









## The Future of Reverse Engineering with Large Language Models

---

Tim Blazytko

 @mr\_phrazer  
 synthesis.to  
 tim@blazytko.to

Moritz Schloegel

 @m\_u00d8  
 mschloegel.me  
 moritz.schloegel@cispa.de

# About Us

- Tim Blazytko
  - Chief Scientist & Head of Engineering, co-founder of emproof
  - designs software protections for embedded devices
  - trainer for (de)obfuscation and reverse engineering techniques



- Moritz Schloegel
  - fresh postdoc at CISPA Helmholtz Center
  - working with bugs by day (mostly fuzzing)
  - code deobfuscation by night





Using LLMs for RE



Local LLMs



Enhancements through Static Analysis

# LLMs in Reverse Engineering

- **hyped** and fast-developing field
- **teasing** powers and limitations for RE
- **not specific** to tools or LLMs

- **hyped** and fast-developing field

- **teasing** powers and limitations for RE

current snapshot, maybe soon outdated

- **not specific** to tools or LLMs

- renaming functions

## Applications to Reverse Engineering

- renaming functions
- renaming variables



## Applications to Reverse Engineering

- renaming functions
- renaming variables
- commenting code

## Applications to Reverse Engineering

- renaming functions
- renaming variables
- commenting code
- explaining code

## Applications to Reverse Engineering

- renaming functions
- renaming variables
- commenting code
- explaining code
- answering questions

## Applications to Reverse Engineering

- renaming functions
- renaming variables
- commenting code
- explaining code
- answering questions
- scripting support

- renaming functions
- renaming variables
- commenting code
- explaining
- answering questions
- scripting support

today: focus on use cases

# Use Case Function Preselection

“For the given decompiler output, analyze the code and suggest a meaningful function name.”

Renamed function at 0x10002b50 to CallWithArguments

Renamed function at 0x10004b50 to DecodeComplexAlgorithm

Renamed function at 0x10008b60 to SetValueToMemoryLocation

Renamed function at 0x10002b70 to CallFunctionPointerWithArguments



Renamed function at 0x10002b50 to CallWithArguments

Renamed function at 0x10004b50 to DecodeComplexAlgorithm

Renamed function at 0x10006b50 to DecodeComplexAlgorithm

Renamed function at 0x10008b50 to DecodeComplexAlgorithm

often too generic

Renamed function at 0x40cbb5 to `DecompileCodeAnalyze`  
Renamed function at 0x4033c2 to `DecompileCodeAnalyze`  
Renamed function at 0x4024c2 to `DecompileCodeAnalyze`  
Renamed function at 0x402d58 to `DecompileCodeAnalyze`  
Renamed function at 0x40ed62 to `DecompileAndProtectMemoryPage`  
Renamed function at 0x409e53 to `DecompileAndFindMatchingStringInMemory`  
Renamed function at 0x40e4dd to `DecompileAndAnalyzeFunction`

Renamed function at 0x40cbb5 to `DecompileCodeAnalyze`

Renamed function at 0x4033c2 to `DecompileCodeAnalyze`

Renamed function at 0x4024c2 to `DecompileCodeAnalyze`

Renamed funct

sometimes entirely useless

Renamed funct

Renamed function at 0x409e53 to `DecompileAndFindMatchingStringInMemory`

Renamed function at 0x40e4dd to `DecompileAndAnalyzeFunction`

Renamed function at 0x1000cf10 to InitializeKeyLoggerAndHandleErrors  
Renamed function at 0x10014ae0 to CreateNamedPipesAndRunShellCommands  
Renamed function at 0x100186d0 to SearchForAProcessByName  
Renamed function at 0x1001d880 to SendHTTPPOSTRequestAndHandleResponse

Renamed function at 0x1000cf10 to InitializeKeyLoggerAndHandleErrors

Renamed function at 0x10014ae0 to CreateNamedPipesAndRunShellCommands

Renamed function at can be very helpfulame

Renamed function at ndHandleResponse

Use Case

Identification of Library Functions

```
000007ac void* copy_backward_string(void* arg1, void* arg2, int32_t arg3)
```

```
000007ac int32_t i = arg3  
000007b0 void* r4 = arg2 - 1  
000007b4 void* r3 = arg1 - 1  
000007c0 do {  
000007b8     char r5 = *(r4 + 1)  
000007b8     r4 = r4 + 1  
000007bc     *(r3 + 1) = r5  
000007bc     r3 = r3 + 1  
000007c0     i = i - 1  
000007c0 } while (i != 0)  
000007c4 return r3
```

Log

Search log

[Default] Renaming sub\_7ac to copy\_backward\_string

[Default] Renaming sub\_820 to return\_address\_of\_data\_838

[Default] Renaming sub\_3e4 to process\_input\_and\_copy\_string

[Default] Renaming sub\_77c to save\_and\_clear\_fpu\_registers

```
000007ac void* copy_backward_string(void* arg1, void* arg2, int32_t arg3)
```

```
000007ac int32_t i = arg3  
000007b0 void* r4 = arg2 - 1  
000007b4 void* r3 = arg1 - 1  
000007c0 do {  
000007b8     char r5 = *(r4 + 1)  
000007b8     r4 = r4 + 1  
000007bc     *(r3 + 1) = r5
```

good approximation, but incorrect

```
000007c4     return r3
```

Log

```
[Default] Renaming sub_7ac to copy_backward_string  
[Default] Renaming sub_820 to return_address_of_data_838  
[Default] Renaming sub_3e4 to process_input_and_copy_string  
[Default] Renaming sub_77c to save_and_clear_fpu_registers
```



# Enhancing Decompiler Output

“Enhance the following decompiler output by suggesting more meaningful variable names. Also, add comments.”

c



```
000007ac void* copy_memory(void* destination, void* source, int32_t count)

000007ac     int32_t remaining_bytes = count;           // Initialize remaining bytes to be copied
000007b0     void* source_ptr = source - 1;           // Source pointer starts before the actual source
000007b4     void* dest_ptr = destination - 1;        // Destination pointer starts before the actual
destination
000007c0     do {
000007b8         char byte_to_copy = *(source_ptr + 1); // Fetch the next byte from the source
000007b8         source_ptr = source_ptr + 1;           // Move the source pointer to the next byte
000007bc         *(dest_ptr + 1) = byte_to_copy;       // Write the byte to the destination
000007bc         dest_ptr = dest_ptr + 1;               // Move the destination pointer to the next byte
000007c0         remaining_bytes = remaining_bytes - 1; // Decrement the remaining byte count
000007c0     } while (remaining_bytes != 0);           // Repeat until all bytes are copied
000007c4     return dest_ptr;                          // Return the pointer to the last copied byte in
the destination
```

c

 Copy

```
000007ac void* copy_memory(void* destination, void* source, int32_t count)
000007ac     int32_t remaining_bytes = count;           // Initialize remaining bytes to be copied
000007b0     void* source_ptr = source - 1;           // Source pointer starts before the actual source
000007b4     void* dest_ptr = destination - 1;         // Destination pointer starts before the actual
destination
000007c0     d
000007b8     source_ptr = source_ptr + 1;               // Move the source pointer to the next byte
000007b8     *(dest_ptr + 1) = byte_to_copy;           // Write the byte to the destination
000007bc     dest_ptr = dest_ptr + 1;                   // Move the destination pointer to the next byte
000007c0     remaining_bytes = remaining_bytes - 1;   // Decrement the remaining byte count
000007c0     } while (remaining_bytes != 0);           // Repeat until all bytes are copied
000007c4     return dest_ptr;                           // Return the pointer to the last copied byte in
the destination
```

works well for common code

```
// Function to perform a transformation on input string based on some algorithm
char transform_string(char* input_string, int32_t param, int32_t* lookup_table) {
    char* source = input_string;
    char* index_ptr = nullptr; // Pointer used to iterate over the input string
    int32_t intermediate_values[1024] = {0}; // Buffer to store intermediate values
    void* buffer; // Temporary buffer to use during memset
    __builtin_memset(&buffer, 0, 0x3fc); // Initialize buffer with zeros
    int32_t accumulator = 0; // Accumulator to store intermediate sums
    int32_t* intermediate_ptr = intermediate_values;

    do {
        uint32_t temp_val = lookup_func(0:index_ptr, lookup_table); // Get value from lookup table
        *(index_ptr + source) = index_ptr->byte_val; // Assign byte value to source
        index_ptr = &index_ptr[1]; // Increment index pointer
        *intermediate_ptr = zero_extend(*(temp_val + param)); // Extend and store value in intermediate
array
        intermediate_ptr = &intermediate_ptr[1]; // Increment intermediate array pointer
    } while (index_ptr < 0x100); // Loop until index_ptr reaches 0x100
}
```

```
// Function to perform a transformation on input string based on some algorithm
char transform_string(char* input_string, int32_t param, int32_t* lookup_table) {
    char* source = input_string;
    char* index_ptr = nullptr; // Pointer used to iterate over the input string
    int32_t intermediate_values[1024] = {0}; // Buffer to store intermediate values
    void* buffer; // Temporary buffer to use during memset
    __builtin_memset(&buffer, 0, 0x3fc); // Initialize buffer with zeros
    int32_t accumulator = 0;
    int32_t* intermediate_ptr

    do {
        uint32_t temp_val = lookup_func(0:index_ptr, lookup_table); // Get value from lookup table
        *(index_ptr + source) = index_ptr->byte_val; // Assign byte value to source
        index_ptr = &index_ptr[1]; // Increment index pointer
        *intermediate_ptr = zero_extend(*(temp_val + param)); // Extend and store value in intermediate
array
        intermediate_ptr = &intermediate_ptr[1]; // Increment intermediate array pointer
    } while (index_ptr < 0x100); // Loop until index_ptr reaches 0x100
}
```

not always helpful

# Use Case Analysis Guidance for Unfamiliar Instruction Sets

“Explain me the following assembly snippet.”



```
; Load Immediate Shifted:
```

```
; Load the immediate value 0xc00 into the upper 16 bits of register r3
```

```
0000076c 3c600c00 lis r3, 0xc00 {0xc0000000}
```

```
; Move to Special Purpose Register:
```

```
; Move the contents of register r3 to special-purpose register 0x230
```

```
00000770 7c708ba6 mtspr 0x230, r3
```

```
; Instruction Synchronize:
```

```
; Synchronize the instruction stream, ensuring all previous instructions  
are completed before continuing
```

```
00000774 4c00012c isync
```

```
; Load Immediate Shifted:  
; Load the immediate value 0xc00 into the upper 16 bits of register r3  
0000076c 3c600c00 lis r3, 0xc00 {0xc0000000}
```

```
; Move to Special Purpose Register:  
; Move the contents of register r3 to special-purpose register 0x230  
00000770
```

works well for assembly snippets

```
; Instruction Synchronize:  
; Synchronize the instruction stream, ensuring all previous instructions  
  are completed before continuing  
00000774 4c00012c isync
```

# Use Case Identification of Algorithms

“Analyze the following code and explain what it does.”

This function performs some kind of **data transformation** or **encryption/decryption** by manipulating bytes and **using loops to iterate through memory**.

This function performs some kind of data transformation or encryption/decryption through memory loops to iterate

often generic or wrong

Use Case  
Code Logic Explanation

“The following code implements a crypto algorithm. Which one and why?”



The code snippet likely implements the **RC4 cryptographic algorithm** due to its characteristic **key scheduling** and pseudo-random generation processes.

The code snippet likely implements the **RC4 cryptographic algorithm** due to its characteristic key **context often helps** generation processes.

# Use Case Verification Oracle

“Does the following code implement XXX? Answer YES or NO.”

“Does the following code implement VVV? Answer YES or NO”

good to filter false positives from heuristics

# Use Case Script Development

“The following code implements a string decryption routine.  
Rebuilt it in Python.”

```
def decrypt(input_string):  
    xor_key = 0x22  
    decrypted_chars = [chr(ord(char) ^ xor_key) for char in input_string]  
    return ''.join(decrypted_chars)
```



```
def decrypt(input_string):  
    xor_key = 0x22  
    decrypted_chars = [chr(ord(char) ^ xor_key) for char in input_string]  
    return ''.join(decrypted_chars)
```

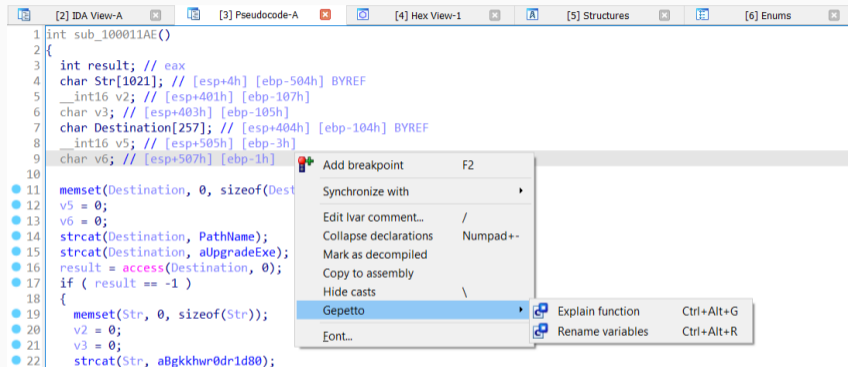
works fine for small tasks

## Interim Conclusion

- good for **approximations** and **high-level** understanding
- answers sometimes **generic** or **wrong**
- **adding context** often helps
- **no** real (code) **understanding**

Tools

- various tools and wrappers for **commercial LLMs**
- **IDA Pro**: Gepetto, Copilot for IDA Pro
- **Ghidra**: GhidraChatGPT, GptHidra
- **Binary Ninja**: Sidekick, BinaryNinja-OpenAI



The screenshot shows the IDA Pro interface with a pseudocode window. The code is as follows:

```
1 int sub_100011AE()  
2 {  
3     int result; // eax  
4     char Str[1021]; // [esp+4h] [ebp-504h] BYREF  
5     __int16 v2; // [esp+401h] [ebp-107h]  
6     char v3; // [esp+403h] [ebp-105h]  
7     char Destination[257]; // [esp+404h] [ebp-104h] BYREF  
8     __int16 v5; // [esp+505h] [ebp-3h]  
9     char v6; // [esp+507h] [ebp-1h]  
10  
11     memset(Destination, 0, sizeof(Destination));  
12     v5 = 0;  
13     v6 = 0;  
14     strcat(Destination, PathName);  
15     strcat(Destination, aUpgradeExe);  
16     result = access(Destination, 0);  
17     if ( result == -1 )  
18     {  
19         memset(Str, 0, sizeof(Str));  
20         v2 = 0;  
21         v3 = 0;  
22         strcat(Str, aBggkhr0dr1d80);
```

A context menu is open over the code, with the 'Gepetto' option selected. The menu items are:

- Add breakpoint F2
- Synchronize with
- Edit lvar comment... /
- Collapse declarations Numpad+-
- Mark as decompiled
- Copy to assembly
- Hide casts \
- Gepetto** (selected)
- Font...
- Explain function Ctrl+Alt+G
- Rename variables Ctrl+Alt+R

<https://github.com/JusticeRage/Gepetto>

# Binary Ninja Sidekick

The screenshot displays the Binary Ninja Sidekick interface for analyzing a string decryption routine. The main window shows assembly code for a function named `sub_18778`. The code includes instructions for loading pointers, calculating offsets, and performing XOR operations on input strings `arg1` and `arg2` using a key derived from `data_1ec28`. Comments in the assembly describe the XOR process and the use of `data_1ec28` as a key.

On the left, the Assistant pane provides a summary of the routine and a Python function that replicates the decryption logic. The Python function `decrypt_string` takes an input string and XORs each byte with the corresponding byte from `data_1ec28`.

Below the Assistant pane, the Suggestions pane lists various pointers and lengths used in the routine, such as `arg1`, `rbp`, `arg2`, and `data_1ec28`.

At the bottom, the Strings pane shows a list of strings found in the memory, including `RD00`, `CFDKL`, `00ACFK0K`, `CFDKLQVPCWVP`, `OSKLOD`, `OSFTKAG`, `QMSFTKQMP`, `EWZCV`, `CFDKL`, `CFDKLQVPCWVP`, `NRLV`, `HTWXP`, `CLIM`, `XKZZ`, `WwJM`, `TKXZT`, `WwJM`, `QVQD`, `KDIP`, and `FRZCOZM`.

Downsides



Internet connection required





Every query costs \$\$\$



Privacy risks



## Local LLMs

### Pros:

- offline
- privacy-sensitive

### Pros:

- offline
- privacy-sensitive

### Cons:

- slower

### Pros:

- offline
- privacy-sensitive

### Cons:

- slower
- less powerful

### Pros:

- offline
- privacy-sensitive

### Cons:

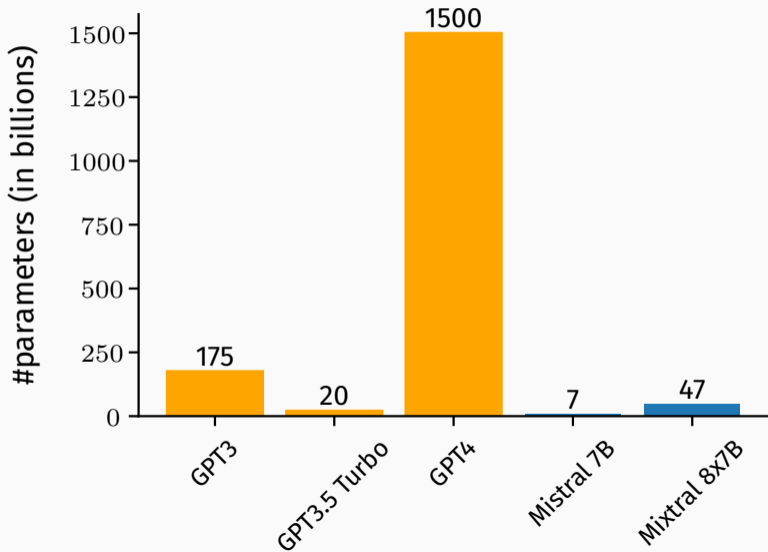
- slower
- less powerful
- computation resources

Anecdotally, for some function renaming queries:

- GPT query: <2s
- Mistral 7B on M1 Macbook Pro: 5-8s
- Mistral 7B on M3 Macbook Air: 10s



## Local LLMs are .. less powerful



### Good GPU:

- NVIDIA GTX 3090
- NVIDIA GTX 4090

### ARM-based Mac:

- M1/M2/M3 Macbook
- or workstation

Good news:  
We can already use local LLMs for RE

## (Unfair) Comparison to GPT4

Use Case	Mistral 7B	Mixtral 8x7B
function renaming	✓	✓
identify library functions	✓	✓
enhance decompiler output	✗	✓
annotate assembly	✗	✓
explain code logic	✗	✓
helper script development	✗	✓
verification oracle	✗	✓

Can we do better?

Context helps..

..so let's use available information!

⇒ incorporate insights from static analysis



example: function renaming

💡 rename only “relevant” functions



improves accuracy and speed

# Context-sensitive Annotations

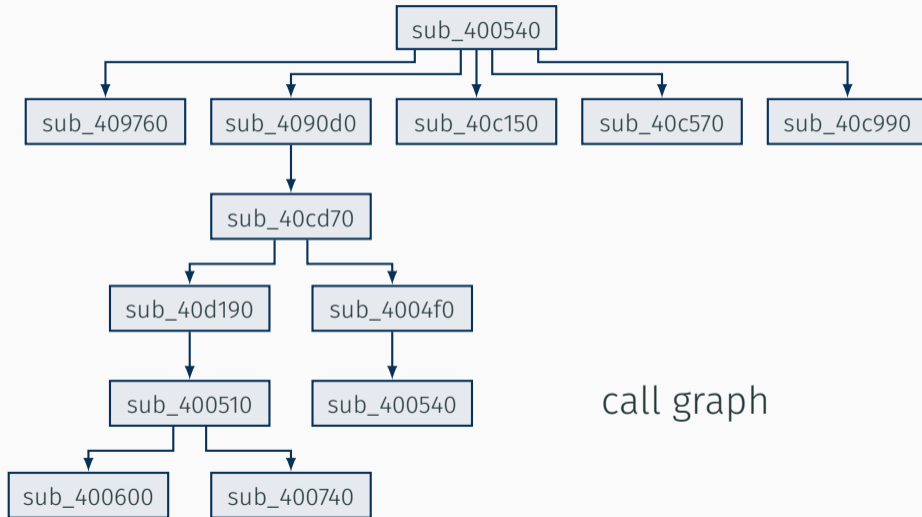
```
1000ead0 enum WIN32_ERROR __fastcall configure_proxy_settings(void* arg1)
1000ead0     int32_t var_4_2 = 0xffffffff
1000ead2     int32_t (* var_8)() = sub_1003bf4b
1000eadd     int32_t* fsbase
1000eadd     int32_t var_c = *fsbase
1000eade     *fsbase = &var_c
1000eaee     int32_t var_114 = 0
1000eb08     int32_t var_4 = 0
1000eb1a     if (sub_10005d00(&var_114, 0x80000001, "Software\Microsoft\Windows\Curre... ", 1, 0) != 0)
1000eb2f         int32_t var_110
1000eb2f         sub_10005db0(&var_114, "ProxyEnable", &var_110)
1000eb3a         if (var_110 != 0)
1000eb4a             sub_10005db0(&var_114, "ProxyServer", &var_110)
1000eb59             char* eax_3 = sub_1002e3d0(&var_110, "http=")
1000eb65             void* esi_2
1000eb65             if (eax_3 != 0)
1000eb6d                 esi_2 = &eax_3[5]
```

# Context-sensitive Annotations

```
1000ead0 enum WIN32_ERROR __fastcall configure_proxy_settings(void* arg1)
1000ead0 int32_t var_4_2 = 0xffffffff
1000ead2 int32_t (* var_8)() = sub_1003bf4b
1000eadd int32_t* fsbase
1000eadd int32_t var_c = *fsbase
1000eade *fsbas
1000eaee int32_
1000eb08 int32_
1000eb1a if (sub_10005d00(&var_114, 0x80000001, "Software\Microsoft\Windows\Curre... ", 1, 0) != 0)
1000eb2f int32_t var_110
1000eb2f sub_10005db0(&var_114, "ProxyEnable", &var_110)
1000eb3a if (var_110 != 0)
1000eb4a sub_10005db0(&var_114, "ProxyServer", &var_110)
1000eb59 char* eax_3 = sub_1002e3d0(&var_110, "http=")
1000eb65 void* esi_2
1000eb65 if (eax_3 != 0)
1000eb6d esi_2 = &eax_3[5]
```

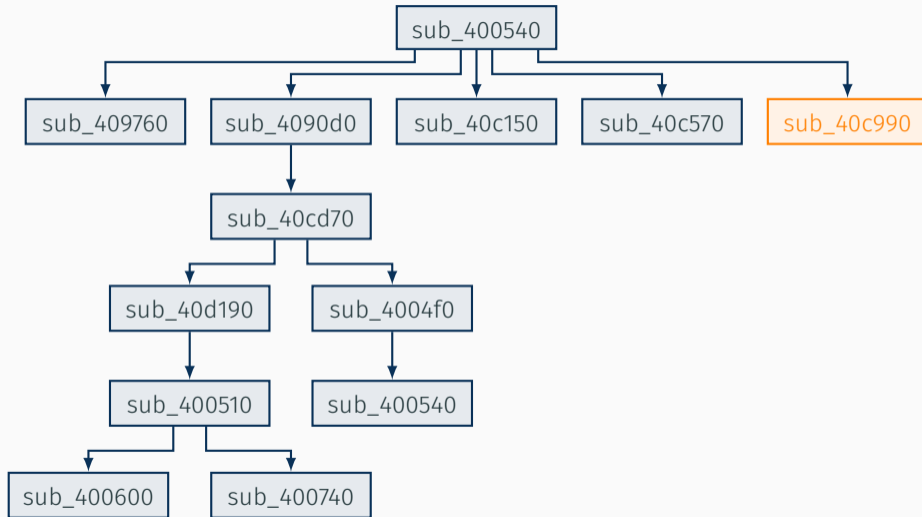
strings and API functions

# Context-sensitive Annotations

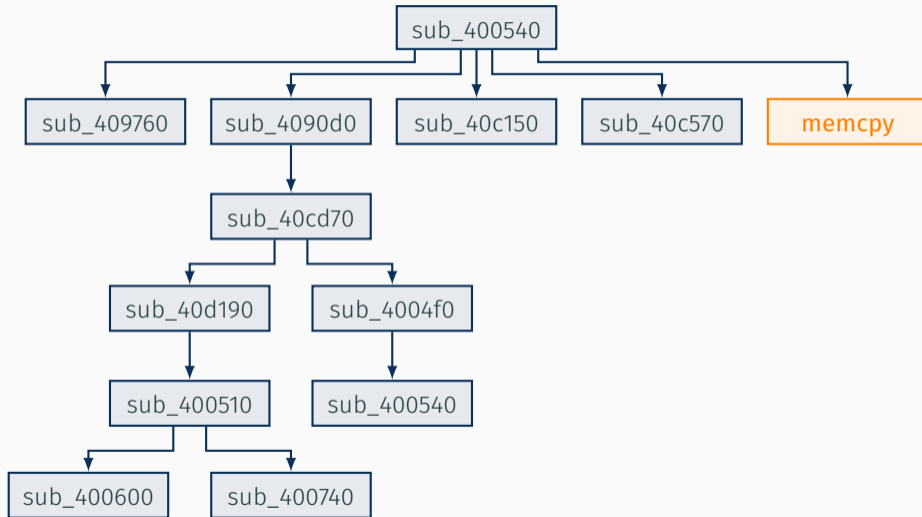


call graph

# Context-sensitive Annotations

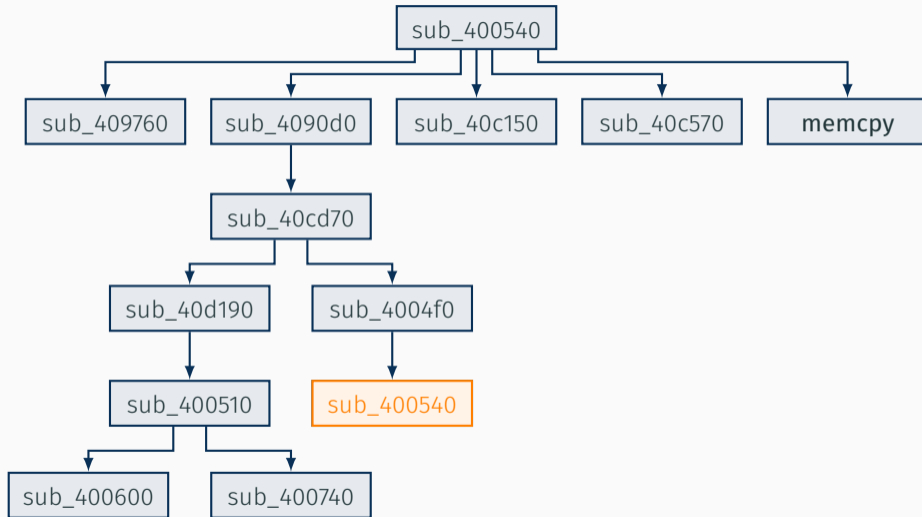


# Context-sensitive Annotations

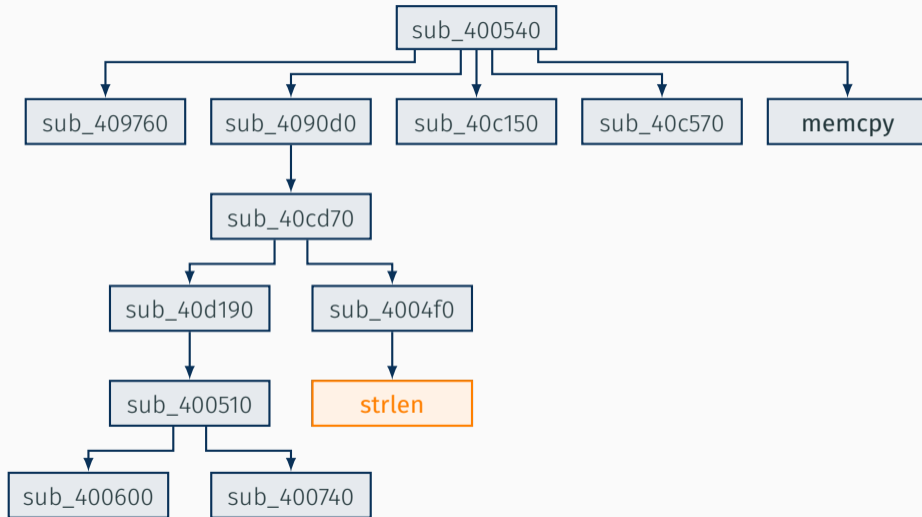




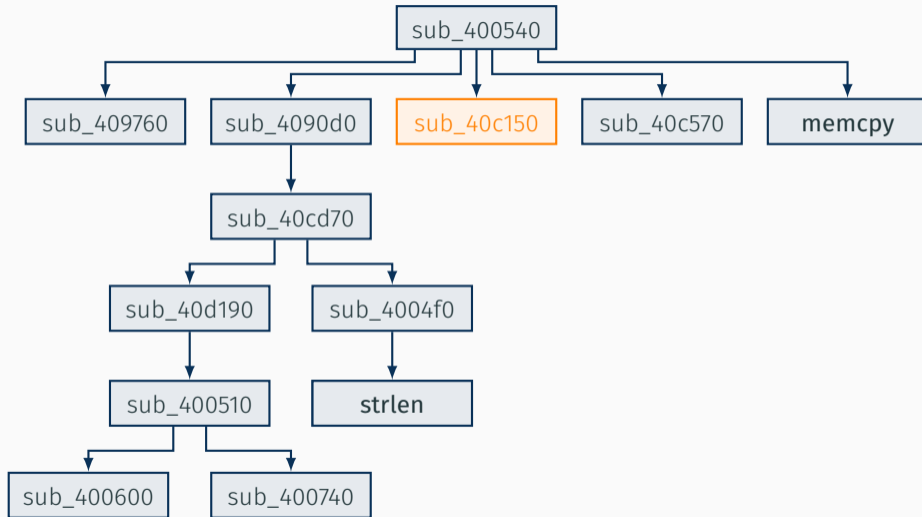
# Context-sensitive Annotations



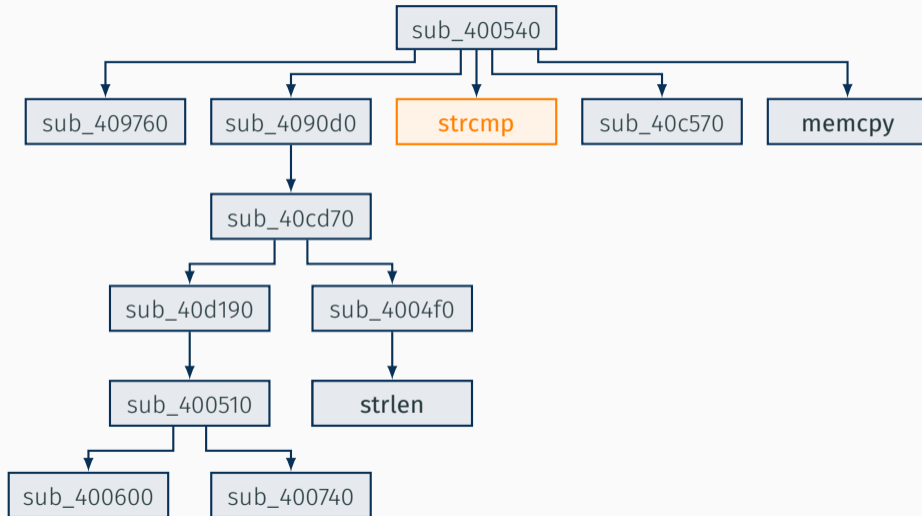
# Context-sensitive Annotations



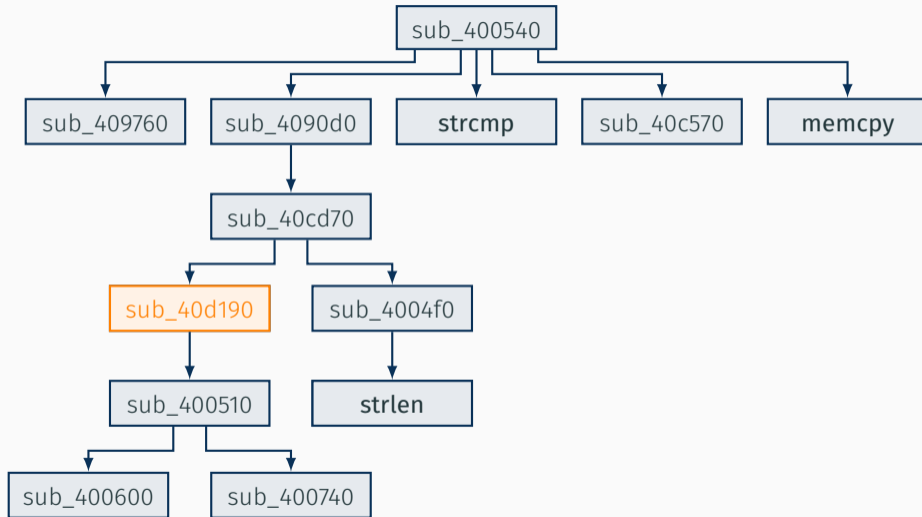
# Context-sensitive Annotations



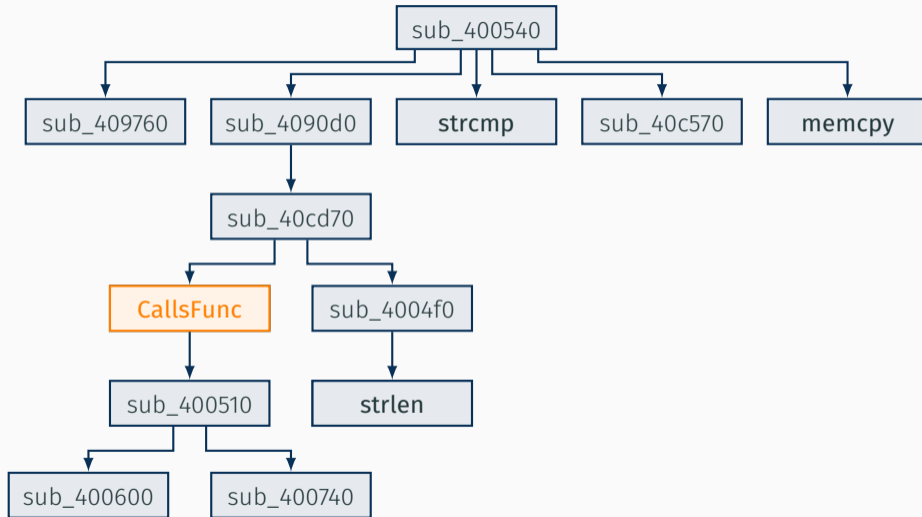
# Context-sensitive Annotations



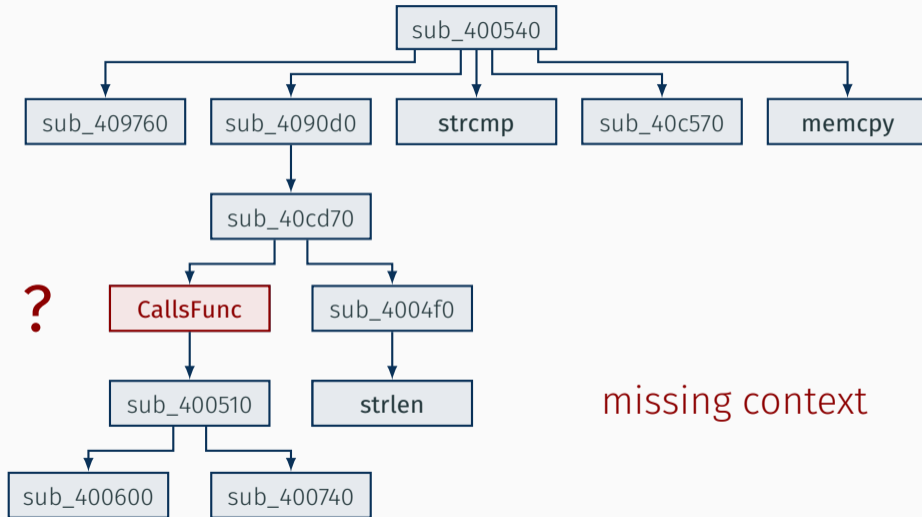
# Context-sensitive Annotations



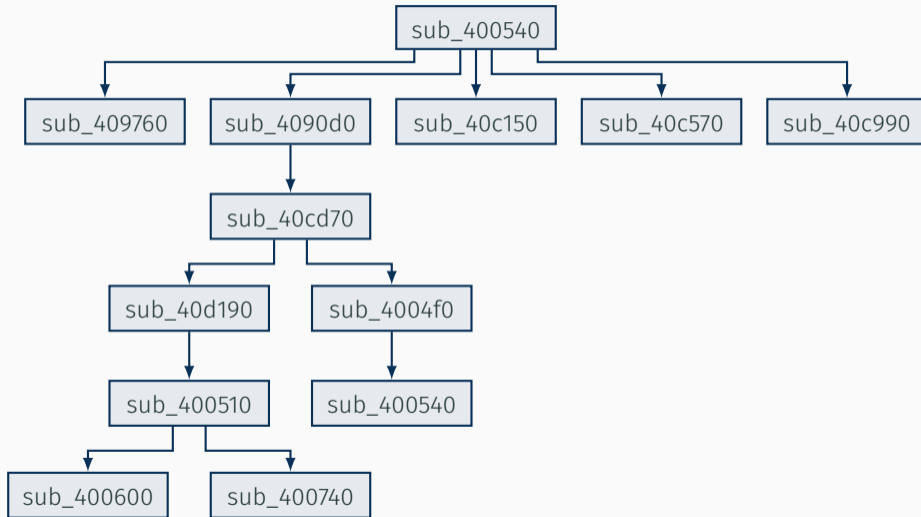
# Context-sensitive Annotations



# Context-sensitive Annotations

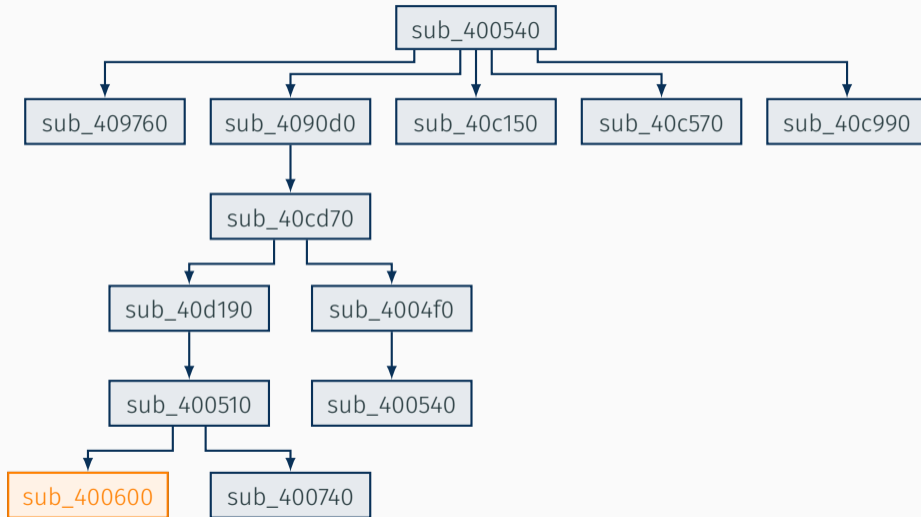


# Context-sensitive Annotations: Bottom-up Propagation

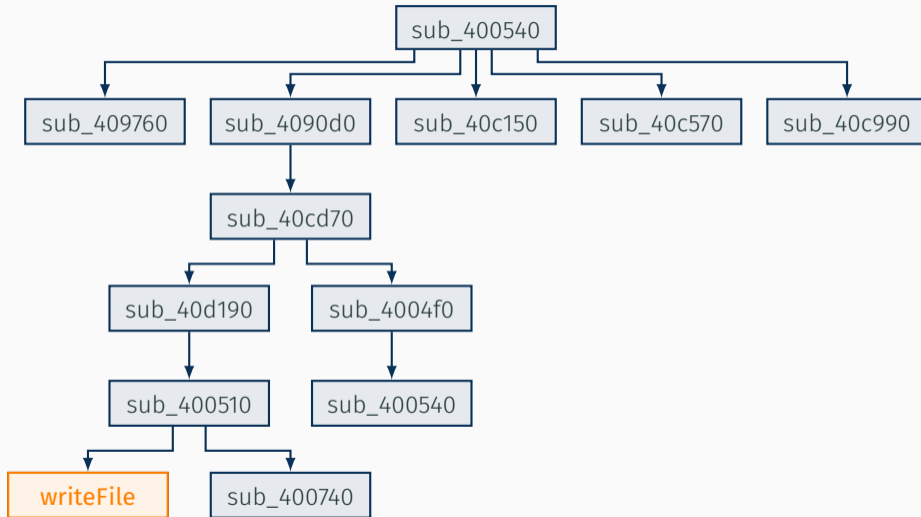




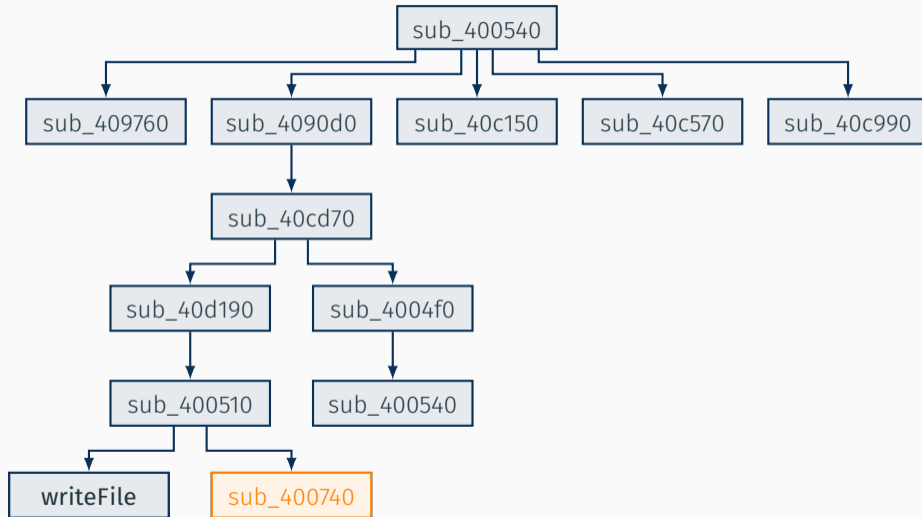
# Context-sensitive Annotations: Bottom-up Propagation



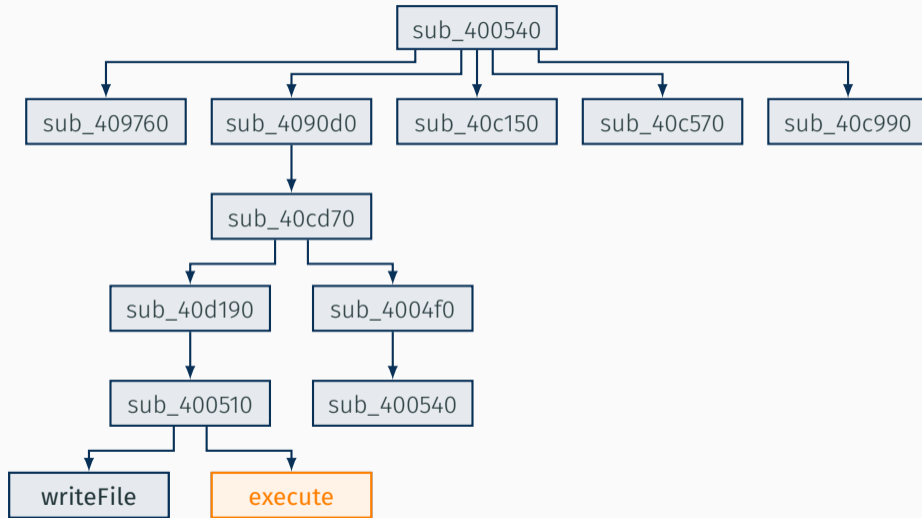
# Context-sensitive Annotations: Bottom-up Propagation



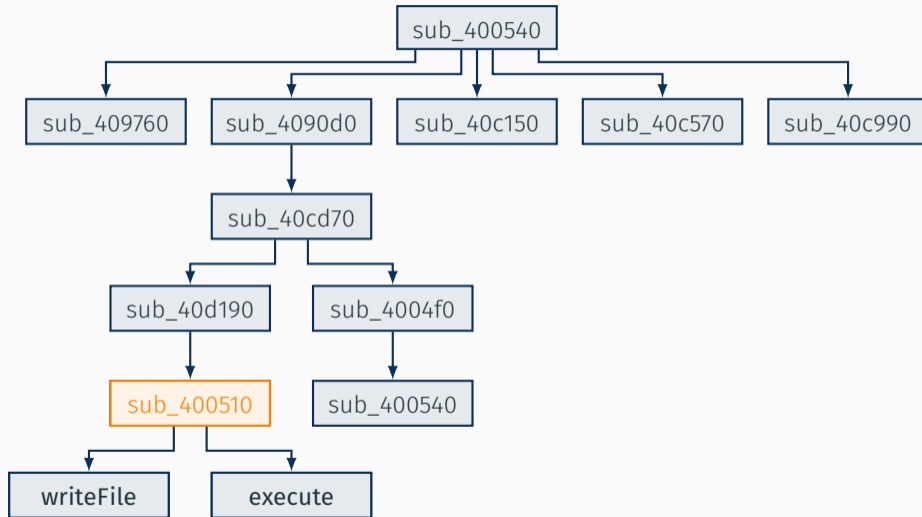
# Context-sensitive Annotations: Bottom-up Propagation



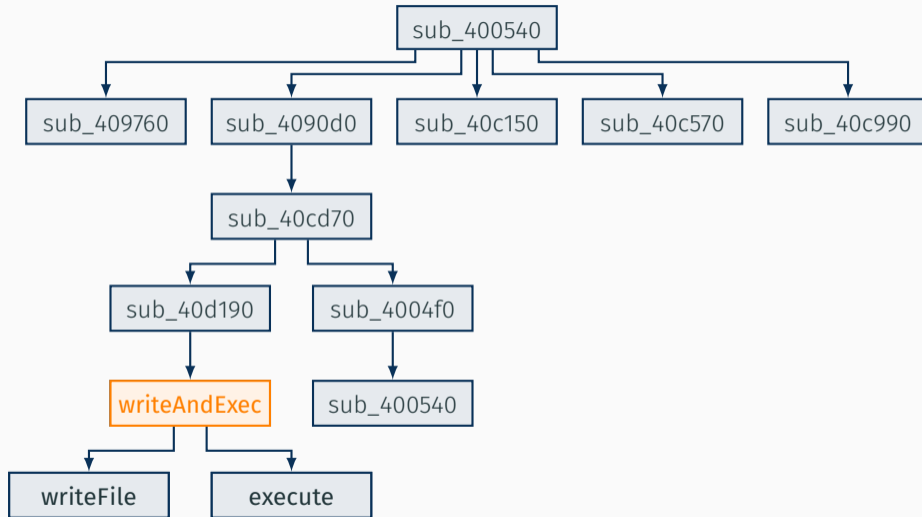
# Context-sensitive Annotations: Bottom-up Propagation



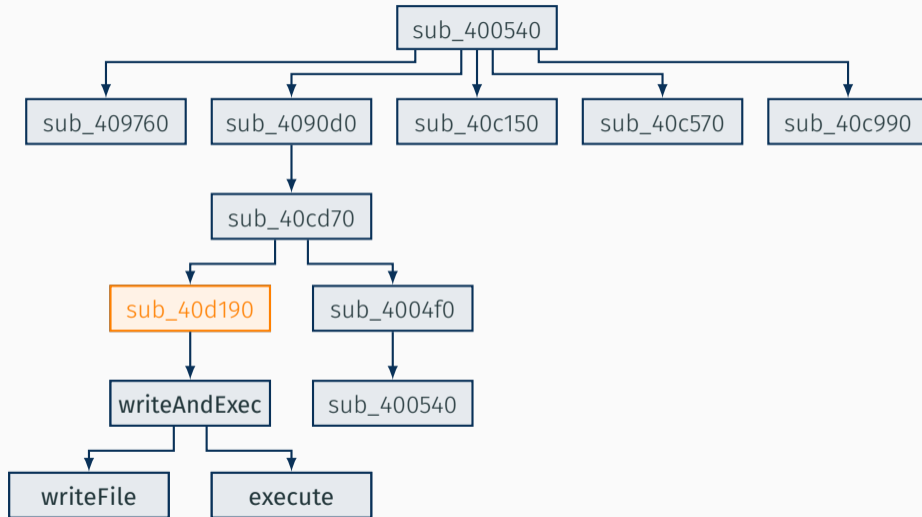
# Context-sensitive Annotations: Bottom-up Propagation



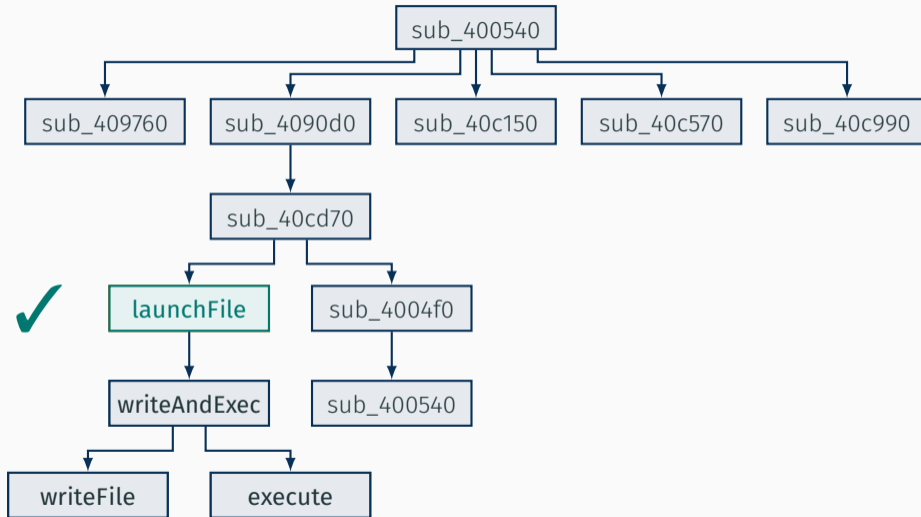
# Context-sensitive Annotations: Bottom-up Propagation



# Context-sensitive Annotations: Bottom-up Propagation

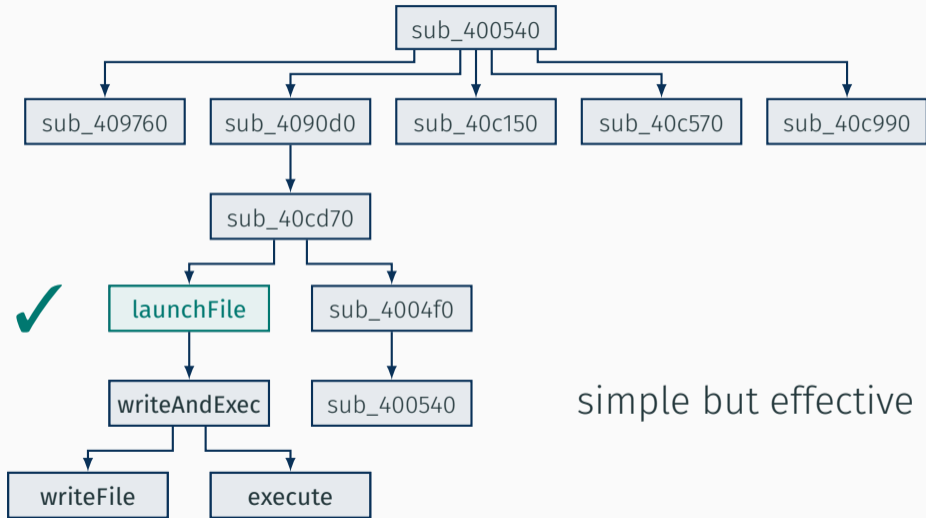


# Context-sensitive Annotations: Bottom-up Propagation





# Context-sensitive Annotations: Bottom-up Propagation



simple but effective

# Tools

📖 README 📄 GPL-2.0 license



## ReverserAI (v1.1)

---

Author: **Tim Blazytko**

*Provides automated reverse engineering assistance through the use of local large language models (LLMs) on consumer hardware.*

### Description:

---

*ReverserAI* is a research project designed to automate and enhance reverse engineering tasks through the use of locally-hosted large language models (LLMs). Operating entirely offline, this initial release features the automatic suggestion of high-level, semantically meaningful function names derived from decompiler output. *ReverserAI* is provided as a Binary Ninja plugin; however, its architecture is designed to be extended to other reverse engineering platforms such as IDA and Ghidra.

[https://github.com/mrphrazer/reverser\\_ai](https://github.com/mrphrazer/reverser_ai)

- Binja plugin to include local LLMs
- more playground than finished product
- supports two models (Mistral 7B and Mixtral 8x7B)

Can we do better?

## Things to improve

- better queries (prompt engineering)
- better models: <https://github.com/eugeneyan/open-llms>
- fine-tuned models

## Things to improve

- better queries (prompt engineering)
- better models: <https://github.com/eugeneyan/open-llms>
- fine-tuned models

The screenshot shows the GitHub repository page for 'aidapal'. At the top, the repository name 'aidapal' is displayed with a 'Public' badge. To the right, there are buttons for 'Watch 7', 'Fork 9', and 'Star 94'. Below this, the main branch is 'main', with '1 Branch' and '0 Tags' indicated. A search bar for files and an 'Add file' button are present. A commit history table shows the following entries:

Commit Message	Author	Time
Update idapal_qt_interface.py	AverageBusinessUser	yesterday
Update README.md		3 weeks ago
initial plugin upload		3 weeks ago
Update idapal_qt_interface.py		yesterday

Below the commit history, the 'README' file is selected. The README content shows the title 'aiDAPal'. On the right side of the repository page, the 'About' section states: 'aiDAPal is an IDA Pro plugin that uses a locally running LLM that has been fine-tuned for Hex-Rays pseudocode to assist with code analysis.' Below this, there are links for 'Readme', 'Activity', 'Custom properties', and statistics: '94 stars', '7 watching', and '9 forks'. A 'Report repository' link is also present.

<https://github.com/atredispartners/aidapal>



- plugin for IDA with focus on enhancing decompiler output
- fine-tuned Mistral 7B  $\Rightarrow$  can keep up with GPT4
- also uses available context information

- plugin for IDA with focus on enhancing decompiler output
- fine-tuned fine-tuning works well<sup>PT4</sup>
- also uses available context information

## Currently Impossible

- *real* code analysis
- bug finding (beyond easy patterns)
- cross-function analysis

- *real* code analysis

- bug finding (beyond easy patterns)

LLMs as helper, not automated analysts

- cross-function analysis

# Future Trends

- **enhanced scalability** for broader inputs
- advancements in **on-device** LLMs

- semantic code search
- identification of noteworthy code segments
- patch recommendation systems
- binary similarity and clustering

Conclusion



## Takeaways

1. LLMs are good for **approximations** and **high-level** understanding
2. But: They can be wrong and have **no** real (code) **understanding**
3. **Adding context** often helps increase accuracy
4. Local LLMs are somewhat worse but ensure **privacy**

- (local) LLMs help, check them out
- but they are no panacea, be wary of the hype

Tim Blazytko

 @mr\_phrazer

 [synthesis.to](https://synthesis.to)

Moritz Schloegel

 @m\_u00d8

 [mschloegel.me](https://mschloegel.me)