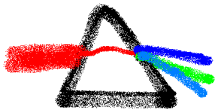


THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Singapore
ICFP / SPLASH 2025

ClearAgent: Agentic Binary Analysis for Effective Vulnerability Detection

Xiang Chen, Anshunkang Zhou, Chengfeng Ye, Charles Zhang

The Hong Kong University of Science and Technology

2025-10-15 @ LMPL '25

Contents

1. Background

Binary Code Analysis

- Analyzable binary code formats: Assembly → IR → Source code

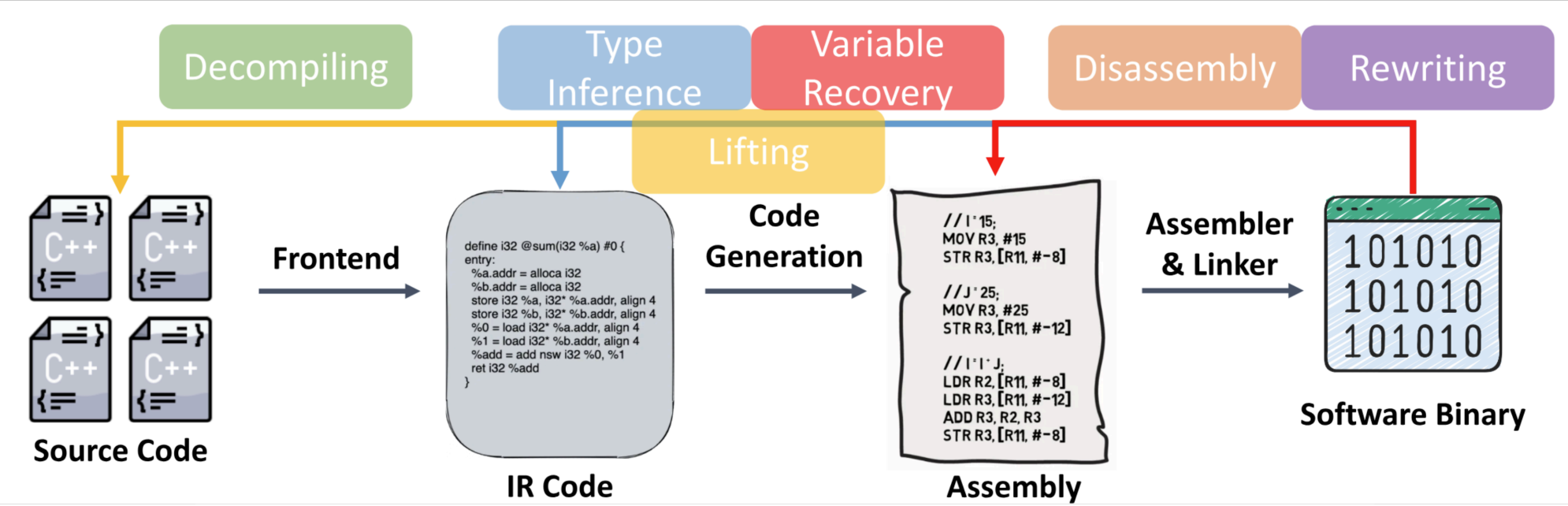


Figure 1: ⟶ Compilation ⟵ Reverse Engineering

[1] Commercial-Off-The-Shelf, the source code or build environment is not available

Binary Code Analysis

- Analyzable binary code formats: Assembly → IR → Source code

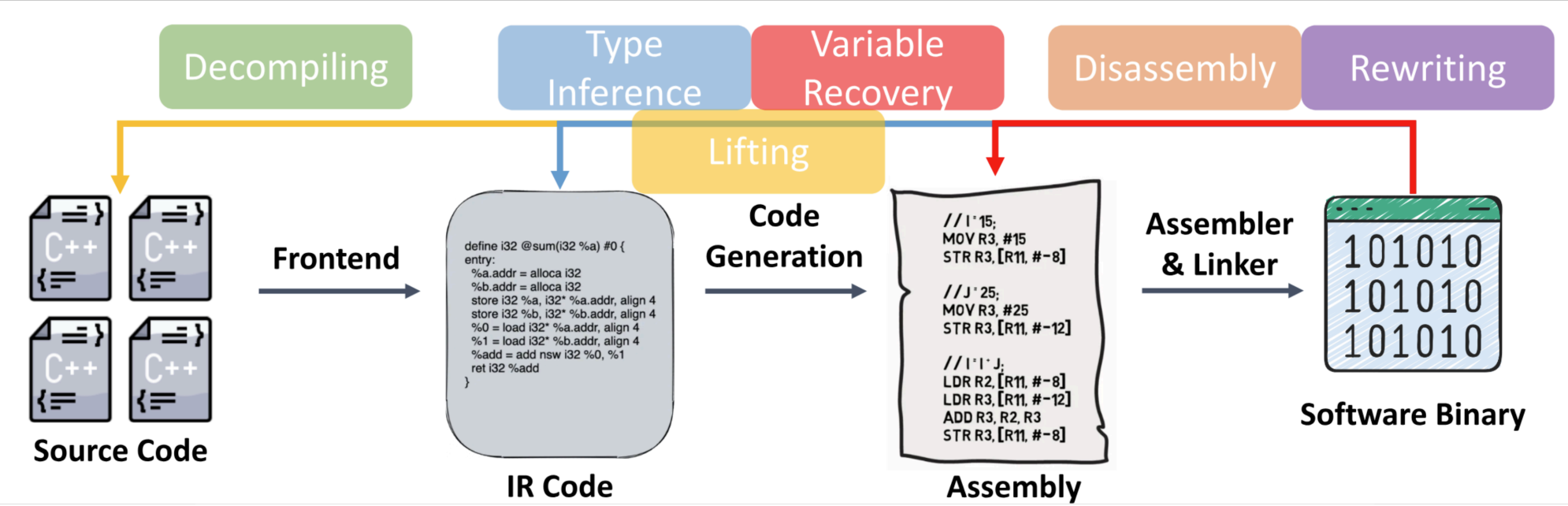


Figure 1: ⟶ Compilation ⟵ Reverse Engineering

- Binary code analysis is crucial for detecting vulnerabilities in COTS[1] software.

[1] Commercial-Off-The-Shelf, the source code or build environment is not available

Challenges

- Traditional Challenges:
  - **Undecidability**: symbol and type information stripped / indirect-call, etc.
  - **Expertise**: ad-hoc human integration and modeling in specific analysis tasks

[1] ReSym: Harnessing LLMs to Recover Variable and Data Structure Symbols from Stripped Binaries, CCS '24

Challenges

- Traditional Challenges:
  - **Undecidability**: symbol and type information stripped / indirect-call, etc.
  - **Expertise**: ad-hoc human integration and modeling in specific analysis tasks
- New challenges in the era of LLM:
  - **Fragmentation**: specialized LLMs are designed for sub-tasks (e.g., variable recovery[1])
    - Hard to merge their knowledge for a comprehensive binary understanding
    - Rely on distinct prompt engineering for effective usage

[1] ReSym: Harnessing LLMs to Recover Variable and Data Structure Symbols from Stripped Binaries, CCS '24

Challenges

- Traditional Challenges:
  - **Undecidability**: symbol and type information stripped / indirect-call, etc.
  - **Expertise**: ad-hoc human integration and modeling in specific analysis tasks
- New challenges in the era of LLM:
  - **Fragmentation**: specialized LLMs are designed for sub-tasks (e.g., variable recovery[1])
    - Hard to merge their knowledge for a comprehensive binary understanding
    - Rely on distinct prompt engineering for effective usage
  - **Hallucination**: general-purpose LLMs struggle to understand large-scale binary code

[1] ReSym: Harnessing LLMs to Recover Variable and Data Structure Symbols from Stripped Binaries, CCS '24

Challenges

- Traditional Challenges:
  - **Undecidability**: symbol and type information stripped / indirect-call, etc.
  - **Expertise**: ad-hoc human integration and modeling in specific analysis tasks
- New challenges in the era of LLM:
  - **Fragmentation**: specialized LLMs are designed for sub-tasks (e.g., variable recovery[1])
    - Hard to merge their knowledge for a comprehensive binary understanding
    - Rely on distinct prompt engineering for effective usage
  - **Hallucination**: general-purpose LLMs struggle to understand large-scale binary code
- 🎯 Our direction: let LLM interact with mature binary analysis tools in a guided manner

[1] ReSym: Harnessing LLMs to Recover Variable and Data Structure Symbols from Stripped Binaries, CCS '24

LLM Agent

- Agent[1]: anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.

[1] Artificial Intelligence: A Modern Approach, 1995

LLM Agent

- Agent[1]: anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.
- LLM Agent workflow:

```python
env = Environment()
tools = Tools(env)
system_prompt = "Goals, constraints, and how to act"
while True:
    action = llm.run(system_prompt + env.state) # agentic 1
    env.state = tools.run(action)               # agentic 2
```

[1] Artificial Intelligence: A Modern Approach, 1995

LLM Agent

- Agent[1]: anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.
- LLM Agent workflow:

```python
env = Environment()
tools = Tools(env)
system_prompt = "Goals, constraints, and how to act"
while True:
    action = llm.run(system_prompt + env.state) # agentic 1
    env.state = tools.run(action)               # agentic 2
```

- ⛳ Agentic:
  1. more informative tool usage
  2. more complex environment, less supervision

[1] Artificial Intelligence: A Modern Approach, 1995

LLM Agent

- Agent[1]: anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.
- LLM Agent workflow:

```python
env = Environment()
tools = Tools(env)
system_prompt = "Goals, constraints, and how to act"
while True:
    action = llm.run(system_prompt + env.state) # agentic 1
    env.state = tools.run(action)               # agentic 2
```

- ⛳ Agentic:
  1. more informative tool usage → help LLM Agent understand binary code
  2. more complex environment, less supervision

[1] Artificial Intelligence: A Modern Approach, 1995

LLM Agent

- Agent[1]: anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.
- LLM Agent workflow:

```python
env = Environment()
tools = Tools(env)
system_prompt = "Goals, constraints, and how to act"
while True:
    action = llm.run(system_prompt + env.state) # agentic 1
    env.state = tools.run(action)               # agentic 2
```

- ⛳ Agentic:
  1. more informative tool usage → help LLM Agent understand binary code
  2. more complex environment, less supervision → guide LLM Agent to explore binary[2]

[1] Artificial Intelligence: A Modern Approach, 1995

[2] Similar to repo-level exploration (RepoAudit, ICML '25) in source code

Motivation

Current binary analysis tools for agents[1] have two limitations:

[1] EnIGMA: Interactive Tools Substantially Assist LM Agents in Finding Security Vulnerabilities, ICML '25

[2] We assume perfect disassembly here

Motivation

Current binary analysis tools for agents[1] have two limitations:

1. Co-operation gap between different binary code formats

```
switch ( v39[10] & 0xF )
{
  case 1:
    p_s2 = &s2;
    goto LABEL_80;
  case 2:
    inet_aton(v41, (struct in_addr *)&s2);
    v41 = strtok(0, ", \t/-");
    v43 = (size_t)v41;
    if ( v41 )
    {
      p_s2 = (char **)&v89;
LABEL_80:
      inet_aton(v41, (struct in_addr *)p_s2);
LABEL_81:
      v43 = 0;
LABEL_89:
      v44 = 0;
    }
```

```
1  int __cdecl read_and_validate_size(int fd, _WORD *a2)
2  {
3    _DWORD input_buffer[2]; // [esp+10h] [ebp-18h] BYREF
4    __int16 v4; // [esp+18h] [ebp-10h]
5    __int16 v5; // [esp+1Ah] [ebp-Eh]
6    int v6; // [esp+1Ch] [ebp-Ch]
7
8    v6 = 0;
9    v5 = 0;
10   input_buffer[0] = 0;
11   input_buffer[1] = 0;
12   v4 = 0;
13   if ( read(fd, input_buffer, 0xAu) == -1 )
14   {
15     perror("read: ");
16     return 0;
17   }
```

(a) Decompiled code

concise (e.g., 50-100 LoC)

fidelity issues: e.g., goto-based control flow, wrong variable length

```
.text:00014E58 sub_14E58                       ; CODE XREF: dhcpd+C64↑p
.text:00014E58                                 ; sub_14F4C+98↓p ...
.text:00014E58                 PUSH            {R4-R8,LR}
.text:00014E5C                 MOV             R5, R0
.text:00014E60                 MOV             R7, R1
.text:00014E64                 MOV             R6, R2
.text:00014E68                 BL              sub_14DFC
.text:00014E6C                 SUBS            R4, R0, #0
.text:00014E70                 BNE             loc_14E80
.text:00014E74                 BL              sub_14DA8
.text:00014E78                 SUBS            R4, R0, #0
.text:00014E7C                 BEQ             loc_14E80
.text:00014E80
.text:00014E80 loc_14E80                       ; CODE XREF: sub_14E58+18↑j
.text:00014E80                 MOV             R0, R5
.text:00014E84                 MOV             R3, R4
.text:00014E88                 ADD             R2, R5, #0x10
.text:00014E8C
.text:00014E8C loc_14E8C                       ; CODE XREF: sub_14E58+40↓j
.text:00014E8C                 LDR             R1, [R0],#4
.text:00014E90                 CMP             R0, R2
.text:00014E94                 STR             R1, [R3],#4
.text:00014E98                 BNE             loc_14E8C
.text:00014E9C                 MOV             R0, #0  ; timer
.text:00014EA0                 STR             R7, [R4,#0x10]
.text:00014EA4                 BL              time
.text:00014EA8                 ADD             R2, R0, R6
.text:00014EAC                 STR             R2, [R4,#0x14]
```

(b) Assembly code

verbose (e.g., 150-200 LoC)

zero-semantic gap[2], less structural

Figure 3: Relying on decompiled code and/or assembly code is insufficient

[1] EnIGMA: Interactive Tools Substantially Assist LM Agents in Finding Security Vulnerabilities, ICML '25

[2] We assume perfect disassembly here

Motivation

Current binary analysis tools for agents have two limitations:

[1] SV-TrustEval-C: Evaluating Structure and Semantic Reasoning in Large Language Models for Source Code Vulnerability Analysis, IEEE S&P '25

Motivation

Current binary analysis tools for agents have two limitations:

2. Explore at the function level, with limited guide on intra-procedural control and data flow[1]


```
{
  "action": "decompile --function_name FUN_00400df0 rap\n",
  "observation": "Decompilation Found!\n\nundefined4 FUN_00400df0(void)\n\n{\n  int iVar1;\n  basic_ostream *pbVar2;\n  basic_string local_30 [36];\n  undefined4 local_c;\n  \n  local_c = 0;\n  FUN_00400cc0();\n  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string();\n                    /* try { // try from 00400e0d to 00400e7a has its
  CatchHandler @ 00400e7f */\n  pbVar2 = std::operator<<((basic_ostream *)std::cout,\"help me\");\n  std::basic_ostream<char,std::char_traits<char>>::operator<<\n            ((basic_ostream<char,
  std::char_traits<char>> *)pbVar2,\n             std::endl<char,std::char_traits<char>>);\n  std::operator>>((basic_istream *)std::cin,local_30);\n  iVar1 = FUN_00400ce0(local_30);\n  if (iVar1
  != 0) {\n    pbVar2 = std::operator<<((basic_ostream *)std::cout,\"you found me!\");\n    std::basic_ostream<char,std::char_traits<char>>::operator<<\n              ((basic_ostream<char,
  std::char_traits<char>> *)pbVar2,\n               std::endl<char,std::char_traits<char>>);\n  }\n  local_c = 0;\n  std::__cxx11::basic_string<char,std::char_traits<char>,
  std::allocator<char>>::~basic_string\n            ((basic_string<char,std::char_traits<char>,std::allocator<char>> *)local_30);\n  return local_c;\n}\n\n",
  "response": "DISCUSSION\nThe entry function doesn't provide much information. Let's try to decompile the function that's being called as the main function, which is FUN_00400df0.
  \n\n```\ndecompile --function_name FUN_00400df0 rap\n```",
  "state": "{\"open_file\": \"n/a\", \"working_dir\": \"/__home__udiboy__projects__ctf_datasets__LLM_CTF_Database__2020__CSAW-Finals__rev__rap\", \"interactive_session\": \"n/a\"}",
  "thought": "DISCUSSION\nThe entry function doesn't provide much information. Let's try to decompile the function that's being called as the main function, which is FUN_00400df0.\n\n",
  "execution_time": 0.10650779701245483
},
```


Figure 4: Trajectories from EnIGMA, ICML '25

💥 Cascading effect: fail to understand the tools' output → hallucinate on next action → stuck in exploration → finally might exceed context length

[1] SV-TrustEval-C: Evaluating Structure and Semantic Reasoning in Large Language Models for Source Code Vulnerability Analysis, IEEE S&P '25

2. ClearAgent

Overview

A novel Binary Interface for software engineering agent[1] with two tool sets:

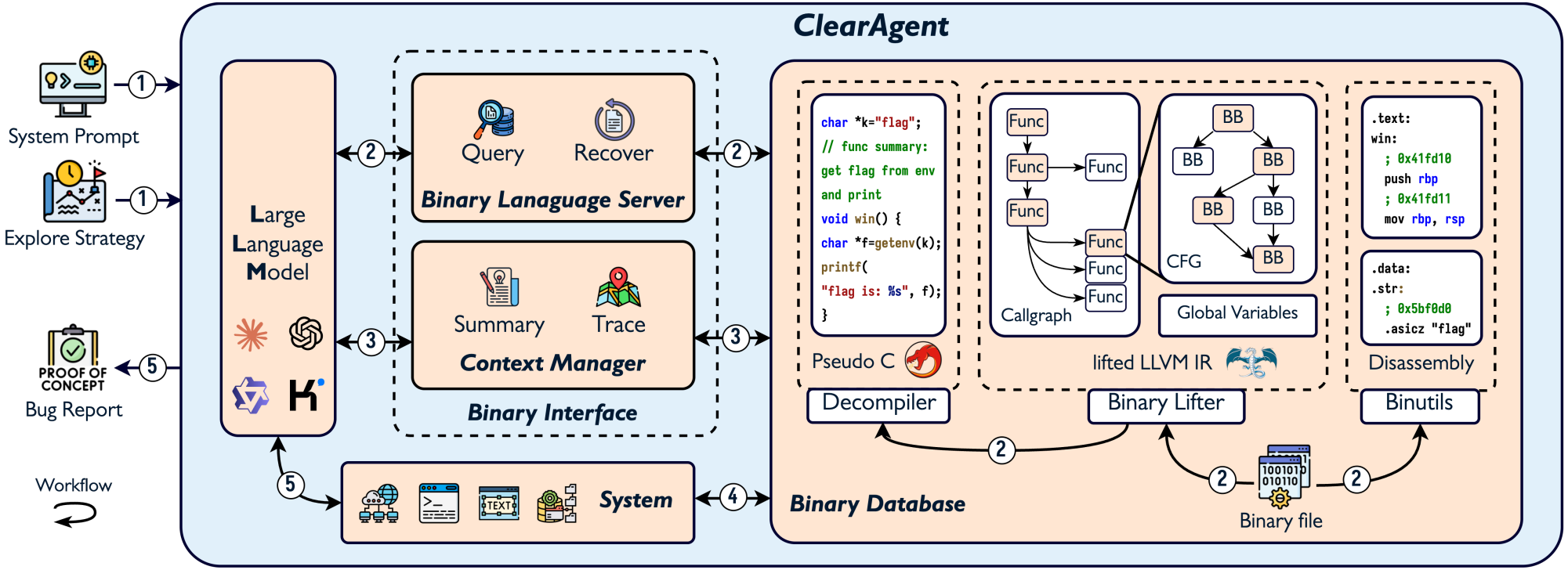
1. **Binary Language Server**: Information retrieval from / update to binary code
2. **Context Manager**: Multi-level exploring binary with summary and feedback

[1] Under the Clearblue research project: https://clearblueinnovations.org/

Overview

A novel Binary Interface for software engineering agent[1] with two tool sets:

1. **Binary Language Server**: Information retrieval from / update to binary code
2. **Context Manager**: Multi-level exploring binary with summary and feedback



[1] Under the Clearblue research project: https://clearblueinnovations.org/

Binary Language Server

- Aims at providing a friendly interface for LLM to understand binary code

[1] Plankton: Reconciling Binary Code and Debug Information, ASPLOS '24

[2] Pinpoint: fast and precise sparse value flow analysis for million lines of code, PLDI '18

Binary Language Server

- Aims at providing a friendly interface for LLM to understand binary code
- Two-fold friendliness:
  - LLM-friendly: mapping between three formats (Assembly, IR, Pseudo code)
  - Analyzer-friendly: IR[1] as the primary query target, supporting symbolic analysis[2]

[1] Plankton: Reconciling Binary Code and Debug Information, ASPLOS '24

[2] Pinpoint: fast and precise sparse value flow analysis for million lines of code, PLDI '18

Binary Language Server

- Aims at providing a friendly interface for LLM to understand binary code
- Two-fold friendliness:
  - LLM-friendly: mapping between three formats (Assembly, IR, Pseudo code)
  - Analyzer-friendly: IR[1] as the primary query target, supporting symbolic analysis[2]
- Scoped design, narrows down from binary to instruction

Format \ Scope	Binary	Function	Basicblock	Instruction	Global Var
Assembly	✓, target triple	✓, address range	✓, address range	✓, address, size	✓, raw bytes
LLVM IR	✓, symbol	✓, Caller/Callee Signature	✓, Predecessors Successors	✓, Def-Use	✓, User
Pseudo code	✓	✓	✓		

Table 1: Queries supported by Binary Language Server, ✓ represents directly querying the code body

[1] Plankton: Reconciling Binary Code and Debug Information, ASPLOS '24

[2] Pinpoint: fast and precise sparse value flow analysis for million lines of code, PLDI '18

Binary Language Server example

How does ClearAgent build the symbolic summary of a function?

```
int32_t do_sta_enrollee_wifi(int32_t arg1)

    int32_t $v0 = *__stack_chk_guard
    void var_11c
    memset(&var_11c, 0, 0x80)
    void var_9c
    memset(&var_9c, 0, 0x40)
    char var_5c
    memset(&var_5c, 0, 0x40)
    int32_t $v0_1 = getenv(0x461da8)  {"wps_sta_enrollee_pin"}
    sprintf(0x495850, 0x4628d8, arg1)  {"qcawifi.%s.wps_enable"}
    int32_t $v0_3 = atoi(uci_safe_get(0x495850))
    sprintf(0x495850, 0x4628f0, arg1)  {"qcawifi.%s.wps_vap"}
    strcpy(&var_9c, uci_safe_get(0x495850))
    sprintf(0x495850, 0x462904, &var_9c)  {"qcawifi.%s.up"}
    int32_t $v0_6 = atoi(uci_safe_get_state(0x495850))
    sprintf(0x495850, 0x462914, &var_9c)  {"qcawifi.%s.ifname"}
    strcpy(&var_5c, uci_safe_get_state(0x495850))

    if (access(0x461dd0, 0) == 0)  {"/var/tmp/wps_result"}
        unlink(0x461dd0)  {"/var/tmp/wps_result"}

    if ($v0_1 ≠ 0 && $v0_3 ≠ 0 && $v0_6 ≠ 0 && sx.d(var_5c) ≠ 0)
        sprintf(&var_11c, 0x462944, &var_5c, $v0_1, &_gp)  {"
            hostapd_cli -i %s wps_pin any %s"}
        system(&var_11c)
```

(a) Decompiled code

```
  %tmp38 = load [25 x i8]** @global.global_var_49188c, !asm !6193
  %tmp39 = getelementptr [25 x i8]* %tmp38, i32 303, i32 17, !asm !6194
  %tmp40 = call i8* @getenv(i8* %tmp39), !asm !6194
  %tmp41 = ptrtoint i8* %tmp40 to i32, !asm !6194

...

block_416508:                                     ; preds = %block_4164bc
  %tmp94 = load [25 x i8]** @global.global_var_49188c, !asm !6233
  %tmp95 = getelementptr [25 x i8]* %tmp94, i32 422, i32 14, !asm !6234
  %tmp96 = call i32 (i8*, i8*, ...)* @sprintf(i8* %stack.alias.prop18_-284, i8*
  %tmp95, i32 %tmp36, i32 %tmp41), !asm !6234
  %tmp97 = call i32 @system(i8* %stack.alias.prop18_-284), !asm !6235
```

(b) IR code

```
004164a0  8fbc0010   lw      $gp, 0x10($sp) {var_128}

004164a4  12200008   beqz    $s1, 0x4164c8
004164a8  8f828cdc   lw      $v0, -0x7324($gp)  {__stack_chk_guard}

004164ac  12600006   beqz    $s3, 0x4164c8
004164b0  00000000   nop

004164b4  12400003   beqz    $s2, 0x4164c4
004164b8  83a200dc   lb      $v0, 0xdc($sp) {var_5c}

004164bc  14400012   bnez    $v0, 0x416508
004164c0  8f85801c   lw      $a1, -0x7fe4($gp)  {data_49188c}
```

```
00416508  27a4001c   addiu   $a0, $sp, 0x1c {var_11c}
0041650c  8f998b70   lw      $t9, -0x7490($gp)  {sprintf}
00416510  27a600dc   addiu   $a2, $sp, 0xdc {var_5c}
00416514  02203821   move    $a3, $s1
00416518  0320f809   jalr    $t9
0041651c  24a52944   addiu   $a1, $a1, 0x2944  {0x462944, "hostapd_cli
00416520  8fbc0010   lw      $gp, 0x10($sp) {var_128}
00416524  8f998b6c   lw      $t9, -0x7494($gp)  {system}
00416528  0320f809   jalr    $t9
0041652c  27a4001c   addiu   $a0, $sp, 0x1c {var_11c}
00416530  1000ffe4   b       0x4164c4
00416534  8fbc0010   lw      $gp, 0x10($sp) {var_128}  {_gp}
```

(c) Assembly code

Figure 7: Three binary code formats of the function `do_sta_enrollee_wifi` in CVE-2022-46593. ClearAgent (1) understands this function using decompiled code, then (2) builds the data dependency between `%tmp40` and `%stack.alias.prop18_-284` while (3) checking the fidelity issue using assembly code

Context Manager

- Guiding LLM Agent to explore the binary at function- and basicblock-level
- Contexts are updated with each query to the binary language server

[1] Standardized Operating Procedures (SOPs) in MetaGPT, ICLR '24

[2] RE-Mind: a First Look Inside the Mind of a Reverse Engineer, Usenix Security '22

Context Manager

- Guiding LLM Agent to explore the binary at function- and basicblock-level
- Contexts are updated with each query to the binary language server

Query \ Scope	Binary	Function	Basicblock
Summary	natural language	natural language, symbolic	natural language, symbolic
Count		visit count of itself, caller/callee	visit count of itself, preds/succs
Trace	visited sub-callgraph	visited sub-CFG	

Table 2: Queries supported by Context Manager

[1] Standardized Operating Procedures (SOPs) in MetaGPT, ICLR '24

[2] RE-Mind: a First Look Inside the Mind of a Reverse Engineer, Usenix Security '22

Context Manager

- Guiding LLM Agent to explore the binary at function- and basicblock-level
- Contexts are updated with each query to the binary language server

Query \ Scope	Binary	Function	Basicblock
Summary	natural language	natural language, symbolic	natural language, symbolic
Count		visit count of itself, caller/callee	visit count of itself, preds/succs
Trace	visited sub-callgraph	visited sub-CFG	

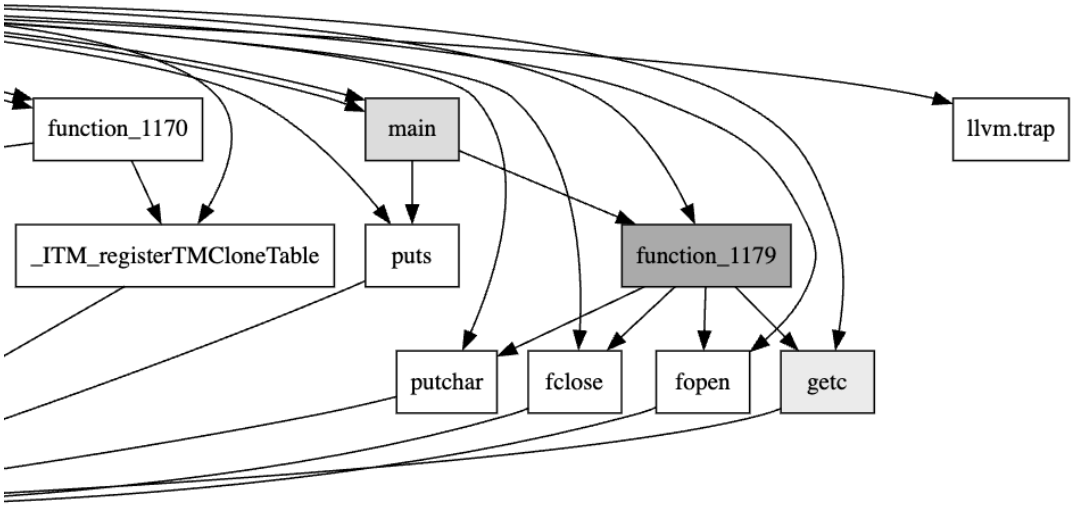
Table 2: Queries supported by Context Manager

- Collaborating with the exploration strategy
  - Context is managed through phased exploration[1]
    - e.g., recovery $\longrightarrow$ analysis $\longrightarrow$ validation
  - Each phase is guided by a pre-defined searching strategy[2]
    - e.g., Forward / Backward, BFS / DFS

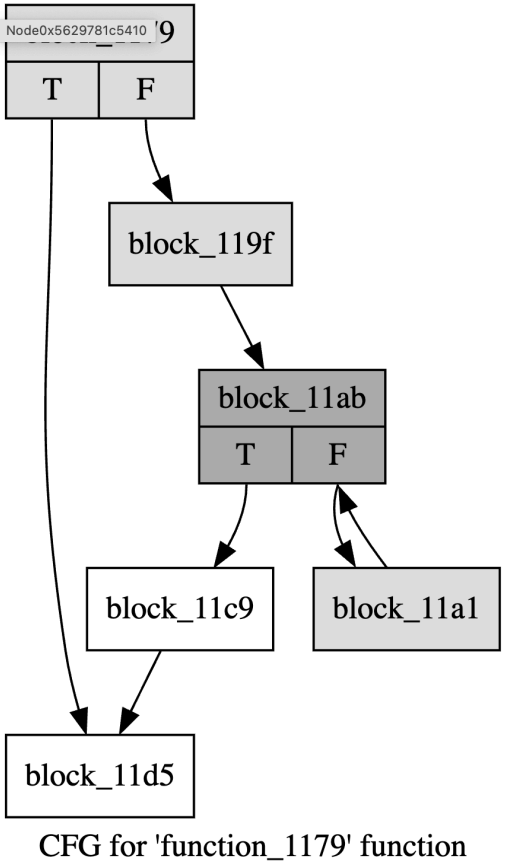
[1] Standardized Operating Procedures (SOPs) in MetaGPT, ICLR '24

[2] RE-Mind: a First Look Inside the Mind of a Reverse Engineer, Usenix Security '22

Context Manager example



(a) binary's Callgraph



(b) function's CFG

Figure 9: Visited node and count formed sub-graph on callgraph and CFG
The darker the color, the more visit

3. Preliminary Evaluation

RQ1 - Binary Language Server

- Dataset: 90 binary challenges from an entry-level CTF contest[1]
  - Only 19/90 can be solved by the SOTA CTF Agents

[1] NYU CTF Bench: A Scalable Open-Source Benchmark Dataset for Evaluating LLMs in Offensive Security, NIPS '24
[2] CRAKEN: Cybersecurity LLM Agent with Knowledge-Based Execution, Arxiv '25
[3] EnIGMA: Interactive Tools Substantially Assist LM Agents in Finding Security Vulnerabilities, ICML '25

RQ1 - Binary Language Server

- Dataset: 90 binary challenges from an entry-level CTF contest[1]
  - Only 19/90 can be solved by the SOTA CTF Agents
- Metrics:
  - Pwn: Can ClearAgent identify the buggy point and write a PoC script?
  - Reversing: Can ClearAgent understand the logic and get the flag?

[1] NYU CTF Bench: A Scalable Open-Source Benchmark Dataset for Evaluating LLMs in Offensive Security, NIPS '24
[2] CRAKEN: Cybersecurity LLM Agent with Knowledge-Based Execution, Arxiv '25
[3] EnIGMA: Interactive Tools Substantially Assist LM Agents in Finding Security Vulnerabilities, ICML '25

RQ1 - Binary Language Server

- Dataset: 90 binary challenges from an entry-level CTF contest[1]
  - Only 19/90 can be solved by the SOTA CTF Agents
- Metrics:
  - Pwn: Can ClearAgent identify the buggy point and write a PoC script?
  - Reversing: Can ClearAgent understand the logic and get the flag?

Agent	LLM	Reversing	Pwn	All
ClearAgent	Qwen 3 Plus	7/11	8/8	15/19
CRAKEN[2]	Claude 3.5 Sonnet	11/11	6/8	17/19
EnIGMA[3]	Claude 3.5 Sonnet	6/11	6/8	12/19

Table 3: Preliminary evaluation on CTF challenges with an early version of ClearAgent (no guidance, only IR)

[1] NYU CTF Bench: A Scalable Open-Source Benchmark Dataset for Evaluating LLMs in Offensive Security, NIPS '24
[2] CRAKEN: Cybersecurity LLM Agent with Knowledge-Based Execution, Arxiv '25
[3] EnIGMA: Interactive Tools Substantially Assist LM Agents in Finding Security Vulnerabilities, ICML '25

Limitation 1 - Inaccurate IR

The lifted binary code is inaccurate due to obfuscation or errors in the lifter.

Limitation 1 - Inaccurate IR

The lifted binary code is inaccurate due to obfuscation or errors in the lifter.

```
loc_400D24:                             ; CODE XREF: .text:loc_400D24↑j
                jmp     short near ptr loc_400D24+1
; ---------------------------------------------------------------------------
                dw 48C0h
                dq 8B48FFFFFF388563h, 0FFFF30858948F87Dh, 8B48FFFFFDF2E8FFh
                dq 0C13948FFFFFF308Dh, 858B00000089830Fh, 0F87D8B48FFFFFF38h
                dq 89FFFFFF38B56348h, 0FD86E8FFFFFF2C85h, 48B84808BE0FFFFFh
                dq 0EB04EBC03140EC83h, 0FFFF2C958BB848F5h, 31C231B84806EBFFh
                dq 3B84802EBCA31DAh, 4802EBFFFFFF3895h, 0EBFFFFFF388D8BB8h
                dq 6348C0FF48B84805h, 943B31C0C303EBC1h, 0C70A74FFFFFF4085h
                dq 0FFFFFF3C85h, 0FFFF38858B00EB00h, 0FF38858901C083FFh
                dq 8BFFFFFF4DE9FFFFh, 0C48148FFFFFF3C85h, 2E66C35D000000E0h
                dq 841F0Fh
```

```
loc_400D24:                             ; CODE XREF: sub_400CE0+F2↓j
                nop
                inc     eax
                movsxd  rax, [rbp+i]
                mov     rdi, [rbp+string]
                mov     [rbp+var_D0], rax
                call    __ZNKSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEE6lengthEv
```

Figure 11: Junk bytes in 2020f-rev-rap

Limitation 1 - Inaccurate IR

The lifted binary code is inaccurate due to obfuscation or errors in the lifter.

```
loc_400D24:                             ; CODE XREF: .text:loc_400D24↑j
                jmp     short near ptr loc_400D24+1
; ---------------------------------------------------------------------------
                dw 48C0h
                dq 8B48FFFFFF388563h, 0FFFF30858948F87Dh, 8B48FFFFFDF2E8FFh
                dq 0C13948FFFFFF308Dh, 858B00000089830Fh, 0F87D8B48FFFFFF38h
                dq 89FFFFFF38B56348h, 0FD86E8FFFFFF2C85h, 48B84808BE0FFFFFh
                dq 0EB04EBC03140EC83h, 0FFFF2C958BB848F5h, 31C231B84806EBFFh
                dq 3B84802EBCA31DAh, 4802EBFFFFFF3895h, 0EBFFFFFF388D8BB8h
                dq 6348C0FF48B84805h, 943B31C0C303EBC1h, 0C70A74FFFFFF4085h
                dq 0FFFFFF3C85h, 0FFFF38858B00EB00h, 0FF38858901C083FFh
                dq 8BFFFFFF4DE9FFFFh, 0C48148FFFFFF3C85h, 2E66C35D000000E0h
                dq 841F0Fh

loc_400D24:                             ; CODE XREF: sub_400CE0+F2↓j
                nop
                inc     eax
                movsxd  rax, [rbp+i]
                mov     rdi, [rbp+string]
                mov     [rbp+var_D0], rax
                call    __ZNKSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEE6lengthEv
```

Figure 11: Junk bytes in 2020f-rev-rap

```
.text:00000000000012A4 ; __unwind {
.text:00000000000012A4 push    rbp
.text:00000000000012A5 mov     rbp, rsp
.text:00000000000012A8 sub     rsp, 20h
.text:00000000000012AC mov     rax, 6E3762597041 6742h
.text:00000000000012B6 mov     edx, 44777343h
.text:00000000000012BB mov     qword ptr [rbp+var_20], rax
.text:00000000000012BF mov     qword ptr [rbp+var_20+8], rdx
```

```
define i64 @main(i64 %arg, i8** %arg1) {
block_12a4:
%stack.stack_var_-12 = alloca i64
%stack.stack_var_-16 = alloca i64
%stack.stack_var_-24 = alloca i64
%stack.stack_var_-40 = alloca i64
store i64 7941924604166104898, i64* %stack.stack_var_-40, !asm !0
```

Figure 12: Over-optimized stack variable

Limitation 2 - Insufficient Guidance

Stuck or hallucinate during inter-procedural exploration, and exceed the round limitation

Limitation 2 - Insufficient Guidance

Stuck or hallucinate during inter-procedural exploration, and exceed the round limitation

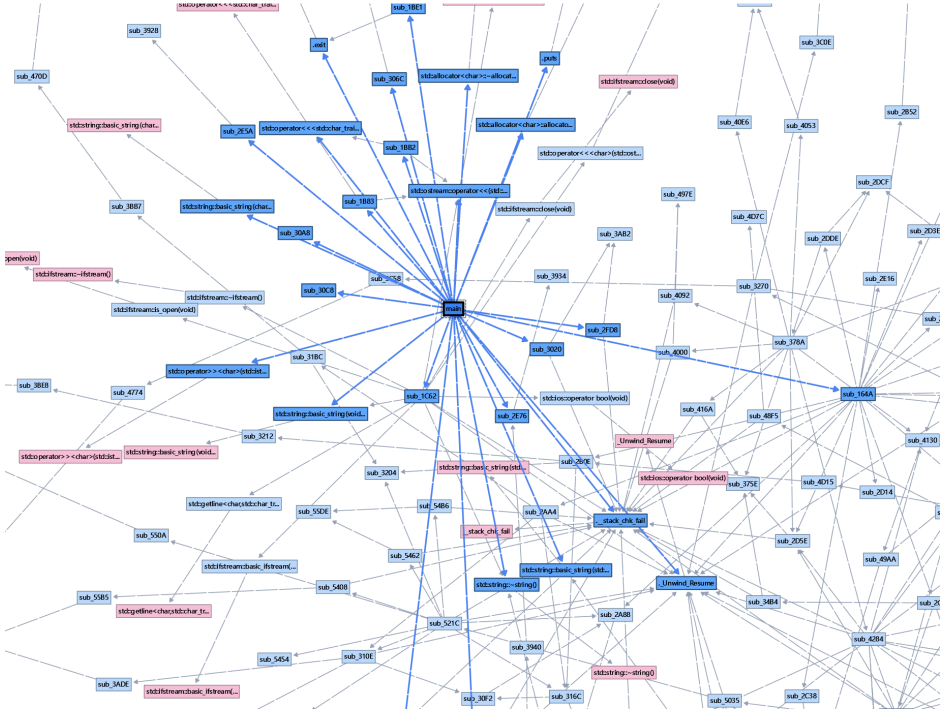


Figure 13: Deep calling context in C++ std functions

Limitation 2 - Insufficient Guidance

Stuck or hallucinate during inter-procedural exploration, and exceed the round limitation

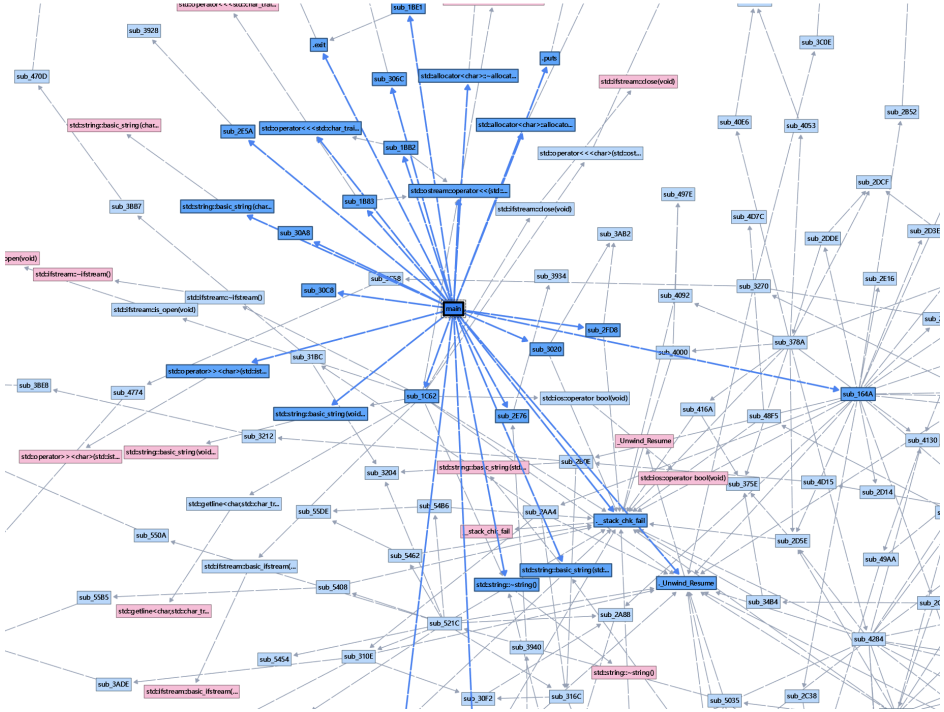


Figure 13: Deep calling context in C++ std functions

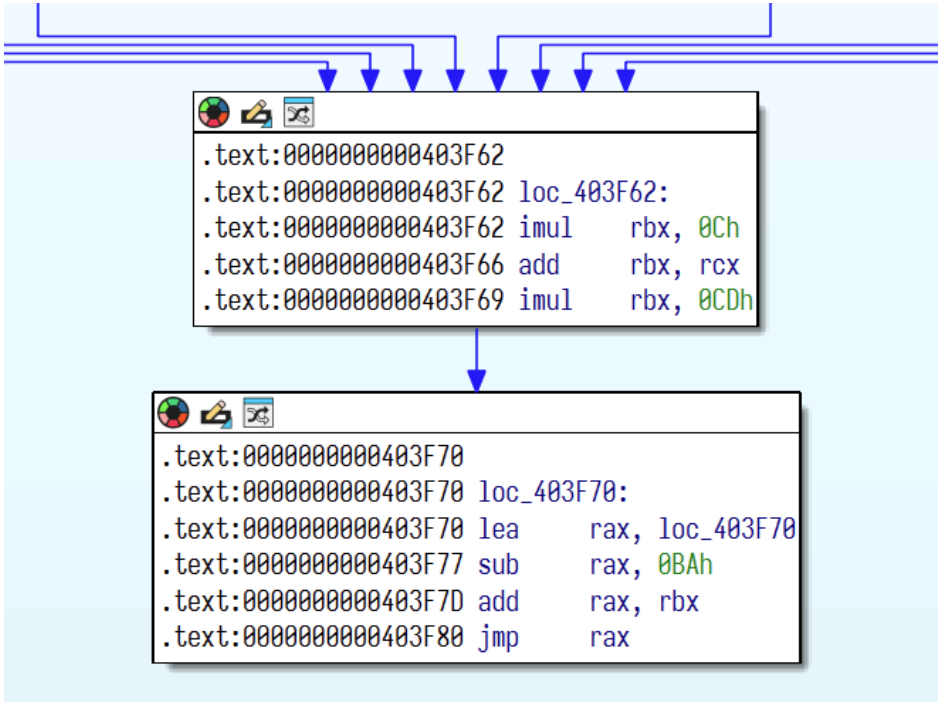
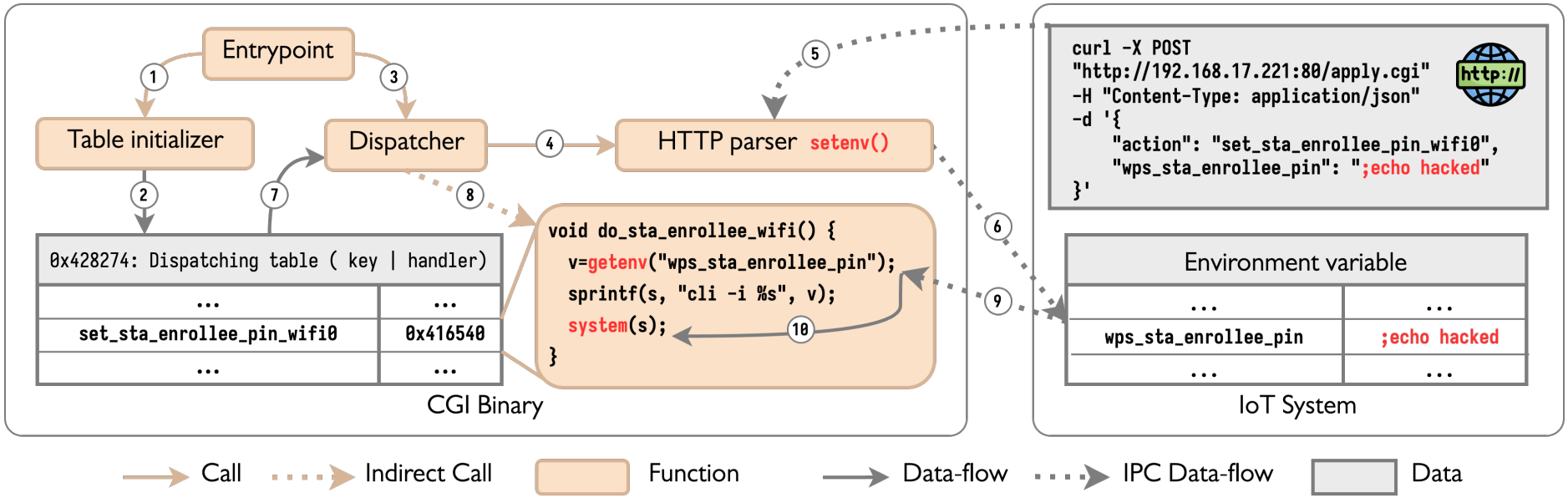


Figure 14: Indirect call targets are missed by static analysis

RQ2 - Context Manager

- Demo: CGI Binary in TRENDnet IoT firmware[1]
  - **Recovery**: find and recover the dispatching table using DFS/Forward strategy
  - **Analysis**: analyze each handler using BFS/Forward strategy at the indirect callsite
  - **Validation**: find the `setenv` operation using DFS/Backward strategy



[1] https://nvd.nist.gov/vuln/detail/CVE-2022-46593

4. Future Work

Directions

We envision the following three future directions:

- Agentic IR refinement: fixing fidelity issues in lifted IR, such as type recovery[1]
- Multi-binary analysis[2]: cross-binary vulnerability in IoT firmware
- Huge binary analysis (~1GB): pruning uninteresting functions in rigorous static analysis

[1] Think of IDA Pro: https://hex-rays.com/blog/igors-tip-of-the-week-42-renaming-and-retyping-in-the-decompiler

[2] Karonte: Detecting Insecure Multi-binary Interactions in Embedded Firmware, IEEE S&P '20

Any Question