext\\: \\%s\\",L"venus");
    lstrcatW(DAT_0043c8e4,DAT_0043ab08);
    return;
}
```



```

int __fastcall FUN_00402e90(int param_1,uint param_2,uint param_3,uint param_4)
{
    int iVar1;
    int iVar2;
    uint uVar3;
    int iVar4;
    uint uVar5;
    uint local_8;

    uVar5 = 0;
    local_8 = 0xffffffff;
    iVar4 = *(int *)*(int *)(param_1 + 0x3c) + 0x78 + param_1;
    iVar1 = *(int *)(iVar4 + 0x24 + param_1);
    iVar4 = iVar4 + param_1;
    iVar2 = *(int *)(iVar4 + 0x20);
    if (*(int *)(iVar4 + 0x18) != 1) {
        do {
            uVar3 = FUN_00417be0((int *)*(int *)(iVar2 + param_1 + uVar5 * 4) + param_1,param_3,param_4)
            ;
            if (param_2 == uVar3) {
                local_8 = (uint)*(ushort *)(iVar1 + param_1 + uVar5 * 2);
                break;
            }
            uVar5 = uVar5 + 1;
        } while (uVar5 < *(int *)(iVar4 + 0x18) - 1U);
    }
    return *(int *)*(int *)(iVar4 + 0x1c) + local_8 * 4 + param_1 + param_1;
}

```

```

FUN_00402e90 -> findMatchingStringIndex
param_1 -> pImportDesc
param_2 -> searchedHash
param_3 -> checksum
param_4 -> funcRva
iVar1 -> nameTableRva
iVar2 -> addrTableRva
uVar3 -> currChecksum
iVar4 -> pThunk
uVar5 -> i
local_8 -> matchingIndex

```

Update Rating



LLMs in the Lab

LLM empowered Malware Analysis.

Create a training dataset that will work with hugging face transformers

```
import os
import json
import click
from datasets import Dataset
import pandas as pd
from multiprocessing import Pool, cpu_count

def process_file(json_file):
    data = []
    with open(json_file, 'r') as f:
        try:
            data_dict = json.load(f)
            for function in data_dict['decompile']:
                try:
                    code = data_dict['decompile'][function]['decompiled_code']

                    # Remove lines that start with '//'
                    code = "\n".join(line for line in code.split('\n') if not line.strip().startswith('//'))

                    llm_name = data_dict['decompile'][function]['llm_name']
                    llm_name = llm_name.rpartition('_')[0]
                    text = "<s>" # Begin of Sentence token
                    text += "You are a malware analyst. You have been given the following decompiled code:\n"
                    text += code
                    text += "\n\nYour task is to determine what the code does and provide a descriptive name for the function.\n"
                    text += "Please enter your answer below:\n"
                    text += "Answer:" + llm_name + "</s>" # End of Sentence token
                    data.append({"text": text})
                except KeyError as e:
                    print(f'Error processing function in JSON file: {json_file} Error: {e}')
            except Exception as e:
                print(f'Error loading JSON file: {json_file} Error: {e}')
    return data

@click.command()
@click.option('--dir', required=True, help='The directory to search for JSON files.')
def create_dataset(dir):
    """
    Recursively walk through directory and find JSON files.
    Create a HuggingFace Dataset from the data and split into train and validation sets.
    """
    json_files = [os.path.join(root, file) for root, _, files in os.walk(dir) for file in files if file.endswith('.json')]

    # Create a pool of workers and distribute the file processing among them
    with Pool(cpu_count()) as p:
        data_list = p.map(process_file, json_files)

    # Flatten the list of lists into a single list
    data = [item for sublist in data_list for item in sublist]


    dataset = Dataset.from_pandas(pd.DataFrame(data))

    # split into train and validation sets
    datasets = dataset.train_test_split(test_size=0.2)

    # save datasets to disk
    datasets['train'].save_to_disk('train_dataset-large-v2')
    datasets['test'].save_to_disk('validation_dataset-large-v2')

    print('Saved training set to disk.')
    print('Saved validation set to disk.')

if __name__ == "__main__":
    create_dataset()
```

 Dataset card  Files and versions  Community

Dataset Viewer

 Auto-converted to Parquet  API  Go to dataset viewer

Split

train (60.5k rows)

 Search this dataset

instruction
string · *classes*

1 value

input
string · *lengths*

42 9.67k

output
string · *lengths*

3 561

Analyze the supplied decompiled code and produce a descriptive name for the function. The function...

```
void fcn.140030b80(ulong param_1, ulong param_2,
ulong param_3) { ulong uVar1; uVar1 =...
```

process_with_params

Analyze the supplied decompiled code and produce a descriptive name for the function. The function...

```
void __thiscall fcn.005ac32c(uint *param_1, uint
param_2, uint *param_3, uint param_4, uint...
```

call_function_with_parameter

Analyze the supplied decompiled code and produce a descriptive name for the function. The function...

```
uchar __fastcall fcn.0049b228(uint param_1,
int32_t param_2, int32_t param_3, int32_t param_4...
```

find_first_negative_index

Analyze the supplied decompiled code and produce a descriptive name for the function. The function...

```
int32_t __thiscall fcn.0048264a(int32_t param_1,
uint param_2) { int32_t iVar1; fcn.0047ae5b();...
```

process_status_update

Analyze the supplied decompiled code and produce a descriptive name for the function. The function...

```
void fcn.1000c7ad(void) { int32_t in_EAX;
fcn.1000cb20(); *(in_EAX + 0x54) = 0x100518a4;...
```

set_process_address

Analyze the supplied decompiled code and produce a

```
uint __fastcall fcn.004cd798(uint param_1, char
```

...

< Previous 1 2 3 ... 605 Next >



LLMs in the Lab


LLM empowered Malware Analysis.

fine tuned StarCode->StarDecoder
(It sucked)


```
train_data.start_iteration = 0


print("Starting main loop")

training_args = TrainingArguments(
    output_dir=args.output_dir,
    dataloader_drop_last=True,
    evaluation_strategy="steps",
    max_steps=args.max_steps,
    eval_steps=args.eval_freq,
    save_steps=args.save_freq,
    logging_steps=args.log_freq,
    per_device_train_batch_size=args.batch_size,
    per_device_eval_batch_size=args.batch_size,
    learning_rate=args.learning_rate,
    lr_scheduler_type=args.lr_scheduler_type,
    warmup_steps=args.num_warmup_steps,
    gradient_accumulation_steps=args.gradient_accumulation_steps,
    gradient_checkpointing=not args.no_gradient_checkpointing,
    fp16=not args.no_fp16,
    bf16=args.bf16,
    weight_decay=args.weight_decay,
    run_name="2600-StarCoder-finetuned",
    report_to="wandb",
    ddp_find_unused_parameters=False,
    optim="paged_adamw_8bit"
)
```


binaryninja Update README.md
0a62ac1 on Jun 2
🕒 76 comm

📁 chat	Add example to fine-tune StarCoder for chat-b...	4 months a
📁 finetune	enabled qlora / 4 bit training	4 months a
📄 .gitignore	Add example to fine-tune StarCoder for chat-b...	4 months a
📄 LICENSE	Create LICENSE	5 months a
📄 README.md	Update README.md	4 months a
📄 requirements.txt	Update requirements.txt	5 months a


README.md



2600-StarCoder QLoRA 4bit

I followed this guide [Making LLMs even more accessible with bitsandbytes, 4-bit quantization and QLoRA](#) by huggingface and implement the code in this repo to load the model in 4bit int and train using the methods outlined in the paper.

To make this work you're going to need the latest accelerate, transformers, and bitsandbytes libs.


```
pip install -q -U bitsandbytes
pip install -q -U git+https://github.com/huggingface/transformers.git
pip install -q -U git+https://github.com/huggingface/peft.git
pip install -q -U git+https://github.com/huggingface/accelerate.git
```

The model can be loaded in 11GB of VRAM at 4bit.



Introducing Code Llama, a state-of-the-art large language model for coding

August 24, 2023



Code Llama

PROMPT

In Bash, how do I list all text files in the current directory (excluding subdirectories) that have been modified in the last month?

Clear

Submit

RESPONSE

You can use the 'find' command in Bash to list all text files in the current directory (excluding subdirectories) that have been modified in the last month. The command would be:

```
find . -maxdepth 1 -type f -mtime -30 -name "*.txt"
```

Here is a breakdown of the command:

- '.' specifies the current directory
- '-maxdepth 1' tells 'find' to only search the current directory and not any subdirectories
- '-type f' tells 'find' to only search for files (not directories or other types of files)
- '-mtime -30' tells 'find' to only search for files that have been modified in the last 30 days
- '-name "*.txt"' tells 'find' to only search for files with the '.txt' extension

Note that the 'm-time' option uses the number of days since the file was last modified, so '-mtime -30' means "modified in the last 30 days." If you want to search for files modified in the last month, you can use the '-mtime -30' option.



Takeaways

- Code Llama is a state-of-the-art LLM capable of generating code, and natural language about code, from both code and natural language prompts.
- Code Llama is free for research and commercial use.
- Code Llama is built on top of Llama 2 and is available in three models:
 - Code Llama, the foundational code model;
 - Code Llama - Python specialized for Python;
 - and Code Llama - Instruct, which is fine-tuned for understanding natural language instructions.
- In our own benchmark testing, Code Llama outperformed state-of-the-art publicly available LLMs on code tasks

<https://ai.meta.com/blog/code-llama-large-language-model-coding/>



Finetune codellama for SQL Task

A complete guide to fine-tuning Code Llama

In this guide I show you how to fine-tune Code Llama to become a beast of an SQL developer. For coding tasks, you can generally get much better performance out of Code Llama than Llama 2, especially when you specialise the model on a particular task:

- I use the [b-mc2/sql-create-context](#) which is a bunch of text queries and their corresponding SQL queries
- A Lora approach, quantizing the base model to int 8, freezing its weights and only training an adapter
- Much of the code is borrowed from [alpaca-lora](#), but I refactored it quite a bit for this

I used an A100 GPU machine with Python 3.10 and cuda 11.8 to run this notebook. It took about an hour to run. (I also tested that this code works on Colab Pro.)

*This the corresponding [notebook](#).

```
eval_prompt = """You are a powerful text-to-SQL model. Your job is to a

You must output the SQL query that answers the question.
### Input:
Which Class has a Frequency MHz larger than 91.5, and a City of license

### Context:
CREATE TABLE table_name_12 (class VARCHAR, frequency_mhz VARCHAR, city_

### Response:
"""

model_input = tokenizer(eval_prompt, return_tensors="pt").to("cuda")

model.eval()
with torch.no_grad():
    print(tokenizer.decode(model.generate(**model_input, max_new_tokens
```

And the model outputs:

```
SELECT class FROM table_name_12 WHERE frequency_mhz > 91.5 AND city_of_
```

<https://ragntune.com/blog/guide-fine-tuning-code-llama>



Large Language models

Fine-tuning Notes

Data parallel vs model parallel

Context size limits based on memory

Batch size / gradient / accumulation steps.

Local training (Speed and context suffers)

Coreweave training (\$\$\$)



Large Language models

Fine-tuning: Review

Define the set of tasks

Collect the data (answers)

- Leverage existing tools to automate the generation of your dataset
- Use larger language models (eg: GPT4) to generate training data
- Get creative (ttp_code_finder.py)

Construct the prompt

- Read the paper! Implement system / init prompts to match training, add special tokens for end of sequence, instruction, and input

Tokenization

- Ensure prompt + code is under the max len. Don't poison your training with truncated samples.



Large Language models

Fine-tuning: PEFT/QLoRA

PEFT (Parameter-Efficient Fine-Tuning) and QLoRA are mechanisms aimed at training large language models (LLMs) on modest hardware.

1. PEFT:

PEFT is a subset of fine-tuning that emphasizes parameter efficiency. Unlike traditional fine-tuning where all coefficients of the model are altered, PEFT selectively tunes a subset of them, which significantly reduces the computational and memory requirements, making it a desirable approach for training large models like Falcon 7B where efficiency is crucial [1].

PEFT methods optimize the adaptation of LLMs to specific tasks, enhancing their performance and speed without demanding extensive GPU power and memory resources [2].

Some notable methods under PEFT include LoRA and QLoRA, which are specifically designed for fine-tuning LLMs in a parameter-efficient manner [3].

[1] <https://www.analyticsvidhya.com/blog/2023/10/llm-fine-tuning-with-peft-techniques/#:~:text=,7B%2C%20where%20efficiency%20is%20crucial>

[2] <https://www.analyticsvidhya.com/blog/2023/08/lora-and-qlora/#:~:text=Overview%20As%20we%20delve%20deeper,LLMs%29%20to%20specific%20tasks>

[3] <https://abvijaykumar.medium.com/fine-tuning-llm-parameter-efficient-fine-tuning-peft-lora-qlora-part-1-571a472612c4#:~:text=,We%20will%20understand%20how>



Large Language models

Fine-tuning: PEFT/QLoRA

QLoRA [1] is a method derived by combining Quantization, a weight reduction technique, with LoRA, a parameter-efficient fine-tuning technique. This combination facilitates the fine-tuning of large models with very resource-efficient utilization⁴.

Specifically, QLoRA enables the fine-tuning of 7-billion-parameter models such as Llama-2 7B or Bloom 7B on a 16GB GPU. The process involves adding quantization to LoRA, which allows for the training of significantly larger and more powerful models while maintaining resource efficiency [2].

An example highlighted is the fine-tuning of a 7-billion-parameter Bloom model on a T4 16GB GPU in Google Colab, which was made possible through the application of QLoRA⁴.

[1] <https://www.arxiv-vanity.com/papers/2305.14314/>

[2] <https://towardsai.net/p/machine-learning/qlora-training-a-large-language-model-on-a-16gb-gpu>



AiAnalyzer.py >> output x 5000

```
35 dataset2 = load_dataset("dyngnosis/function_names_v2", split="train")
36 (variable) split_data: Any | DatasetDict
37 # Split the dataset into training and evaluation sets
38 split_data = dataset2.train_test_split(test_size=0.01)
39 train_dataset = split_data["train"]
40 eval_dataset = split_data["test"]
41
42 base_model = "codellama/CodeLlama-34b-hf"
43 model = AutoModelForCausalLM.from_pretrained(
44     base_model,
45     load_in_8bit=True,
46     torch_dtype=torch.float16,
47     device_map="auto",
48 )
49 tokenizer = AutoTokenizer.from_pretrained("codellama/CodeLlama-34b-hf")
50
```



Setup prompt template. Tokenize and prep model for 8bit training

```
84
85 def generate_and_tokenize_prompt(data_point):
86     full_prompt =f"""You are an advanced malware reverse engineer capable of understanding decompiled C code and identifying malicious functionality
87
88     You must output a descriptive ### Function name for the decompiled code provided in ### Context.
89
90     ### Context:
91     {data_point["input"]}
92
93     ### New Function Name:
94     {data_point["output"]}
95
96     """
97     return tokenize(full_prompt)
98
99     tokenized_train_dataset = train_dataset.map(generate_and_tokenize_prompt)
100     tokenized_val_dataset = eval_dataset.map(generate_and_tokenize_prompt)
101
102     model.train() # put model back into training mode
103     model = prepare_model_for_int8_training(model)
```



LORA Config

```
105 config = LoraConfig(  
106     r=16,  
107     lora_alpha=16,  
108     target_modules=[  
109         "q_proj",  
110         "k_proj",  
111         "v_proj",  
112         "o_proj",  
113     ],  
114     lora_dropout=0.05,  
115     bias="none",  
116     task_type="CAUSAL_LM",  
117 )  
118 model = get_peft_model(model, config)
```



Setup training arguments

```
135 if torch.cuda.device_count() > 1:
136     # keeps Trainer from trying its own DataParallelism when more than 1 gpu is available
137     model.is_parallelizable = True
138     model.model_parallel = True
139
140 batch_size = 4
141 per_device_train_batch_size = 1
142 gradient_accumulation_steps = batch_size // per_device_train_batch_size
143 output_dir = "decode-llama-4096-34b"
144
145 training_args = TrainingArguments(
146     per_device_train_batch_size=per_device_train_batch_size,
147     gradient_accumulation_steps=gradient_accumulation_steps,
148     #this was defaulting to 8 and causing OOM errors
149     per_device_eval_batch_size=per_device_train_batch_size,
150     warmup_steps=50,
151     max_steps=10000,
152     learning_rate=3e-4,
153     fp16=True,
154     logging_steps=10,
155     optim="adamw_torch",
156     evaluation_strategy="steps", # if val_set_size > 0 else "no",
157     save_strategy="steps",
158     #eval_steps=100,
159     #set to 10 for debugging
160     eval_steps=500,
161     save_steps=500,
162     output_dir=output_dir,
163     # save_total_limit=3,
164     load_best_model_at_end=False,
165     # ddp_find_unused_parameters=False if ddp else None,
166     group_by_length=True, # group sequences of roughly the same length together to speed up training
167     report_to="wandb", # if use_wandb else "none",
168     run_name=f"decodellama-{datetime.now().strftime('%Y-%m-%d-%H-%M')}", # if use_wandb else None,
169 )
170
```



GPUs go Brrrrr....

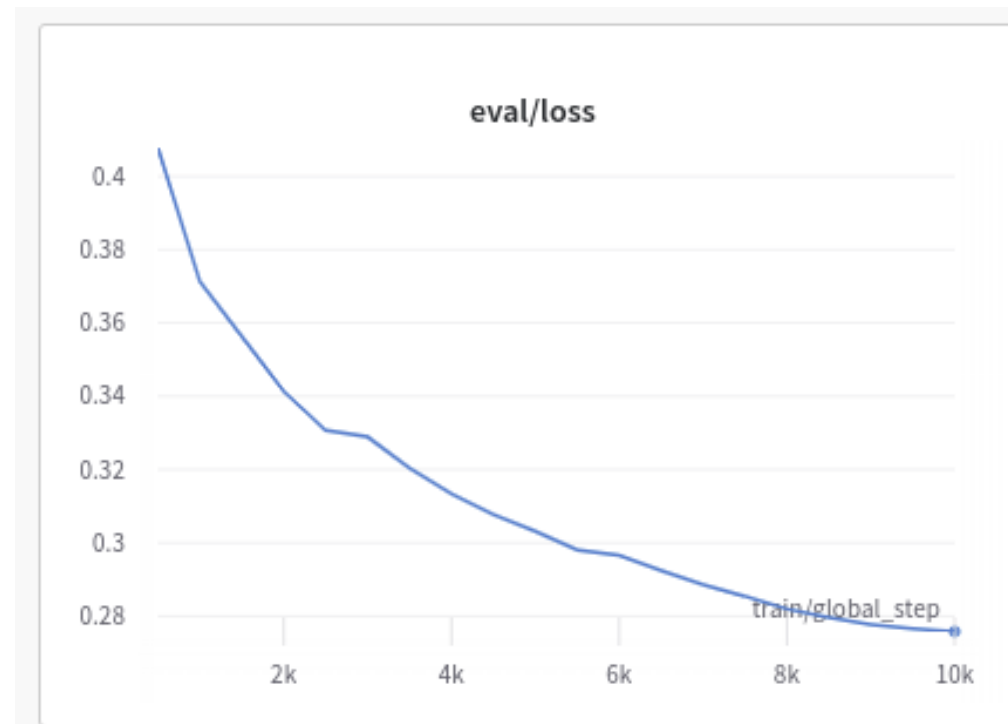
Every 2.0s: nvidia-smi

Fri Jun 2 16:57:47 2023

NVIDIA-SMI 515.65.01 Driver Version: 515.65.01 CUDA Version: 11.7									
GPU Name		Persistence-M		Bus-Id		Disp.A		Volatile Uncorr. ECC	
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage		GPU-Util		Compute M.	
								MIG M.	
=====									
0	NVIDIA GeForce ...	Off		00000000:0B:00.0	On			N/A	
100%	86C	P0	202W / 350W	23679MiB / 24576MiB		100%		Default	
								N/A	

1	NVIDIA GeForce ...	Off		00000000:0C:00.0	Off			N/A	
71%	58C	P2	165W / 390W	22741MiB / 24576MiB		100%		Default	
								N/A	

Processes:							
GPU	GI ID	CI ID	PID	Type	Process name	GPU Memory Usage	
0	N/A	N/A	4335	G	/usr/lib/xorg/Xorg	597MiB	
0	N/A	N/A	4476	G	/usr/bin/gnome-shell	94MiB	
0	N/A	N/A	9099	G	...RendererForSitePerProcess	47MiB	
0	N/A	N/A	75265	C	...nda3/envs/4bit/bin/python	22605MiB	
0	N/A	N/A	85563	G	...135759701571808370,131072	207MiB	
0	N/A	N/A	93120	G	...veSuggestionsOnlyOnDemand	43MiB	
1	N/A	N/A	4335	G	/usr/lib/xorg/Xorg	4MiB	
1	N/A	N/A	75266	C	...nda3/envs/4bit/bin/python	22605MiB	



**SEVERAL
DAYS
LATER**



GPUs go Brrrrr....

Function Summary:

This function appears to be a part of a malware that is designed to perform a service-based attack. The function takes three parameters: a handle to a service, a pointer to a wide-character string, and an integer. The function first opens a service manager and then opens a service using the handle and the wide-character string. It then checks if the service is running and if it is not, it changes the service configuration and starts the service. The function then queries the service status and waits for the service to start. If the service does not start within a certain time limit, the function calls another function that is likely to be a part of the malware.

New Function Name:

ServiceStart

[INST]<<SYS>> You are an advanced malware reverse engineer capable of understanding decompiled C code and identifying malicious functionality<</SYS>>

You must output a descriptive ### Function Summary that describes the following decompiled code followed by a descriptive ### New Function Name

Do not provide any extra information in the ### New Function Name. Only provide the name of the function. Do not include any extra information.[/INST]

Code:

```
void __fastcall fcn.004029c9(uint param_1, uint param_2, int32_t param_3)
{
    int32_t iVar1;
    int32_t iVar2;
    int32_t iVar3;
    int32_t iVar4;
    bool bVar5;
    uchar auStack_30 [4];
    int32_t iStack_2c;
    uint32_t uStack_18;
    uchar auStack_c [4];
    uint32_t uStack_8;

    uStack_8 = *0x423008 ^ &stack0xfffffffffc;
    bVar5 = false;
    iVar1 = (*_sym.imp.ADVAPI32.dll_OpenSCManagerW)(0, 0, 0xf003f);
    if (iVar1 != 0) {
        iVar2 = (*_sym.imp.ADVAPI32.dll_OpenServiceW)(iVar1, param_2, 0x2c);
        if (iVar2 != 0) {
            if (param_3 != -1) {
                iVar3 = (*_sym.imp.ADVAPI32.dll_ChangeServiceConfigW)
                    (iVar2, 0xffffffff, param_3, 0xffffffff, 0, 0, 0, 0, 0);
                bVar5 = iVar3 != 0;
            }
            iVar3 = (*_sym.imp.ADVAPI32.dll_QueryServiceStatusEx)(iVar2, 0, auStack_30, 0x24, auStack_c);
            if (((iVar3 != 0) && (iStack_2c != 1)) &&
                (iVar3 = (*_sym.imp.ADVAPI32.dll_ControlService)(iVar2, 1, auStack_30), iVar3 != 0)) {
                iVar3 = (*_sym.imp.KERNEL32.dll_GetTickCount());
                if (15000 < uStack_18) {
                    uStack_18 = 15000;
                }
                if (iStack_2c != 1) {
                    while( true ) {
                        (*_sym.imp.KERNEL32.dll_Sleep)(uStack_18);
                        iVar4 = (*_sym.imp.ADVAPI32.dll_QueryServiceStatusEx)(iVar2, 0, auStack_30, 0x24, auStack_c);

                        if (iVar4 == 0) break;
                        if (iStack_2c == 1) {
                            bVar5 = true;
                            break;
                        }
                        iVar4 = (*_sym.imp.KERNEL32.dll_GetTickCount());
                        if ((15000 < iVar4 - iVar3) || (iStack_2c == 1)) break;
                    }
                }
                if (!bVar5) {
                    fcn.00402821();
                }
            }
            (*_sym.imp.ADVAPI32.dll_CloseServiceHandle)(iVar2);
        }
        (*_sym.imp.ADVAPI32.dll_CloseServiceHandle)(iVar1);
    }
    fcn.00408842();
    return;
}
```

Function Summary:

This function appears to be a part of a malware that is designed to perform a service-based attack. The function takes three parameters: a handle to a service, a pointer to a wide-character string, and an integer. The function first opens a service manager and then opens a service using the handle and the wide-character string. It then checks if the service is running and if it is not, it changes the service configuration and starts the service. The function then queries the service status and waits for the service to start. If the service does not start within a certain time limit, the function calls another function that is likely to be a part of the malware.

New Function Name:

ServiceStart



GPUs go Brrrrr....

```
void MaliciousFunction(void)
{
    WCHAR WVar1;
    int *piVar2;
    bool bVar3;
    int iVar4;
    DWORD nBufferLength;
    LPWSTR lpBuffer;
    UINT UVar5;
    int iVar6;
    LPWSTR lpString1;
    HANDLE pvVar7;
    int iVar8;
    undefined4 extraout_ECX;
    undefined4 extraout_ECX_00;
    undefined4 uVar9;
    undefined4 extraout_EDX;
    undefined4 extraout_EDX_00;
    undefined4 uVar10;
    LPCWSTR lpRootPathName;
    int iVar11;
    uint uVar12;
    int *piVar13;
    undefined4 uStack_1e0;
    WCHAR aWStack_1dc [34];
    undefined local_198 [4];
    undefined auStack_194 [400];

    memset(local_198,0,400);
    DAT_0042f894 = LoadAndExecuteKernel32Function(0x16d34cd7,&DAT_0042f0fc,0x14);
    iVar4 = Ordinal_115(0x202,auStack_194);
    if (iVar4 != 0) {
        Ordinal_116();
    }
    nBufferLength = GetLogicalDriveStringsW(0,(LPWSTR)0x0);
    lpBuffer = (LPWSTR)FUN_00415650(nBufferLength * 2);
    GetLogicalDriveStringsW(nBufferLength,lpBuffer);
    WVar1 = *lpBuffer;
    lpRootPathName = lpBuffer;
    while (WVar1 != L'\0') {
        bVar3 = false;
        UVar5 = GetDriveTypeW(lpRootPathName);
        wprintf(aWStack_1dc,L"\\?\\%2s\\",lpRootPathName);
        if (((UVar5 == 2) || (UVar5 == 3)) || (UVar5 == 4)) {
            if ((DAT_004342e8 != 0) && (iVar4 == 0, 0 < DAT_004342e8)) {
                do {
                    iVar6 = BoyerMooreSearch_1((LPCWSTR)(&DAT_00435700)[iVar4],aWStack_1dc);
                    if (iVar6 != 0) {
                        bVar3 = true;
                    }
                    iVar4 = iVar4 + 1;
                } while (iVar4 < DAT_004342e8);
                if (bVar3) goto LAB_004160fc;
            }
            lpString1 = (LPWSTR)FUN_00415650(0x104);
            (&DAT_00435700)[DAT_004342e8] = lpString1;
            lstrcpyW(lpString1,aWStack_1dc);
            pvVar7 = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,FUN_0041ab70,
                (LPVOID)(&DAT_00435700)[DAT_004342e8],0,
                (LPDWORD)(&DAT_00435300 + DAT_004342ec * 4));
            *(HANDLE *)(&DAT_004342f0 + DAT_004342ec * 4) = pvVar7;
            DAT_004342ec = DAT_004342ec + 1;
            DAT_004342e8 = DAT_004342e8 + 1;
        }
        LAB_004160fc:
        iVar4 = lstrlenW(lpRootPathName);
        lpRootPathName = lpRootPathName + iVar4 + 1;
        WVar1 = *lpRootPathName;
    }
    FUN_00415630((int)lpBuffer);
}
```

```
if (iVar6 != 0x6f) {
    iVar11 = iVar4;
    if (iVar6 == 0) {
        do {
            for (iVar6 = *(int *)(&iVar11 + 0x10); iVar6 != 0; iVar6 = *(int *)(&iVar6 + 8)) {
                iVar8 = Ordinal_12(0,undefined4 *)(&iVar6 + 0xc) + 4);
                if (iVar8 != 0) {
                    FUN_004202d0(iVar8);
                }
            }
            for (iVar6 = *(int *)(&iVar11 + 0x14); iVar6 != 0; iVar6 = *(int *)(&iVar6 + 8)) {
                iVar8 = Ordinal_12(0,undefined4 *)(&iVar6 + 0xc) + 4);
                if (iVar8 != 0) {
                    FUN_004202d0(iVar8);
                }
            }
            piVar2 = (int *)(&iVar11 + 8);
            iVar11 = *piVar2;
        } while (*piVar2 != 0);
        FUN_00415630(iVar4);
    }
    break;
}
FUN_00415630(iVar4);
uVar12 = uVar12 + 1;
} while (uVar12 < 3);
iVar4 = 0;
if (0 < DAT_004346f0) {
    do {
        NetworkShareEnumerator((char *)(&DAT_00434700)[iVar4]);
        iVar4 = iVar4 + 1;
    } while (iVar4 < DAT_004346f0);
}
bVar3 = false;
do {
    Sleep(240000);
    if ((5 < DAT_004340e4) && (iVar4 == 0, 0 < DAT_004346f0)) {
        do {
            NetworkShareEnumerator((char *)(&DAT_00434700)[iVar4]);
            iVar4 = iVar4 + 1;
        } while (iVar4 < DAT_004346f0);
    }
    iVar4 = DAT_004342ec;
    piVar2 = *(int *)(&iVar11 + 0);
    piVar13 = piVar2;
    do {
        piVar13 = (int *)piVar13;
        if ((piVar13[6] != 0) &&
            (iVar6 = lstrcmpW((LPCWSTR)piVar13[0xc],L"kernel32.dll"), iVar6 == 0)) break;
    } while (piVar2 != piVar13);
    DAT_00436f0c = (code *)HashTableLookup(piVar13[6],0x1f3ffdb,0xd,0x7d10e76b);
    iVar6 = (* DAT_00436f0c)(&DAT_0042f0fc);
    DAT_004340c4 = (code *)HashTableLookup(iVar6,0xadc9faf4,0x17,0x7e990fb0);
    (* DAT_004340c4)(iVar4,&DAT_004342f0,1,0xffffffff);
    DAT_004342ec = 0;
    if (!bVar3) {
        bVar3 = true;
        iVar4 = FUN_00419b00();
        uVar9 = extraout_ECX;
        uVar10 = extraout_EDX;
        if (iVar4 != 0) {
            ExecuteCmdShell(&DAT_0042f040,0x91);
            uVar9 = extraout_ECX_00;
            uVar10 = extraout_EDX_00;
        }
        FUN_00419bc0(uVar9,uVar10);
    }
} while( true );
}
```



GPUs go Brrrrr....

Function Summary:

This function is a malicious function that is designed to perform a variety of malicious activities, including:

1. Loading and executing a kernel32.dll function
2. Enumerating logical drives and creating threads to scan for specific files
3. Enumerating network resources and creating threads to scan for specific files
4. Enumerating network shares and creating threads to scan for specific files
5. Creating a new thread to perform a malicious activity

New Function Name:

MaliciousFunction

summary match name match



```
1
2 /* WARNING: Globals starting with '_' overlap smaller symbols at the same address */
3
4 void entry(void)
5 {
6
7     int *piVar1;
8     code *pcVar2;
9     LPWSTR pWVar3;
10    bool bVar4;
11    int iVar5;
12    LPWSTR piWVar6;
13    DWORD DVar7;
14    BOOL BVar8;
15    ushort *puVar9;
16    undefined3 extraout_var;
17    undefined4 uVar10;
18    HANDLE hHeap;
19    LPSTR lpMultiByteStr;
20    undefined4 extraout_ECX;
21    undefined4 extraout_ECX_00;
22    undefined4 extraout_ECX_01;
23    undefined4 extraout_ECX_02;
24    undefined4 extraout_ECX_03;
25    undefined4 extraout_ECX_04;
26    undefined4 extraout_ECX_05;
27    undefined4 extraout_ECX_06;
28    undefined4 extraout_ECX_07;
29    undefined4 extraout_ECX_08;
30    undefined4 extraout_ECX_09;
31    undefined4 extraout_ECX_10;
32    undefined4 extraout_ECX_11;
33    undefined4 extraout_ECX_12;
34    undefined4 extraout_ECX_13;
35    int iVar11;
36    undefined4 extraout_ECX_14;
37    undefined4 extraout_ECX_15;
38    undefined4 extraout_ECX_16;
39    undefined4 extraout_ECX_17;
40    undefined4 extraout_ECX_18;
41    undefined4 extraout_ECX_19;
42    undefined4 extraout_EDX;
43    uint uVar12;
44    undefined4 extraout_EDX_00;
45    undefined4 extraout_EDX_01;
46    undefined4 extraout_EDX_02;
47    char *pcVar13;
48    uint uVar14;
49    int *piVar15;
50    undefined8 uVar16;
51    SIZE_T dwBytes;
52    undefined *puVar17;
53    undefined *puStack_370;
54    undefined auStack_36c [4];
55    undefined auStack_368 [16];
56    undefined auStack_358 [16];
57    void *pvStack_348;
58    WCHAR aWStack_334 [8];
59    undefined auStack_324 [20];
60    WCHAR aWStack_310 [2];
61    undefined auStack_30c [2];
62    undefined auStack_30a [518];
63    undefined auStack_104 [248];
64    undefined4 uStack_c;
65
66    memset(&DAT_004340e4,0,0x118);
67    _DAT_004340e0 = 0x11c;
68    GetVersionExW((LPOSVERSIONINFO) &DAT_004340e0);
69    uVar16 = FUN_004010a4(extraout_ECX,extraout_EDX,0x42f0fc,0xd,(undefined *) &DAT_0042f47c,3);
70    uVar16 = FUN_004010a4(extraout_ECX_00,(int)((ulonglong)uVar16 >> 0x20),0x42f14c,9,
71        (undefined *) &DAT_0042f47c,3);
72    _DAT_00436f80 = DAT_0042f510;
73    DAT_0042f510 = DAT_0042f514;
74    _DAT_0042f518 = 0;
75    DAT_0042f514 = 0;
76    DAT_00436920 = 0;
77    uVar16 = FUN_004010a4(extraout_ECX_01,(int)((ulonglong)uVar16 >> 0x20),0x42f138,0xd,
78        (undefined *) &DAT_0042f47c,3);
79    _DAT_00436f80 = DAT_0042f510;
80    DAT_0042f510 = DAT_0042f514;
81    _DAT_0042f518 = 0;
82    DAT_0042f514 = 0;
83    uVar16 = FUN_004010a4(extraout_ECX_02,(int)((ulonglong)uVar16 >> 0x20),0x42f418,0xd,
84        (undefined *) &DAT_0042f47c,3);
85    _DAT_00436f80 = DAT_0042f510;
86    DAT_0042f510 = DAT_0042f514;
87    _DAT_0042f518 = 0;
```

```
87    _DAT_0042f518 = 0;
88    DAT_0042f514 = 0;
89    DAT_00436920 = 0;
90    uVar16 = FUN_004010a4(extraout_ECX_03,(int)((ulonglong)uVar16 >> 0x20),0x42f110,0xd,
91        (undefined *) &DAT_0042f47c,3);
92    uVar12 = (uint)((ulonglong)uVar16 >> 0x20);
93    for (uVar14 = DAT_00436920; 9 < uVar14; uVar14 = uVar14 % 10) {
94        uVar12 = uVar14 / 10;
95    }
96    DAT_00436920 = uVar14;
97    FUN_004010a4(extraout_ECX_04,uVar12,0x42f468,0xb,(undefined *) &DAT_0042f47c,3);
98    FUN_00418d90(extraout_ECX_05,uVar14);
99    _DAT_00436f80 = DAT_0042f510;
100   DAT_0042f510 = DAT_0042f514;
101   _DAT_0042f518 = 0;
102   DAT_0042f514 = 0;
103   DAT_00436920 = 0;
104   uVar16 = FUN_004010a4(extraout_ECX_06,extraout_EDX_00,0x42f124,0xd,(undefined *) &DAT_0042f47c,3);
105   uVar16 = FUN_004010a4(extraout_ECX_07,(int)((ulonglong)uVar16 >> 0x20),0x42f010,0xc,
106       (undefined *) &DAT_0042f47c,3);
107   uVar16 = FUN_004010a4(extraout_ECX_08,(int)((ulonglong)uVar16 >> 0x20),0x42f42c,0xc,
108       (undefined *) &DAT_0042f47c,3);
109   FUN_004010a4(extraout_ECX_09,(int)((ulonglong)uVar16 >> 0x20),0x42f024,0xb,
110       (undefined *) &DAT_0042f47c,3);
111   FUN_00418d90(extraout_ECX_10,3);
112   _DAT_00436f80 = 0;
113   DAT_0042f510 = 0;
114   _DAT_0042f518 = 0;
115   DAT_0042f514 = 0;
116   DAT_00436920 = 0;
117   FUN_004010a4(extraout_ECX_11,extraout_EDX_01,0x42f440,0xc,(undefined *) &DAT_0042f47c,3);
118   FUN_00418d90(extraout_ECX_12,0xf);
119   FUN_004010a4(extraout_ECX_13,extraout_EDX_02,0x42f0e8,0xc,(undefined *) &DAT_0042f47c,3);
120   puStack_370 = DAT_0042f47c;
121   DAT_0042fee0 = FUN_00415690(0x2ba92ec1,&DAT_0042f14c,0xc);
122   uVar12 = 0;
123   uVar14 = uVar12;
124   do {
125       for (uVar12 < 5; uVar12 = uVar12 + 1) {
126           if (uVar14 != uVar12) {
127               auStack_36c = (undefined [4])0x0;
128           }
129       }
130       uVar12 = uVar14 + 1;
131       uVar14 = uVar12;
132   } while (uVar12 < 5);
133   piVar1 = *(int **)((int *)((int)ProcessEnvironmentBlock + 0xc) + 0x10);
134   piVar15 = piVar1;
135   do {
136       piVar15 = (int *)piVar15;
137       if ((piVar15[6] != 0) && (iVar5 = lstrcpw((LPWSTR)piVar15[0xc],L"kernel32.dll"), iVar5 == 0))
138           break;
139   } while (piVar1 != piVar15);
140   _DAT_00436f0c = (code *)FUN_00415700(piVar15[6],0x1f3ffddb,0xd,0x7d0e76b);
141   iVar5 = (*_DAT_00436f0c)(&DAT_0042f0fc);
142   _DAT_00434254 = (code *)FUN_00415700(iVar5,0x4dd730b7,0xf,0x36ca1fd);
143   DAT_0042f550 = (*_DAT_00434254)();
144   iVar5 = 5;
145   do {
146       puStack_370 = (undefined *)((uint)puStack_370 ^ 0x4a108947);
147       iVar5 = iVar5 + -1;
148   } while (iVar5 != 0);
149   piVar1 = *(int **)((int *)((int)ProcessEnvironmentBlock + 0xc) + 0x10);
150   piVar15 = piVar1;
151   do {
152       piVar15 = (int *)piVar15;
153       if ((piVar15[6] != 0) && (iVar5 = lstrcpw((LPWSTR)piVar15[0xc],L"kernel32.dll"), iVar5 == 0))
154           break;
155   } while (piVar1 != piVar15);
156   _DAT_00436f0c = (code *)FUN_00415700(piVar15[6],0x1f3ffddb,0xd,0x7d0e76b);
157   iVar5 = (*_DAT_00436f0c)(&DAT_0042f14c);
158   DAT_00434214 = FUN_00415700(iVar5,0xc9a2a74c,0x10,0x5bfa915f);
159   piWVar6 = GetCommandLine();
160   DAT_0042fee4 = (code *)FUN_00415690(0x9b4c2d0,&DAT_0042f0e8,0x13);
161   (*_DAT_0042fee4)(pWVar6,auStack_368);
162   if (((S < _DAT_004340e4) && ((int)puStack_370 < 3)) && (iVar5 = FUN_00417540()), iVar5 != 0) &&
163       (iVar5 = FUN_00419b00(), iVar5 != 0)) {
164       DAT_00434214 = (code *)FUN_00415690(0x2a13bb3,&DAT_0042f418,0xe);
165       (*DAT_00434214)(0x80000001,L"Software\\Microsoft\\Windows\\CurrentVersion\\Run",0,0xf003f,
166           &puStack_370);
167       DVar7 = GetModuleFileNameW((HMODULE)0x0,aWStack_310,0x104);
168       if (DVar7 == 0) {
169           FUN_0041b980(0);
170       }
171   } else {
172       iVar5 = 0;
173       iVar11 = 0;
```

```
191    DAT_00434224 = (code *)FUN_00415690(0x2a5ebfb7,&DAT_0042f418,0xc);
192    (*_DAT_00434224)(puVar17);
193    BVar8 = ShellExecuteExW((SHELLEXECUTEINFO) &auStack_36c);
194    if (BVar8 != 0) {
195        /* WARNING: Subroutine does not return */
196        ExitProcess(0x65);
197    }
198
199 }
200
201 DAT_00436928 = (LPWSTR)FUN_00415610(0x80);
202 DAT_0042fef4 = (code *)FUN_00415690(0xaff20ccb,&DAT_0042f0fc,9);
203 (*DAT_0042fef4)(auStack_104,&DAT_00425170);
204 puVar9 = FUN_0041c220();
205 uVar12 = 0;
206 do {
207     wprintfw(aWStack_334,L"%x",(uint)puVar9[uVar12]);
208     uVar12 = uVar12 + 1;
209 } while (uVar12 < 5);
210 if (5 < _DAT_004340e4) {
211     FUN_0041e130();
212 }
213 pWVar3 = DAT_00436928;
214 _DAT_004342a8 = (code *)FUN_00415690(0x76a4fa0b,&DAT_0042f0fc,0xb);
215 iVar5 = (*_DAT_004342a8)(0x1f0001,0,pWVar3);
216 pWVar3 = DAT_00436928;
217 if (iVar5 != 0) {
218     /* WARNING: Subroutine does not return */
219     ExitProcess(0);
220 }
221 _DAT_00434240 = (code *)FUN_00415690(0xed3e7aaa,&DAT_0042f0fc,0xd);
222 (*_DAT_00434240)(0,0,pWVar3);
223 FUN_00402030();
224 FUN_00415690(0xa38e7de7,&DAT_0042f0fc,0x1b);
225 FUN_00415690(0x97622898,&DAT_0042f0fc,10);
226 bVar4 = FUN_004192e0();
227 iVar5 = CONCAT31(extraout_var,bVar4);
228 if (iVar5 == 0) {
229     return;
230 }
231 _DAT_0042f880 = (code *)FUN_00415690(0x48b64e68,&DAT_0042f0fc,0x12);
232 uVar10 = (*_DAT_0042f880)();
233 _DAT_004342b0 = (code *)FUN_00415690(0xf0df01cc,&DAT_0042f0fc,0x11);
234 (*_DAT_004342b0)(uVar10,0x80);
235 PTR_DAT_0042f038 = (undefined *)FUN_00415650(0x100);
236 wprintfw((LPWSTR)PTR_DAT_0042f038,L"%s.%s",_DAT_00436928,L"anobtanium@tutanota.com",L".pandora"
237 );
238 iVar11 = FUN_00419b00();
239 if (iVar11 != 0) {
240     FUN_004198a0(&DAT_0042f160,0x2b5);
241 }
242 iVar11 = lstrlen((LPWSTR)PTR_DAT_0042f038);
243 DAT_00436930 = iVar11 * 2 + 2;
244 DAT_00434200 = FUN_00415690(0xac74699e,&DAT_0042f0fc,0xc);
245 DAT_0043692c = (LPCSTR)FUN_00415650(0x1fa0);
246 FUN_00415610(0x1000);
247 dwBytes = 0x100;
248 DVar7 = 8;
249 hHeap = GetProcessHeap();
250 lpMultiByteStr = (LPSTR)HeapAlloc(hHeap,DVar7,dwBytes);
251 if (iVar5 == 1) {
252     FUN_0041bc60();
253     FUN_0041be00(extraout_ECX_15,&DAT_00436720);
254     pcVar13 = "<br><strong><p></div></body>";
255     uVar10 = extraout_ECX_16;
256 } else {
257     uVar10 = extraout_ECX_14;
258     if (iVar5 != 3) goto LAB_004186f9;
259     DVar7 = 0;
260     WideCharToMultiByte(0,0,DAT_00436928,-1,lpMultiByteStr,0x80,(LPCSTR)0x0,(LPBOOL)0x0);
261     FUN_0041bc60();
262     FUN_0041be00(extraout_ECX_17,lpMultiByteStr);
263     pcVar13 = "<br><strong><p></div></body>";
264     uVar10 = extraout_ECX_18;
265 }
266 FUN_0041be00(uVar10,pcVar13);
267 uVar10 = extraout_ECX_19;
268 LAB_004186f9:
269     FUN_0041c0b0(uVar10,uStack_c,DVar7);
270     FUN_00417d90();
271     DAT_00436934 = lstrlenA(DAT_0043692c);
272     pcVar2 = (code *)swi(3);
273     (*pcVar2)();
274     return;
275
276
277
```

```

208 } while (uVar12 < 5);
209 if (5 < _DAT_004340e4) {
210     FUN_0041e130();
211 }
212 pWVar3 = DAT_00436928;
213 _DAT_004342a8 = (code *)FUN_00415690(0x76a4fa0b,&DAT_0042f0fc,0xb);
214 iVar5 = (*_DAT_004342a8)(0x1f0001,0,pWVar3);
215 pWVar3 = DAT_00436928;
216 if (iVar5 != 0) {
217     /* WARNING: Subroutine does not return */
218     ExitProcess(0);
219 }
220 _DAT_00434240 = (code *)FUN_00415690(0xed3e7aaa,&DAT_0042f0fc,0xd);
221 (*_DAT_00434240)(0,0,pWVar3);
222 FUN_00402030();
223 FUN_00415690(0xa38e7de7,&DAT_0042f0fc,0x1b);
224 FUN_00415690(0x97622898,&DAT_0042f0fc,10);
225 bVar4 = FUN_004192e0();
226 iVar5 = CONCAT31(extraout_var,bVar4);
227 if (iVar5 == 0) {
228     return;
229 }
230 _DAT_0042f880 = (code *)FUN_00415690(0x48b64e68,&DAT_0042f0fc,0x12);
231 uVar10 = (*_DAT_0042f880)();
232 _DAT_004342b0 = (code *)FUN_00415690(0xf0df01cc,&DAT_0042f0fc,0x11);
233 (*_DAT_004342b0)(uVar10,0x80);
234 PTR_DAT_0042f038 = (undefined *)FUN_00415650(0x100);
235 wsprintfw((LPWSTR)PTR_DAT_0042f038,L"%s,%s",DAT_00436928,L"anobtanium@tutanota.com",L".pandora"
236 );
237 iVar11 = FUN_00419b00();
238 if (iVar11 != 0) {
239     FUN_004198a0(&DAT_0042f160,0x2b5);
240 }
241 iVar11 = strlenW((LPCWSTR)PTR_DAT_0042f038);
242 DAT_00436930 = iVar11 * 2 + 2;
243 DAT_00434200 = FUN_00415690(0xac74699e,&DAT_0042f0fc,0xc);
244 DAT_0043692c = (LPCSTR)FUN_00415650(0x1fa0);
245 FUN_00415610(0x1000);
246 dwBytes = 0x100;
247 DVar7 = 8;
248 hHeap = GetProcessHeap();
249 lpMultiByteStr = (LPSTR)HeapAlloc(hHeap,DVar7,dwBytes);
250 if (iVar5 == 1) {
251     FUN_0041bc60();
252     FUN_0041be00(extraout_ECX_15,&DAT_00436720);
253     pcVar13 = "<br></strong></p></div></body>";
254     uVar10 = extraout_ECX_16;
255 }

```



Summary

1. Initializing and manipulating some global variables.
2. Gathering version information about the operating system.
3. Performing some repetitive calls to `'FUN_004010a4'` with different parameters.
4. Checking certain conditions based on the operating system version and other data.
5. Interacting with the `'kernel32.dll'` module.
6. Formatting strings, potentially building paths or filenames.
7. Conditionally executing certain blocks of code based on computed or retrieved values.
8. Making use of the `'ShellExecuteExW'` function, which suggests it may be launching external processes or commands.
9. Interacting with the system's heap for memory allocations.
10. Possibly interacting with the system's registry (based on the presence of strings like `'"Software\\Microsoft\\Windows\\CurrentVersion\\Run"'`).
1. Formatting and building an email address.
2. Using `'HeapAlloc'` to allocate memory on the heap.
3. Converting wide characters to multibyte character strings.
4. Handling various conditions based on the value of the variable `'iVar5'`.

Hey there it looks like you're trying to reverse engineer a function...



Step-by-Step Description

1. Initializes a memory region (``memset(&DAT_004340e4,0,0x118)``).
2. Retrieves the version of the operating system using ``GetVersionExW``.
3. Calls the function ``FUN_004010a4`` multiple times with various parameters.
4. Manipulates and checks certain global variables and data points.
5. Interacts with the ``kernel32.dll`` module and retrieves function pointers.
6. Formats and builds strings, especially paths or filenames.
7. Checks for certain conditions, including operating system version and other computed data.
8. Potentially executes shell commands or processes using ``ShellExecuteExW``.
9. Retrieves the command line of the current process using ``GetCommandLineW``.
10. Makes additional calls to external functions (``FUN_00415690``, ``FUN_0041c220``, etc.).
11. Builds an email address using ``wsprintfW``.
12. Checks certain conditions and interacts with the system's registry.
13. Allocates memory on the heap using ``HeapAlloc``.
14. Converts wide character strings to multibyte strings using ``WideCharToMultiByte``.
15. Further manipulates and processes strings and data.
16. Makes additional function calls and possibly ends the process using ``ExitProcess``.

Hey there it looks like you're trying to reverse engineer a function...



Suggested Function Names

Based on the operations performed, some suitable new names for the function could be:

- ``initializeAndExecuteShellCommands``
- ``performSystemOperations``
- ``handleRegistryAndShellCommands``
- ``processKernel32operations``

Hey there it looks like you're trying to reverse engineer a function...



Observations and Recommendations

- The function appears to be part of a larger program, and without additional context, its exact purpose remains somewhat obscured.
- The repetitive calls to ``FUN_004010a4`` and interaction with ``kernel32.dll`` suggest this might be some initialization or setup routine.
- The usage of ``ShellExecuteExW`` and interaction with the registry implies that this function has the potential to make significant changes to the system.
- It's recommended to analyze the external functions like ``FUN_004010a4``, ``FUN_00415690``, etc., to get a clearer picture of the overall functionality and goal of this function.

Hey there it looks like you're trying to reverse engineer a function...



```
uint __cdecl fcn.004038f0(uint arg_8h, uint32_t arg_ch, uint32_t arg_10h, uint *arg_14h, uint *arg_18h)
{
    uint *puVar1;
    int32_t iVar2;
    uint uStack72;
    uint uStack68;
    uint uStack64;
    uint uStack60;
    uint uStack56;
    uint uStack52;
    uint uStack48;
    uint uStack44;
    uint uStack40;
    uint uStack36;
    uint uStack32;
    uint uStack28;
    uint uStack24;
    uint uStack20;
    uint uStack16;
    uint uStack12;

    if ((arg_ch | arg_10h) < 0) {
        uStack56 = *arg_14h;
        uStack52 = arg_10h;
        uStack48 = arg_18h;
        uStack44 = arg_14h;
        uStack40 = arg_18h;
        uStack36 = arg_10h;
        uStack32 = arg_18h;
        uStack28 = arg_18h[7];
        uStack72 = 0x61707865;
        uStack20 = *arg_14h;
        uStack16 = arg_14h[1];
        uStack12 = arg_14h[2];
        uStack68 = 0x3320646e;
        uStack64 = 0x79622d32;
        uStack60 = 0x6b206574;
        uStack24 = 0;
        sub.msvcrt.dll_memset(arg_8h, 0, arg_ch);
        fcn.004028f0(arg_8h, arg_ch, arg_10h);
        puVar1 = &uStack72;
        iVar2 = 0x40;
        do {
            *puVar1 = 0;
            puVar1 = puVar1 + 1;
            iVar2 = iVar2 + -1;
        } while (iVar2 != 0);
    }
    return 0;
}
```

Recognizes encryption constants

Step-by-step description

1. The function takes five arguments: `arg_8h`, `arg_ch`, `arg_10h`, `arg_14h` and `arg_18h`.
2. `arg_8h` is a memory address, `arg_ch` and `arg_10h` are integers, `arg_14h` and `arg_18h` are pointers to integers.
3. First, the function checks if `arg_ch` or `arg_10h` are not equal to zero.
4. If either `arg_ch` or `arg_10h` is not zero, function proceeds to do the following operations. Otherwise, it skips to the end, returning 0.
5. It pulls integer values from the memory locations pointed to by `arg_18h` and `arg_14h` and stores them in local variables (`uStack56` to `uStack28` and `uStack20` to `uStack12` respectively)
6. It sets `uStack72` to 0x61707865 (which is "expa" in ASCII form), `uStack68` to 0x3320646e ("2 dn" in ASCII), `uStack64` to 0x79622d32 ("-2yb" in ASCII), and `uStack60` to 0x6b206574 ("te k" in ASCII)
7. It sets `uStack24` to 0
8. The function then clears memory starting from `arg_8h` and extending to `arg_ch` through `memset()`.
9. Then `fcn.004028f0` is executed with `arg_8h`, `arg_ch`, `arg_10h` as its arguments.
10. After `fcn.004028f0` is executed, function clears variables in the stack from `uStack72` to `uStack28` and `uStack20` to `uStack12`, by setting them to zero.
11. Finally, the function returns 0.

Summary

This function seems to perform some form of initial setup or configuration and then cleanup. It seems to be doing some cryptographic operations given the constants that are in ASCII form which resembles constants for Salsa20 / ChaCha encryption algorithms. It doesn't directly seem to engage in networking, lateral movement, or direct exploitation.

Function Name:

prepare_and_cleanup_crypto_config

Interesting Constants

- "expa" (0x61707865)
- "2 dn" (0x3320646e)
- "-2yb" (0x79622d32)
- "te k" (0x6b206574)

Interesting Memory Addresses to dereference

It would be interesting to see the values stored at these memory addresses:

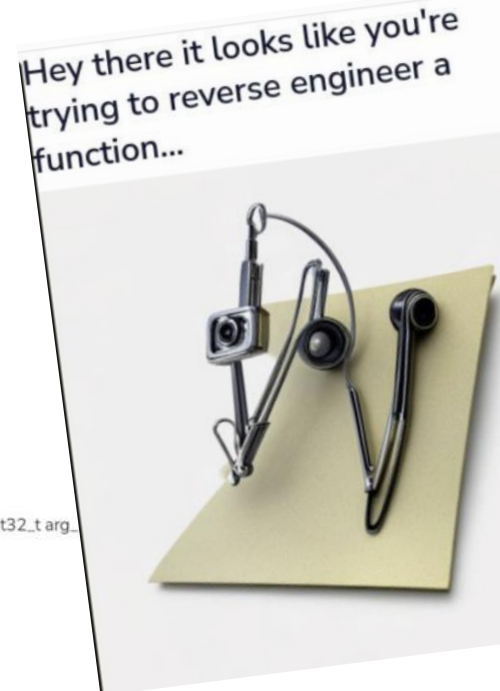
- `arg_8h`
- `arg_14h`
- `arg_18h`

Function Info

- offset: 4208880
- name: fcn.004038f0
- size: 211
- is-pure: false
- realsz: 211
- noretum: false
- stackframe: 88
- calltype: cdecl
- cost: 73
- cc: 3
- bits: 32
- type: fcn
- nbbs: 4
- is-lineal: true
- ninstrs: 65
- edges: 5
- ebbs: 1
- signature: fcn.004038f0 (int32_t arg_8h, int32_t arg_ch, int32_t arg_10h, int32_t arg_14h, int32_t arg_18h);
- minbound: 4208880
- maxbound: 4209091
- indegree: 0
- outdegree: 2
- nlocals: 16
- nargs: 5
- difftype: new

Callrefs

Addr	Type	At
4209083	CODE	4208899
4327813	CALL	4209033
4204784	CALL	4209052
4209072	CODE	4209081



LLMs in the Lab

LLM empowered Malware Analysis.

Function Name:

prepare_and_cleanup_crypto_config

Summary

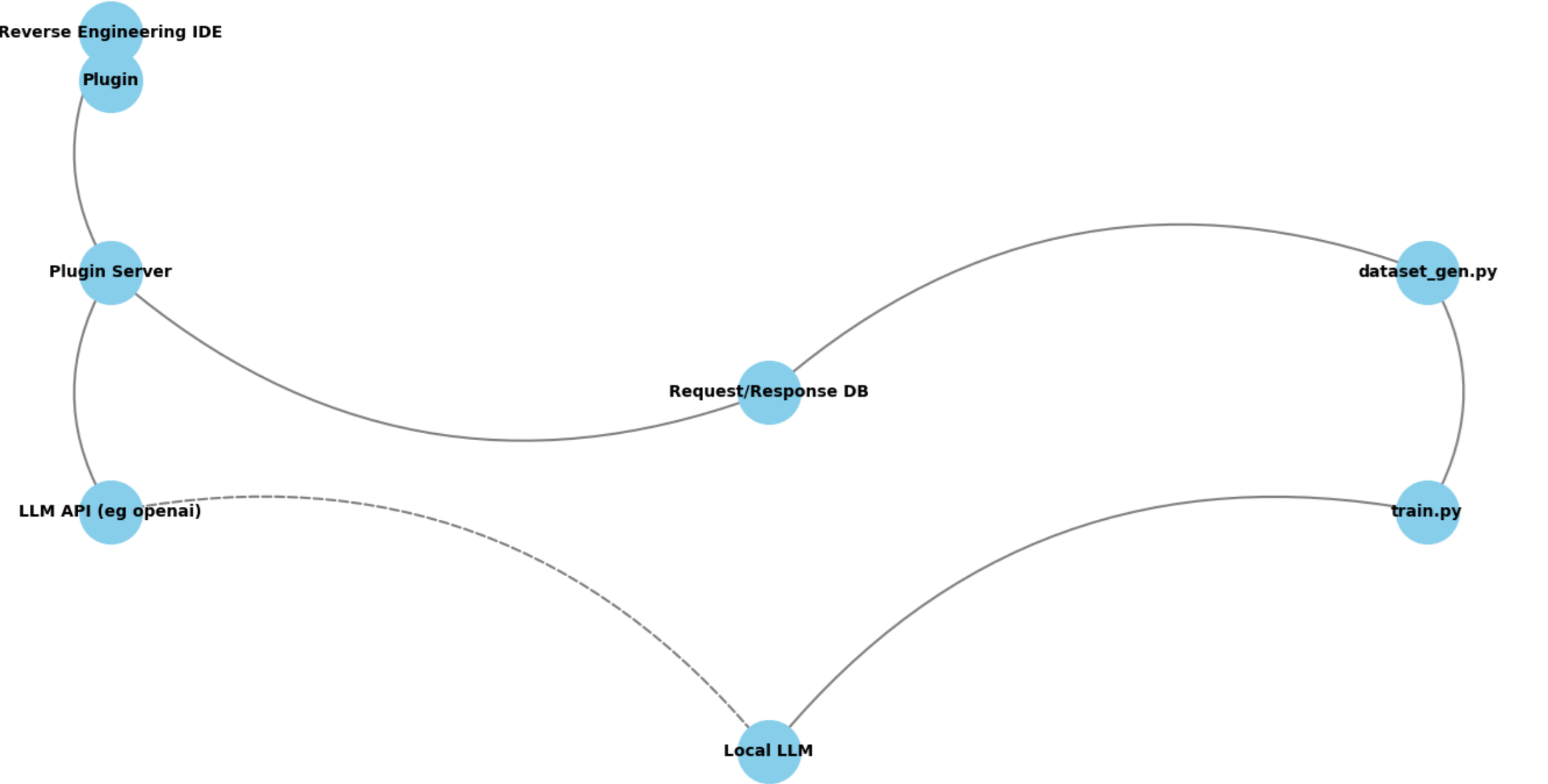
This function seems to perform some form of initial setup or configuration and then cleanup. It seems to be doing some cryptographic operations given the constants that are in ASCII form which resembles constants for Salsa20 / ChaCha encryption algorithms. It doesn't directly seem to engage in networking, lateral movement, or direct exploitation.

Step-by-step description

1. The function takes five arguments: `arg_8h`, `arg_ch`, `arg_10h`, `arg_14h` and `arg_18h`.
2. `arg_8h` is a memory address, `arg_ch` and `arg_10h` are integers, `arg_14h` and `arg_18h` are pointers to integers.
3. First, the function checks if `arg_ch` or `arg_10h` are not equal to zero.
4. If either `arg_ch` or `arg_10h` is not zero, function proceeds to do the following operations. Otherwise, it skips to the end, returning 0.
5. It pulls integer values from the memory locations pointed to by `arg_18h` and `arg_14h` and stores them in local variables (`uStack56` to `uStack28` and `uStack20` to `uStack12` respectively)
6. It sets `uStack72` to 0x61707865 (which is "expa" in ASCII form), `uStack68` to 0x3320646e ("2 dn" in ASCII), `uStack64` to 0x79622d32 ("-2yb" in ASCII), and `uStack60` to 0x6b206574 ("te k" in ASCII)
7. It sets `uStack24` to 0
8. The function then clears memory starting from `arg_8h` and extending to `arg_ch` through `memset()`.
9. Then `fcn.004028f0` is executed with `arg_8h`, `arg_ch`, `arg_10h` as its arguments.
10. After `fcn.004028f0` is executed, function clears variables in the stack from `uStack72` to `uStack28` and `uStack20` to `uStack12`, by setting them to zero.
11. Finally, the function returns 0.



Project Lupine Flowchart



Plugins

Ghidra Plugins

- There are three Ghidra plugins each with their own configured shortcut keys

• **llm.py (CTRL-ALT-L)**

- This script calls your local LLM. It requests a new function name and function description. It renames the function and updates the comments with the description. Your cursor can be anywhere inside the decompiled function that you're interested in.

- The plugin expects api_server.py to be running on localhost on port 8000. Documentation for the API server can be found below.

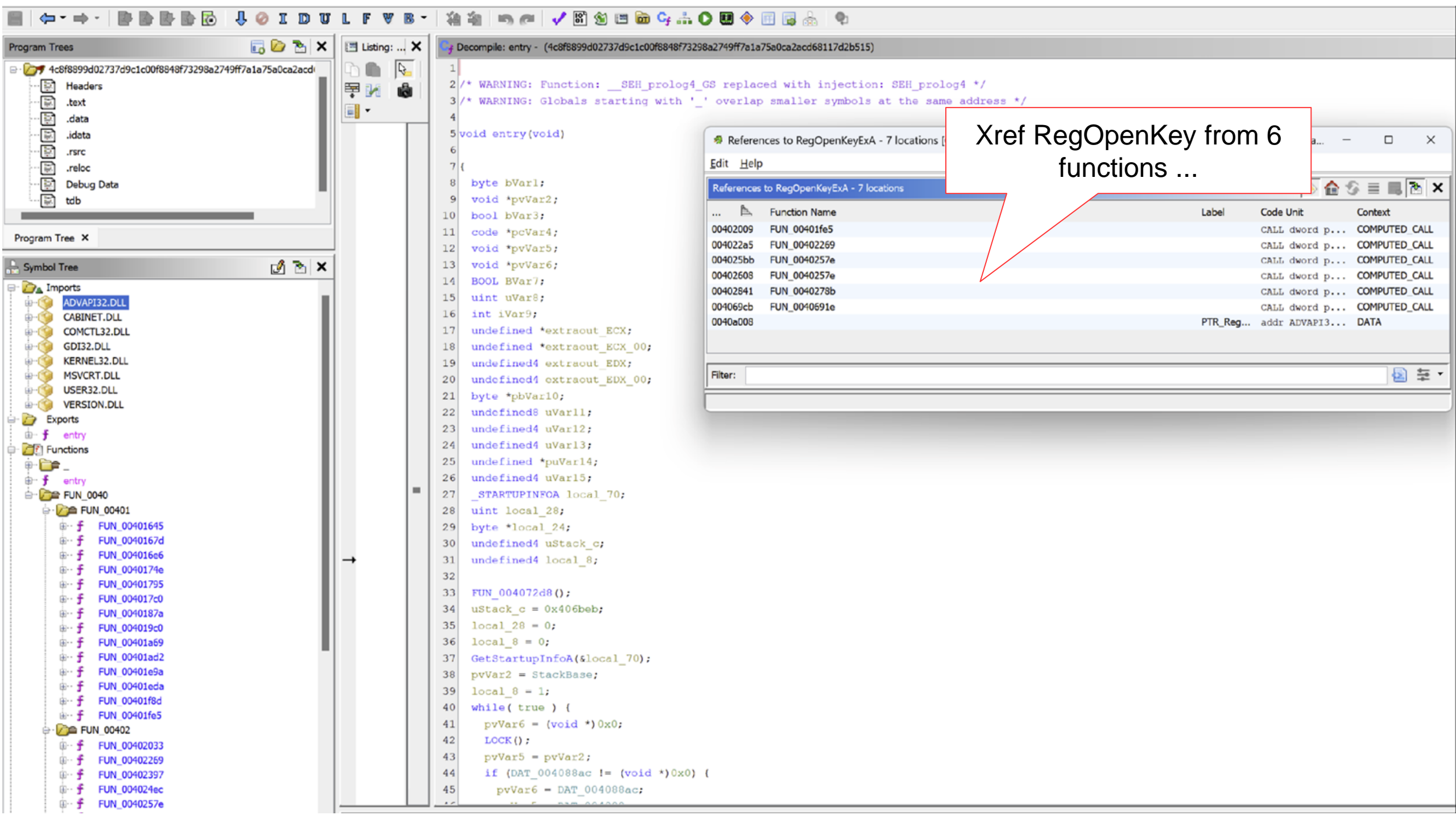
• **llm_remote.pt (CTRL-ALT-O)**

- This script calls the Project Lupine community server. It requests a new function name and function description. It renames the function and updates the comments with the description. Note that it sends the hash, function offset, and decompiled code to the community server.

• **llm_suggest (CTRL-SHIFT-K)**

- This script is useful for contributing back to the community. If you get a summary, function name or step-by-step description that you don't like you can edit the content directly in Ghidra and send your edits back. Note that it sends the hash, function offset, and decompiled code, function name, function comment to the community server





Xref RegOpenKey from 6 functions ...

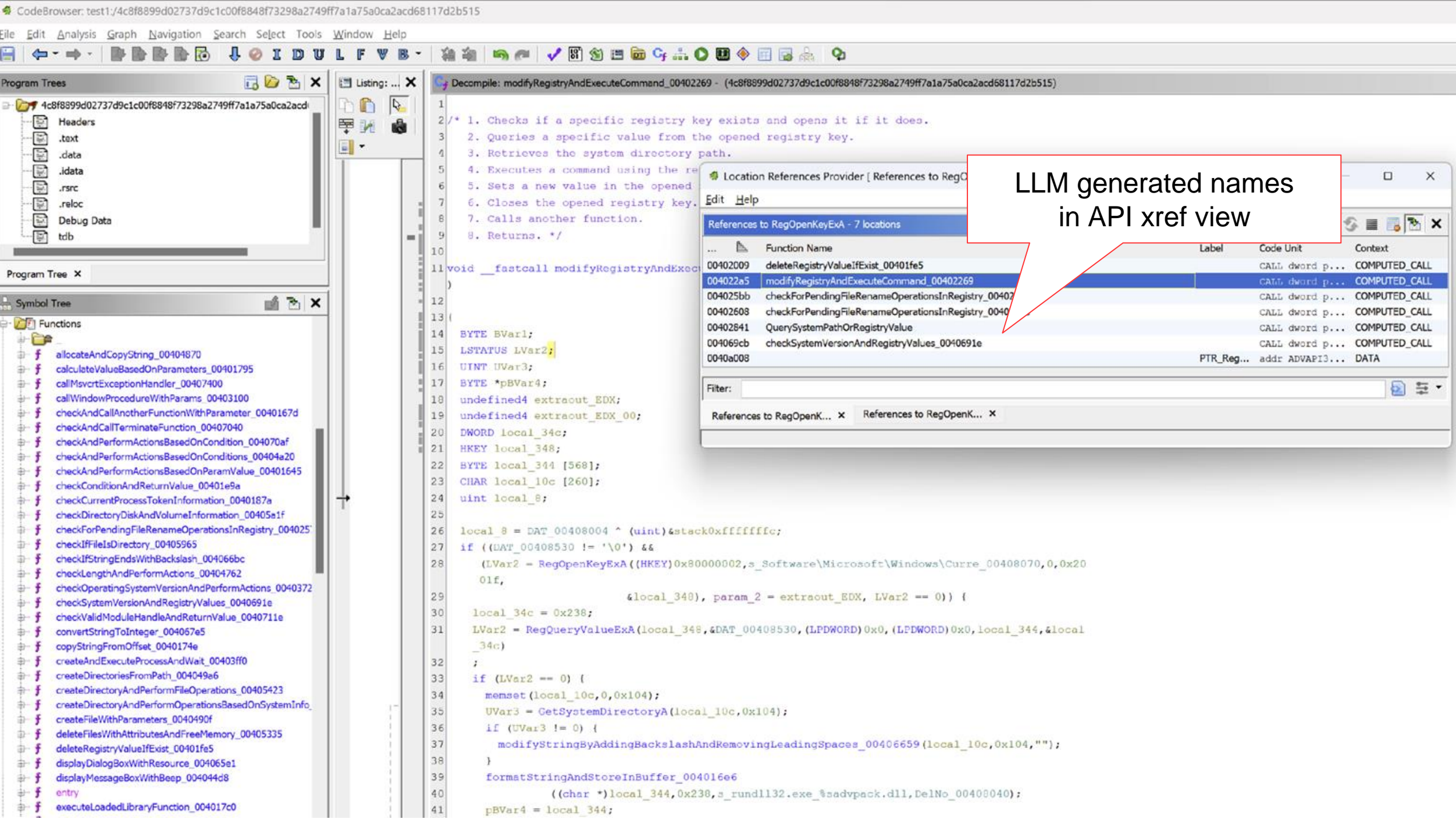
References to RegOpenKeyExA - 7 locations

Edit Help

References to RegOpenKeyExA - 7 locations

...	Function Name	Label	Code Unit	Context
00402009	FUN_00401fe5		CALL dword p...	COMPUTED_CALL
004022a5	FUN_00402269		CALL dword p...	COMPUTED_CALL
004025bb	FUN_0040257e		CALL dword p...	COMPUTED_CALL
00402608	FUN_0040257e		CALL dword p...	COMPUTED_CALL
00402841	FUN_0040278b		CALL dword p...	COMPUTED_CALL
004069cb	FUN_0040691e		CALL dword p...	COMPUTED_CALL
0040a008		PTR_Reg...	addr ADVAPI3...	DATA

Filter:



Ghidra Plugin

Analyst in the loop

- This function name isn't very descriptive...

```
Decompile: MaliciousFunction - (c373d37b5a9427a18dbf93d519968d9fda04f2a262f424d0611830764c8cc69c)
1
2 /* WARNING: Globals starting with '_' overlap smaller symbols at the same address */
3 /* This function is a malicious function that is designed to perform a variety of malicious
4    activities, including:
5
6    1. Loading and executing a kernel32.dll function
7    2. Enumerating logical drives and creating threads to scan for specific files
8    3. Enumerating network resources and creating threads to scan for specific files
9    4. Enumerating network shares and creating threads to scan for specific files
10   5. Creating a new thread to perform a malicious activity */
11
12 void MaliciousFunction(void)
13 {
14     WCHAR WVar1;
15     int *piVar2;
16     bool bVar3;
17     int iVar4;
18     DWORD nBufferLength;
19     LPWSTR lpBuffer;
20     UINT UVar5;
21     int iVar6;
22     LPWSTR lpString1;
23     HANDLE pvVar7;
24     int iVar8;
25     undefined4 extraout_ECX;
26     undefined4 extraout_ECX_00;
27     undefined4 uVar9;
28     undefined4 extraout_EDX;
29     undefined4 extraout_EDX_00;
30     undefined4 uVar10;
31     LPCWSTR lpRootPathName;
32     int iVar11;
33     uint uVar12;
34     int *piVar13;
35     undefined4 uStack_1e0;
36     WCHAR aWStack_1dc [34];
37     undefined local_198 [4];
38     undefined auStack_194 [400];
39
40     memset(local_198,0,400);
41     DAT_0042f894 = LoadAndExecuteKernel32Function(0x16d34cd7,&DAT_0042f0fc,0x14);
42     iVar4 = Ordinal_115(0x202,auStack_194);
43     if (iVar4 != 0) {
44         Ordinal_116();
45     }
46     nBufferLength = GetLogicalDriveStringsW(0,(LPWSTR)0x0);
47     lpBuffer = (LPWSTR)FUN_00415650(nBufferLength * 2);
48     GetLogicalDriveStringsW(nBufferLength,lpBuffer);
49     WVar1 = *lpBuffer;
50     lpRootPathName = lpBuffer;
51     while (WVar1 != L'\0') {
52         bVar3 = false;
53         UVar5 = GetDriveTypeW(lpRootPathName);
54         wsprintfW(aWStack_1dc,L"\\??\\%.2s\\",lpRootPathName);
55         if ((UVar5 == 2) || (UVar5 == 3)) || (UVar5 == 4) {
56             if ((DAT_004342e8 != 0) && (iVar4 = 0, 0 < DAT_004342e8)) {
57                 do {
58
```


Analyst in the loop

- Information Classification: General

Set Comment(s) at Address 00417F50

EOL Comment Pre Comment

Post Comment **Plate Comment** Repeatable Comment

This code is a function that is part of a malware sample. It performs various operations related to the execution of the malware.

1. The function initializes some variables and retrieves the version information of the operating system.
2. It calls several other functions to perform different tasks, such as setting up registry keys, executing shell commands, and allocating memory.
3. Depending on the value of a variable, it executes different code paths to perform specific actions.
4. It converts some strings from wide character format to multi-byte format.
5. It calls additional functions to perform various operations, such as printing output and calculating string lengths.
6. Finally, it calls the system interrupt to exit the process.

Constants

This function references a registry path commonly associated with malware persistence

☐ Enter accepts comment

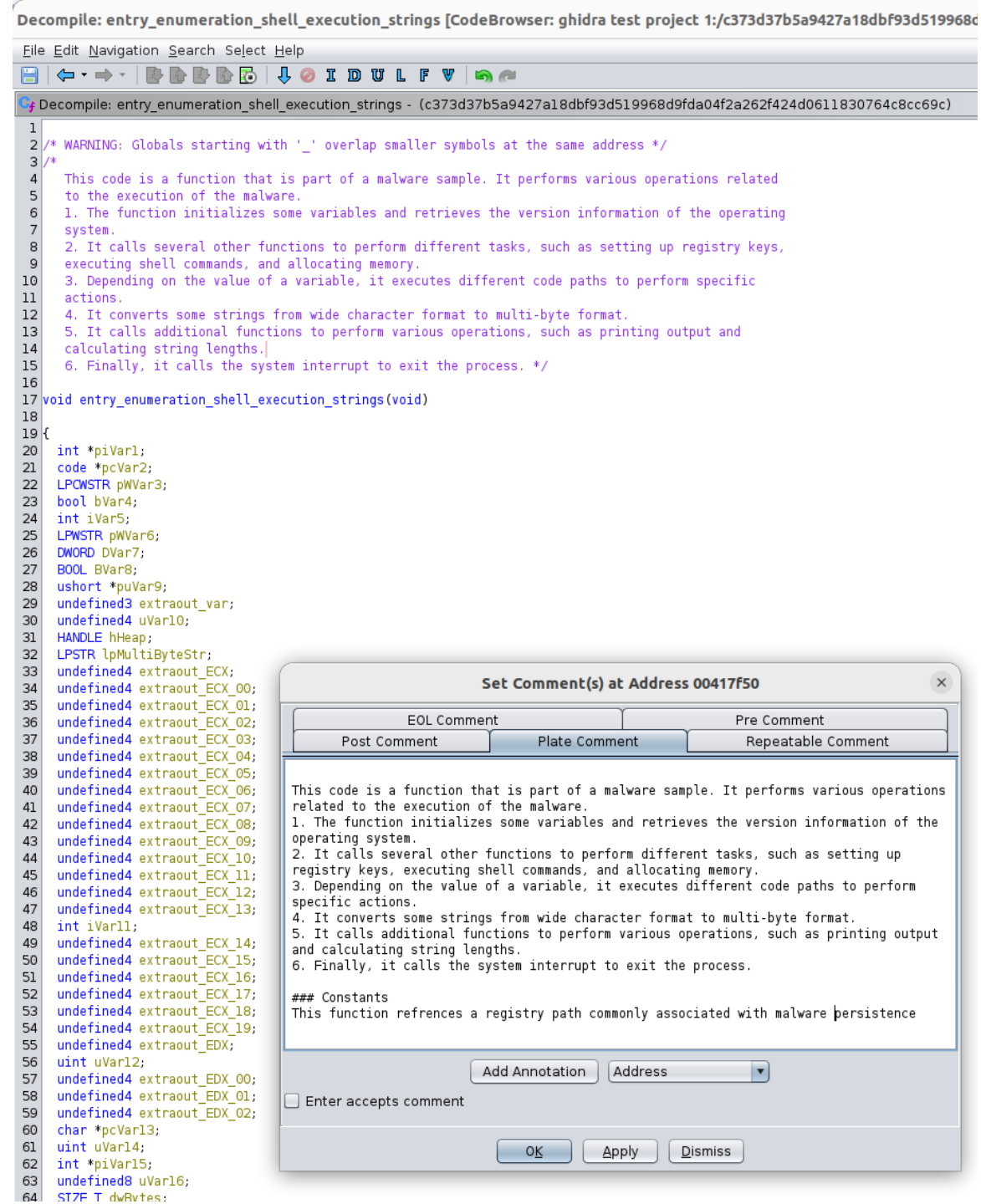
Ghidra Plugin: Feedback

Analyst in the loop

- The Analyst can modify the function name and/or summary description and submit suggestions to the community server.
- CTRL-SHIFT-K

```
Console - Scripting
3. Depending on the value of a variable, it executes different code paths to perform specific actions.
4. It converts some strings from wide character format to multi-byte format.
5. It calls additional functions to perform various operations, such as printing output and calculating string lengths.
6. Finally, it calls the system interrupt to exit the process.

### Constants
This function references a registry path commonly associated with malware persistence
<Response [200]>
Server Response: {'status': 'success', 'message': 'Suggestion added successfully.'}
llm_suggest.py> Finished!
```



Future Work

Analyst in the loop

- YARA/Sigma/Snort
- AutoAnalyst: Using a predefined list of Windows APIs, the auto analyst iterates, gets calling cross xrefs, and pre-decompiles & generates llm analysis results, follows LLM generated tasks from seed exploration, calls dynamic analysis as needed.
- Reporting: Generate/export a report in markdown for threat intelligence analysis
- Smaller models: Quantize existing, explore 7 & 13 B versions.

```
Decompile: MaliciousFunction - (c373d37b5a9427a18dbf93d519968d9fda04f2a262f424d0611830764c8cc69c)
1
2 /* WARNING: Globals starting with '_' overlap smaller symbols at the same address */
3 /* This function is a malicious function that is designed to perform a variety of malicious
4 activities, including:
5
6 1. Loading and executing a kernel32.dll function
7 2. Enumerating logical drives and creating threads to scan for specific files
8 3. Enumerating network resources and creating threads to scan for specific files
9 4. Enumerating network shares and creating threads to scan for specific files
10 5. Creating a new thread to perform a malicious activity */
11
12 void MaliciousFunction(void)
13
14 {
15     WCHAR WVar1;
16     int *piVar2;
17     bool bVar3;
18     int iVar4;
19     DWORD nBufferLength;
20     LPWSTR lpBuffer;
21     UINT UVar5;
22     int iVar6;
23     LPWSTR lpString1;
24     HANDLE pvVar7;
25     int iVar8;
26     undefined4 extraout_ECX;
27     undefined4 extraout_ECX_00;
28     undefined4 uVar9;
29     undefined4 extraout_EDX;
30     undefined4 extraout_EDX_00;
31     undefined4 uVar10;
32     LPCWSTR lpRootPathName;
33     int iVar11;
34     uint uVar12;
35     int *piVar13;
36     undefined4 uStack_1e0;
37     WCHAR aWStack_1dc [34];
38     undefined local_198 [4];
39     undefined auStack_194 [400];
40
41     memset(local_198,0,400);
42     DAT_0042f894 = LoadAndExecuteKernel32Function(0x16d34cd7,&DAT_0042f0fc,0x14);
43     iVar4 = Ordinal_115(0x202,auStack_194);
44     if (iVar4 != 0) {
45         Ordinal_116();
46     }
47     nBufferLength = GetLogicalDriveStringsW(0,(LPWSTR)0x0);
48     lpBuffer = (LPWSTR)FUN_00415650(nBufferLength * 2);
49     GetLogicalDriveStringsW(nBufferLength,lpBuffer);
50     WVar1 = *lpBuffer;
51     lpRootPathName = lpBuffer;
52     while (WVar1 != L'\0') {
53         bVar3 = false;
54         UVar5 = GetDriveTypeW(lpRootPathName);
55         wsprintfW(aWStack_1dc,L"\\?\\%s\\",lpRootPathName);
56         if (((UVar5 == 2) || (UVar5 == 3)) || (UVar5 == 4)) {
57             if ((DAT_004342e8 != 0) && (iVar4 = 0, 0 < DAT_004342e8)) {
58                 do {
```

Conclusion

Did it work?

- We can fine-tune a model to improve results for reverse engineering tasks
- Model size matters and to get decent results for summary and step-by-step tasks 34B parameter models were required. This may change with better training data, more training steps at higher context limit.
- Matching the pre-training prompts is important, so is finding a decent semi-working starting point.
- Context should be treated like a budget where we maximize the context on all calls. When there is additional space we can add xrefs, dref memory, include dynamic analysis tracing.
- When context is tight due to large functions chunking & summarization helps.
- Analyst in the loop for continuous learning. Make the labeling process an improvement to workflow to drive adoption. Reward contribution.
- Confident hallucinations on Community model (also repeating, not knowing when to stop), susceptible to injection / misdirection.



```
56 FUN_00419450();
57 local_28 = (PVOID)0x417ce5;
58 FUN_00419490(local_8e8,0,0x820);
59 local_28 = (PVOID)0x417cfc;
60 DVar1 = GetModuleFileNameW((HMODULE)0x0,local_1928,0x40f);
61 local_28 = (PVOID)0x0;
62 pvVar2 = (HANDLE)(*DAT_0043b498)();
63 nNumberOfBytesToRead = GetFileSize(pvVar2,(LPDWORD)0x0);
64 lpBuffer = (LPVOID)FUN_00418870(nNumberOfBytesToRead);
65 ReadFile(pvVar2,lpBuffer,nNumberOfBytesToRead,(LPDWORD)&stack0xffffffffe8,(LPOVERLAPPED)0x0);
66 CloseHandle(pvVar2);
67 if (unaff_EDI != 0) {
68     iVar3 = 0;
69     iVar6 = 0;
70     if (0 < (int)DVar1) {
71         do {
72             if (local_1928[iVar3] == L'\\') {
73                 iVar6 = iVar3;
74             }
75             iVar3 = iVar3 + 1;
76         } while (iVar3 < (int)DVar1);
77     }
78     RegOpenKeyExW((HKEY)0x80000001,L"Software\\Microsoft\\Windows\\CurrentVersion\\Run",0,0xf003f
```



•FAILURES

Decompile: `malware_dropper` - (0a1dbcff63619c4d9072484bb17b3d06300504e836e42df98eadf57e7ad0d0be)

```
3 /* This function is a malware dropper that downloads and executes a payload from a remote server.
4    uses the Windows API to download the payload using the `URLDownloadToFileW` function, and then
5    executes it using the `ShellExecuteExW` function. The payload is encrypted using the
6    `CryptDecrypt` function, and the decryption key is stored in the `DAT_0043af64` variable.
7
8    The function first calls the `FUN_00419450` function, which is not shown in the decompiled code.
9    This function is likely responsible for setting up the malware's environment, such as creating
10   new process or thread.
11
12   The function then calls the `GetModuleFileNameW` function to get the path of the current module
13   (i.e., the malware executable). It then calls the `GetFileSize` function to get the size of the
14   module, and uses this size to allocate a buffer for the payload.
15
16   The function then calls the `URLDownloadToFileW` function to download the payload from the remote
17   server. The `URLDownloadToFileW` function takes the URL of the payload as an argument, and writes
18   the downloaded payload to the buffer allocated by the `FUN_00418870` function.
19
20   The function then calls the `CryptDecrypt` function to decrypt the payload using the decryption
21   key stored in `DAT_0043af64`. The decrypted payload is then executed using the `ShellExecuteExW`
22   function.
23
24   The function also uses the `RegOpenKeyExW` and `RegSetValueExW` functions to set a registry value
25   that points to the payload. This registry value is used to persist the malware on the system.
```



```

void FUN_140001000(void)

{
    HRESULT HVar1;
    longlong lVar2;
    char *pcVar3;
    CHAR *pCVar4;
    undefined auStackY_88 [32];
    CHAR local_50 [24];
    CHAR local_38 [24];
    ulonglong local_20;

    local_20 = DAT_140003078 ^ (ulonglong)auStackY_88;
    pcVar3 = s_https__www_google_com_140003000;
    pCVar4 = local_38;
    for (lVar2 = 0x17; lVar2 != 0; lVar2 = lVar2 + -1) {
        *pCVar4 = *pcVar3;
        pcVar3 = pcVar3 + 1;
        pCVar4 = pCVar4 + 1;
    }
    pcVar3 = s_C__mpgoogle_html_140003018;
    pCVar4 = local_50;
    for (lVar2 = 0x11; lVar2 != 0; lVar2 = lVar2 + -1) {
        *pCVar4 = *pcVar3;
        pcVar3 = pcVar3 + 1;
        pCVar4 = pCVar4 + 1;
    }
    HVar1 = URLDownloadToFileA((LPUNKNOWN)0x0,local_38,local_50,0,(LPBINDSTATUSCALLBACK)0x0);
    if (HVar1 == 0) {
        printf(s_File_downloaded_successfully__140003030);
    }
    else {
        printf(s_Failed_to_download_file__140003050);
    }
    FUN_1400011f0(local_20 ^ (ulonglong)auStackY_88);
    return;
}

```



Function Summary:

This function appears to be a malicious payload that is designed to download a file from a remote server and execute it on the target system. The function uses the `URLDownloadToFileA` API to download the file, and if the download is successful, it prints a message to console indicating that the file was downloaded successfully. If the download fails, it prints a message indicating that the file could not be downloaded.

The function also includes a call to another function, `FUN_1400011f0`, which is likely used to clean up any resources or data that were created during the execution of the payload.

New Function Name:

`Payload_DownloadAndExecute`



Demo?

How it's done

- Briefly demonstrate how your plugin and LLM assist in reverse engineering.
- Show a few examples or case studies.

(1) abuse_api_thread32first.exe, checkremotedebugger--5d,

(2) /review

(3) /curate

•FAILURES:

...



Q&A

Any questions?

- <http://lupine.richards.ai:9001>

