Faster and Better: Detecting Vulnerabilities in Linux-based IoT Firmware with Optimized Reaching Definition Analysis

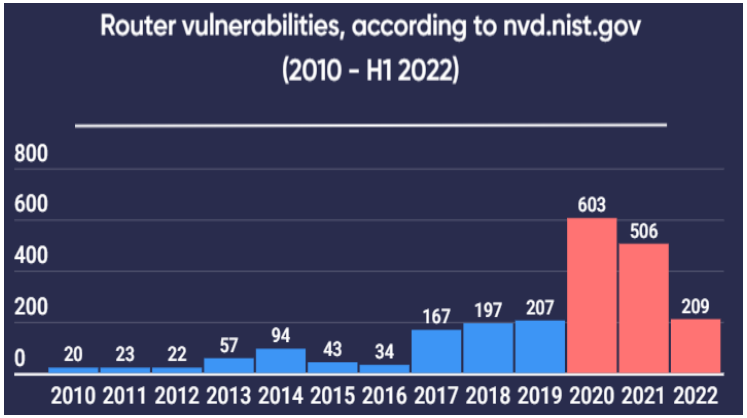
Zicong Gao+, Chao Zhang*, Hangtian Liu, Wenhou Sun, Zhizhuo Tang, Liehui Jiang, Jianjun Chen, and Yong Xie

IoT Devices

- The number of IoT devices has reached 15.14 billion by the end of 2023
- Rich application scenarios from life to production
  - e.g. Smart Devices, Wearables, Webcams, Connected Vehicles, Industrial control system

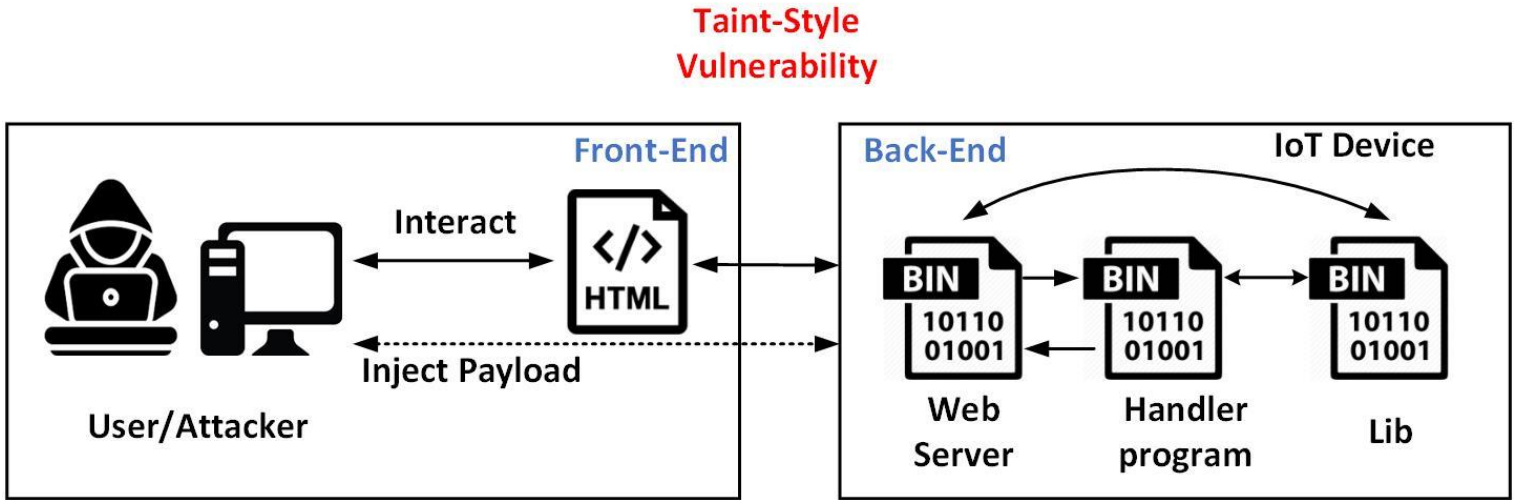
IoT Device Vulnerabilities

- IoT devices suffer from the serious cyber threats
  - Network devices (e.g. router, webcam, firewall) are the most commonly attacked IoT devices
  - Vulnerabilities in network security equipment have extremely serious impacts



Taint-style Vulnerabilities in IoT

- Vulnerabilities, especially **taint-style vulnerabilities,** are significant security threats to IoT devices



Threat model of taint-style vulnerabilities in IoT scenarios

How to detect taint-style vulnerabilities in IoT devices?

Proposed Method

- Dynamic solutions
  - Fuzzing

  Pros: accurate, high true positive rate

  Cons: requiring emulation, difficult to explore deep paths

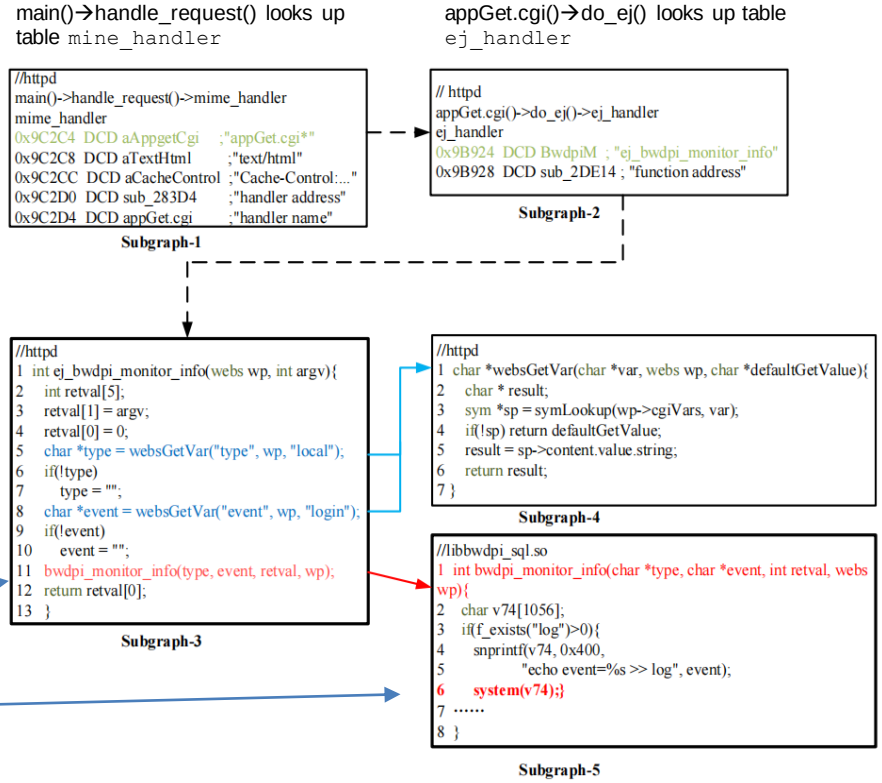
- Static solutions
  - Taint analysis

  Pros: scalability, high coverage

  Cons: high false positive rate, heavyweight symbolic execution

Motivation Example

- Limitation of existing works
  - SaTC & KARONTE fail to alert
  - Reason: functions containing source & sink points are only indirectly called functions or library functions, neglected by common CFG construction strategies
  - Example: the function `ej_hwdpi_monitor_info` is not in the CFG and the `libbwdpi_sql.so` is not analyzed either



Motivation Example

- Limitation of existing works
  - Symbolic execution-based taint tracking is …
    - time consuming: SaTC costs 0.5h~30h per sample, and the analysis time will increase by 2 to 3 times when libraries are included……
    - not practicable in real-world: It faces problems such as state explosion, path explosion, and constraint solving complexity……

Motivation Example

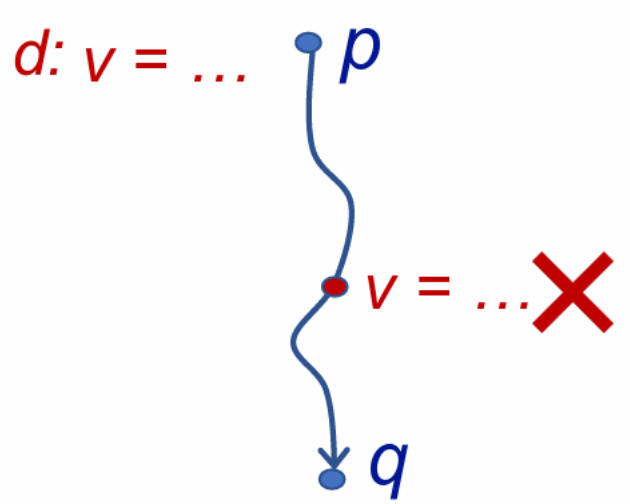
- Limitation of existing works
  - Symbolic execution-based taint tracking is …
    - time consuming: SaTC costs 0.5h~30h per sample, and the analysis time will increase by 2 to 3 times when libraries are included……
    - not practicable in real-world: It faces problems such as state explosion, path explosion, and constraint solving complexity……

More often than not, static analysis should be fast and productive

Our solution

- Reaching definition analysis

A definition d of a variable v at program point p reaches a point q if there is a path from p to q such that d is not “killed” along that path



Our solution

- Reaching definition analysis

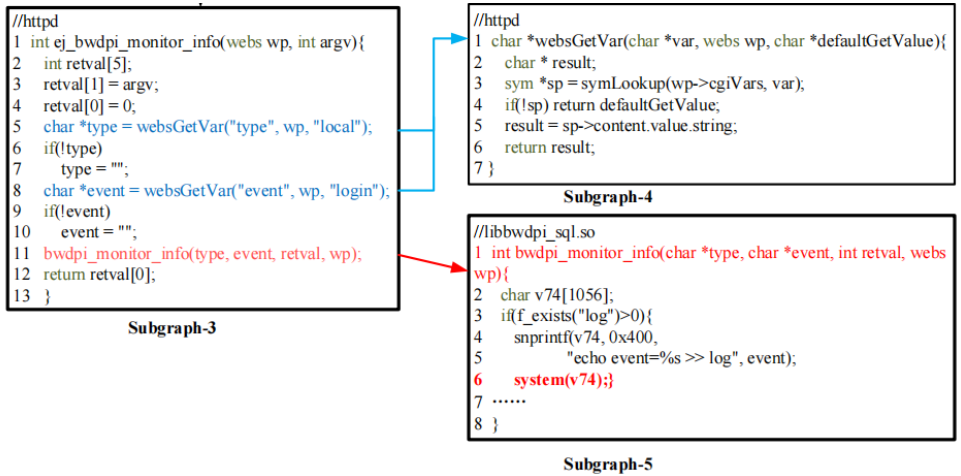
  A definition d of a variable v at program point p reaches a point q if there is a path from p to q such that d is not “killed” along that path

- RDA-based taint analysis

  A definition d of a taint variable v at source p reaches sink q if there is a path from p to q such that d is not “killed” along that path

Our solution

- RDA-based taint analysis
  - Definitions to taint variable `type` and `event` at sources line 5 & 8 assigned by `websGetvar` reaches sink function `system()`
  - Definition value of variable `v74` is related to `type/event` and violates the vulnerability rule

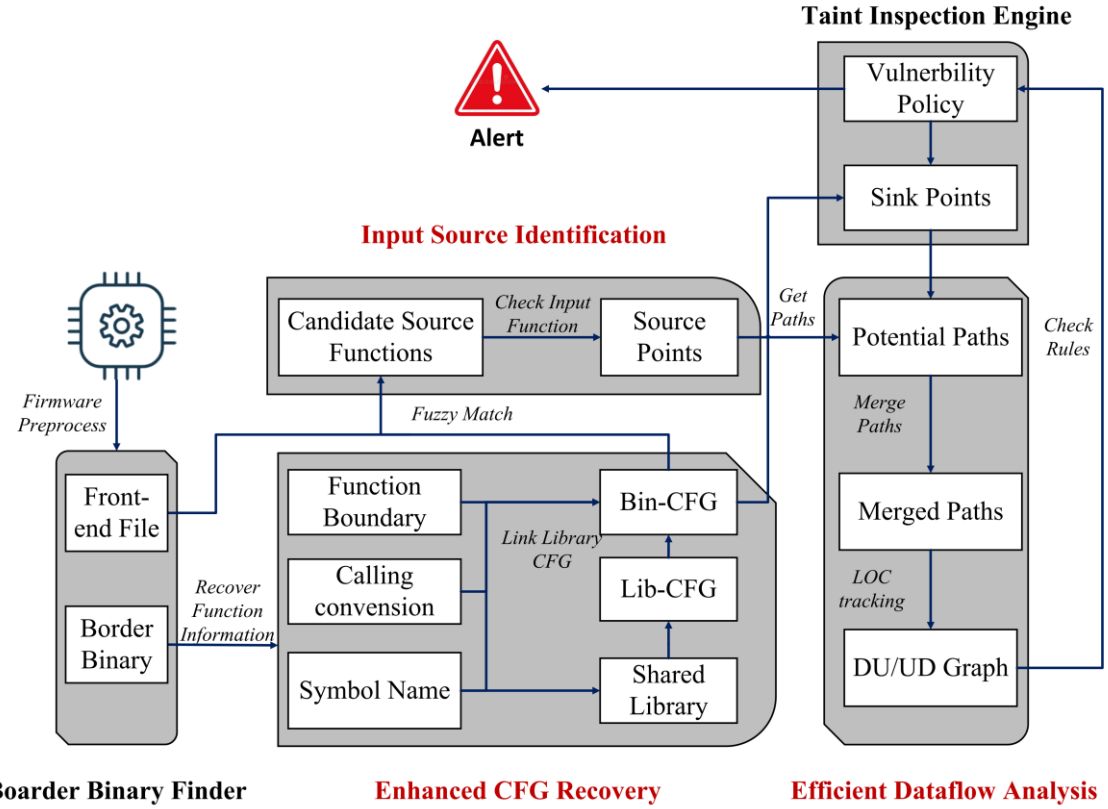


Challenges

- 1. Comprehensive CFG Recovery
  - Functions only invoked by indirect calls are difficult to identify
  - Connecting binary CFG and library CFG increases analysis efforts
- 2. Precise Source Point Identification
  - Manually specifying source functions requires expert knowledge and customization
  - Pattern-based string matching methods cannot utilize semantic information, missing some potential source points
- 3. Efficient Taint Tracking
  - Massive paths between source/sink points can lead to path explosion in RDA

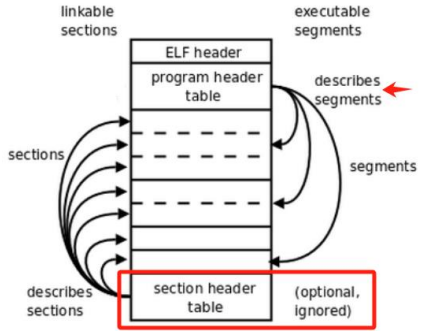
HermeScan

Architecture



Enhanced CFG Recovery

- Function boundary
  - Full binary linear scan by dividing function boundaries according to function prologue
- Symbol table
  - When the section header table is stripped, locate the address of symbol table from metadata in `PT_DYNAMIC` segment
- Calling conventions
  - Aggressive but complete recovery strategy by setting default CC for each function
- Connect the Bin-CFG with the Lib-CFG



Source Input Identification

- Fuzzy Matching
  - Consider the **word form similarity** and **semantic similarity** of keywords appearing in the front and back ends. Regard functions reference these matched strings as candidate sources
  - The normalized edit distance is used to calaulate the FormatSim

$$FormatSim(S1, S2) = 1 - Edit(S1, S2)/(L(S1) + L(S2))$$

e.g. hostname_1.1 ⟷ hostname_%s

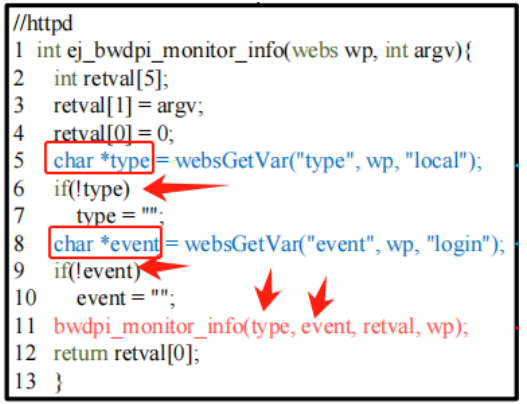
  - The BERT model is used to calculate the semantic similarity of two strings

$$SemanticSim(S1, S2) = \begin{cases} Cosine(S1, S2), & Others \\ 0, & \frac{LCS(S1,S2)}{Min(L(S1),L(S2))} < \theta \end{cases}$$

e.g. “request from %s is banned for security” ⟺ “sec_ip_ban”

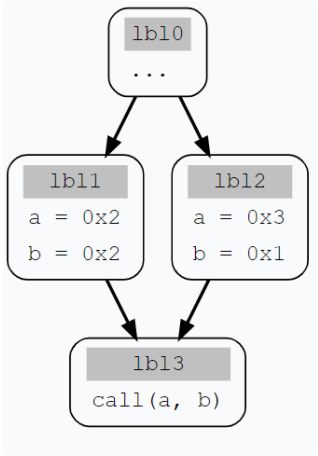
Source Input Identification

- Candidate Function Checking
  - Goal: Remove infeasible candidate functions and mark parts of the function arguments or return values as taint sources
  - Idea: Check whether the parameters would receive values from external input and whether the return values would be used by following operations

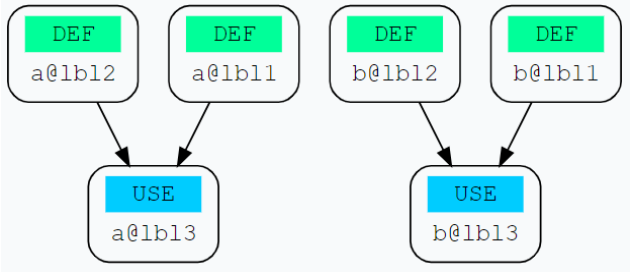


Efficient Dataflow Analysis

- Lightweight, context-sensitive, on-demand, inter-procedural analysis
  - Lightweight: **RDA-based** instead of symbolic exeuction-based taint tracking
  - Context-Sensitive: considers context information to enable fine-grained dataflow analysis
  - On-demand:
    1) Only step into functions with tainted parameters for interprocedural analysis
    2) Use summary for common library functions



Control flow graph



Def-use graph

Efficient Dataflow Analysis

- Lightweight, context-sensitive, on-demand, inter-procedural analysis

```
1  // ssi
2  void setup_wizard_mydlink(int a1){
3      char *v4;
4      v4 = getenv("sys_service");
5      updown_services(0, v4);
6      post2nvram(a1);
7      response_page = get_response_page();
8  }
9  int updown_services(int mode, char *sys_service){
10     if(mode) return func2(sys_service);
11     return func3(sys_service);
12 }
13 int func2(char *a1){
14     char buf[1028];
15     if (a1 && *a1){
16         strcpy(buf, a1);
17     }
18 }
```

Mark variable v4 as a taint source

Efficient Dataflow Analysis

- Lightweight, context-sensitive, on-demand, inter-procedural analysis

```
1  // ssi
2  void setup_wizard_mydlink(int a1){
3      char *v4;
4      v4 = getenv("sys_service");
5      updown_services(0, v4);
6      post2nvram(a1);
7      response_page = get_response_page();
8  }
9  int updown_services(int mode, char *sys_service){
10     if(mode) return func2(sys_service);
11     return func3(sys_service);
12 }
13 int func2(char *a1){
14     char buf[1028];
15     if (a1 && *a1){
16         strcpy(buf, a1);
17     }
18 }
```

Step into the `updown_service` function

Efficient Dataflow Analysis

- Lightweight, context-sensitive, on-demand, inter-procedural analysis

```
1  // ssi
2  void setup_wizard_mydlink(int a1){
3      char *v4;
4      v4 = getenv("sys_service");
5      updown_services(0, v4);
6      post2nvram(a1);
7      response_page = get_response_page();
8  }
9  int updown_services(int mode, char *sys_service){
10     if(mode) return func2(sys_service);
11     return func3(sys_service);
12 }
13 int func2(char *a1){
14     char buf[1028];
15     if (a1 && *a1){
16         strcpy(buf, a1);
17     }
18 }
```

Skip these two functions

Efficient Dataflow Analysis

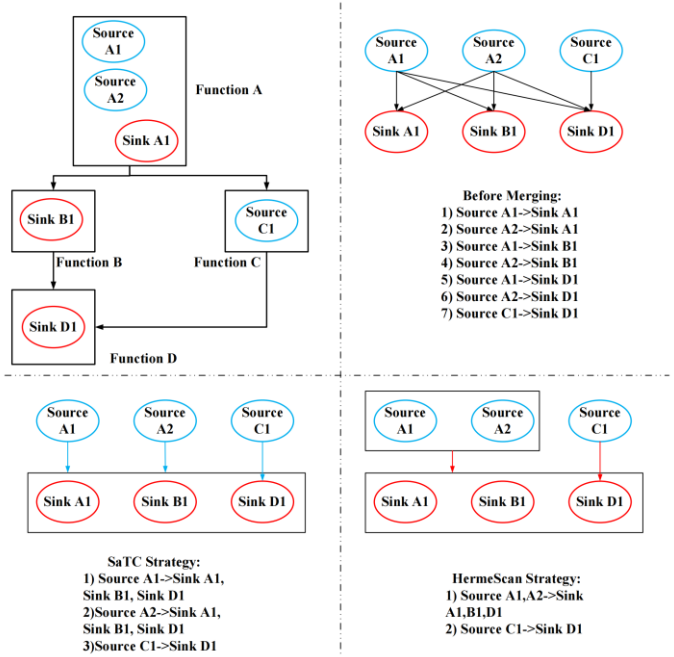
- Lightweight, context-sensitive, on-demand, inter-procedural analysis

```
1  // ssi
2  void setup_wizard_mydlink(int a1){
3      char *v4;
4      v4 = getenv("sys_service");
5      updown_services(0, v4);
6      post2nvram(a1);
7      response_page = get_response_page();
8  }
9  int updown_services(int mode, char *sys_service){
10     if(mode) return func2(sys_service);
11     return func3(sys_service);
12 }
13 int func2(char *a1){
14     char buf[1028];
15     if (a1 && *a1){
16         strcpy(buf, a1);
17     }
18 }
```

Apply function summary

Efficient Dataflow Analysis

- Path merging strategy
  - Leverages the path-insensitive feature of RDA
  - Multi-source taint:
    taint each source point with a different label
    when a function contains multiple source points
  - Multi-sink observation:
    all sinks in a reachability call graph can be
    observed in one pass of RDA analysis



Evaluation

- Q1: How well does HermeScan **find vulnerabilities on real-world devices**? How effective is it **compared to state of-the-art tools**?
- Q2: How does the optimization of **control flow recovery** contribute to the vulnerability detection of HermeScan?
- Q3: Can HermeScan's **input source identification** make the analysis more accurate? How does it work?
- Q4: Can HermeScan's **path merging strategy** alleviate the path explosion problem?

Dataset

- 0-day dataset
  - 30 samples
  - 8 vendors and 19 series
  - architecture: ARM32, ARM64, MIPSEL, and MIPSEB
- N-day dataset
  - 98 samples
  - 25 series from 9 popular IoT vendors
  - contains the data sets of SaTC and KARONTE

Q1: Comparative Evaluation

- **Overview**

  HemreScan raise 297 alerts with 156 vulnerabilities

- **Effectiveness**

  HermeScan reports 120 more vulnerabilities than SaTC, and 152 more vulnerabilities than KARONTE

- **Accuracy**

  HermeScan outperforms SaTC in TPR by 39%

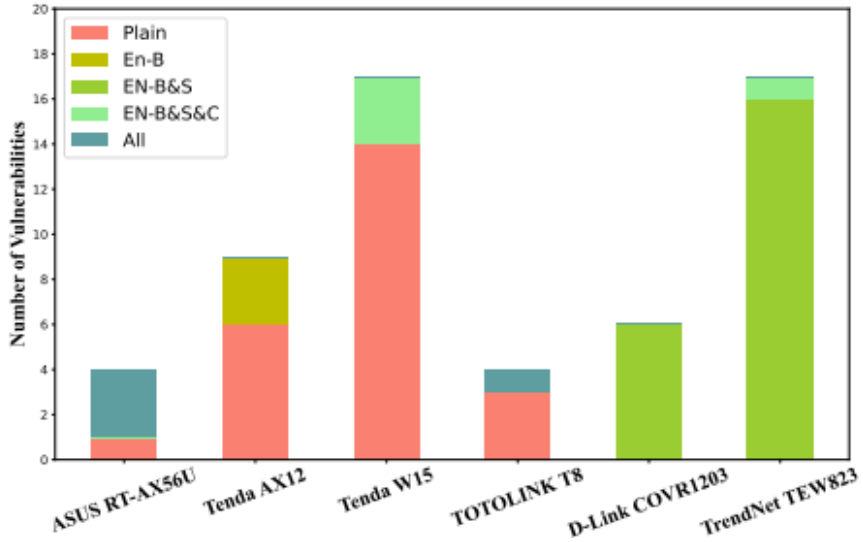
- **Efficiency**

  HermeScan is 7.5x times faster than SaTC and 3.8x times faster than KARONTE

		HermeScan				SaTC			Karnote		
Vendor&Model	**Program Name**	**Alerts**	**Vuls (bof+ci)**	**Vuls (other)**	**Time**	**Alerts**	**Vuls (bof+cli)**	**Time**	**Alerts**	**Vuls (bof+cli)**	**Time**
LINKSYS MR7350	bluetoothd	0	0	0	3min	1	0	30min	0	0	2h22min
LINKSYS E9450	httpd	0	0	0	11min	0	0	24min	0	0	2h08min
LINKSYS EA4500	twonkymediaserver	1	**1**	0	17min	0	0	26h	0	0	1h21min
ASUS GT-AX6000	httpd	0	0	0	9min	5	0	27h28min	0	0	4h31min
ASUS GT-AC2900	cfg-server	0	0	0	12min	2	0	23h58min	0	0	*7min*
ASUS RT-AX56U	httpd	4	**4**	0	13min	5	0	26h21min	0	0	48min
Tenda AX-12	httpd	9	**6**	3	1h20min	0	0	*12min*	0	0	3h36min
Tenda AX-3	httpd	17	**11**	0	2h23min	27	6	36h	0	0	1h24min
Tenda AX-1803	thttpd	20	**12**	2	2h03min	38	8	16h28min	0	0	*5min*
Tenda AX-1806	thttpd	20	**14**	0	2h11min	44	13	18h53min	0	0	42min
Tenda W15E	httpd	17	**15**	2	2h39min	50	5	21h11min	0	0	1h22min
TOTOLINK T8	cstecgi	14	**4**	0	14min	0	0	*3min*	0	0	*2min*
TOTOLINK LR350	cstecgi	24	**9**	0	13min	0	0	47min	0	0	24min
TOTOLINK A7000	cstecgi	18	**12**	0	12min	0	0	4h09min	0	0	39min
TOTOLINK A8000	cstecgi	29	**13**	0	13min	2	0	39min	0	0	6h04min
D-LINK COVR-1201	prog.cgi	9	**4**	2	5h27min	0	0	*10min*	0	0	4h42min
D-LINK COVR-1210	prog.cgi	8	**4**	2	5h16min	0	0	*10min*	0	0	4h28min
Netgear RAX-10	net-cgi	6	0	0	19min	0	0	49min	0	0	2h20min
Netgear RAX-30	ntgr_ra_iot	0	0	0	6min	8	0	5h54min	0	0	3h10min
Netgear RAX-120	net-cgi	1	0	0	37min	0	0	1h09min	0	0	72h
Netgear MR-62	httpd	1	**1**	0	51min	0	0	23h56min	4	0	2h56min
Trendnet twe 829	samba_multicall	1	**1**	0	30min	0	0	*2min*	0	0	1h57min
Trendnet tew 823	ssi	39	**17**	0	9min	0	0	*11min*	0	0	1h54min
Trendnet tew 827	ssi	31	**18**	2	18min	0	0	*27min*	0	0	59min
Trendnet tew 818	rc	12	**5**	0	14min	0	0	18h32min	0	0	2h14min
Trendnet tew 752	cgibin	5	**1**	0	9min	0	0	20min	0	0	2h10min
TP-LINK AX3000	fapi_wlan_cli	0	0	0	*0min*	0	0	*15min*	0	0	2h42min
TP-LINK XDR1850	dms	2	0	0	18min	0	0	*3min*	0	0	*2min*
TP-LINK XDR3060	dms	3	**2**	0	1h44min	0	0	*2min*	0	0	28min
TP-LINK XTR7880	dms	6	**2**	0	2h10min	0	0	*3min*	0	0	19min
Total	/	297	**156(152)**	13(11)	30h4min	182	32(32)	252h19min	4	0	127h58min
Average	/	9.9	5.2	/	1h7min	6.66	1.9	8h25min	0.13	0	4h16min

Q2: Effectiveness of enhanced CFG

- All optimization techniques used to enhance control flow graph construction contribute to the vulnerability detection ability of HermeScan



B	Function Boundary identification
S	Symbol name recovery
C	Shared library CFG included

Q3: Effectiveness of Input Source Identification

- Candidate source function checking reduces the FPs of vulnerabilities by 18% on the zero-day dataset
- Fuzzy matching strategy can find an additional 27% of keywords
- Input source identification effectively helps HermeScan reduce false positives and false negatives

Vendor & Model	Shared Keywords(S)	Shared Keywords(H)	Increased Proportion
LINKSYS MR7350	47	52	10.64%
LINKSYS E9450	56	57	1.79%
LINKSYS EA4500	65	66	1.54%
ASUS GT-AX6000	180	187	3.89%
ASUS GT-AC2900	180	184	2.22%
ASUS RT-AX56U	404	504	24.75%
Tenda AX-12	201	222	10.45%
Tenda AX-3	246	253	2.85%
Tenda AX-1803	254	255	0.39%
Tenda AX-1806	262	269	2.67%
Tenda W15E	437	535	22.43%
TOTOLINK T8	67	69	2.99%
TOTOLINK LR350	66	67	1.52%
TOTOLINK A7000	79	80	1.27%
TOTOLINK A8000	107	116	8.41%
D-LINK COVR-1201	506	625	23.52%
D-LINK COVR-1210	495	618	24.85%
Netgear RAX-10	860	897	4.30%
Netgear RAX-30	107	237	121.50%
Netgear RAX-120	866	1005	16.05%
Netgear MR-62	862	870	0.93%
Trendnet TEW-829	35	35	0.00%
Trendnet TEW-823	1042	1459	40.02%
Trendnet TEW-827	103	365	254.37%
Trendnet TEW-818	176	249	41.48%
Trendnet TEW-752	36	36	0.00%
TP-LINK AX3000	237	238	0.42%
TP-LINK XDR1850	99	188	89.90%
TP-LINK XDR3060	96	117	21.88%
TP-LINK XTR7880	108	213	97.22%
Average	276	336	27.81%

Q4: Effectiveness of Path Merging Strategy

- The path merging strategy reduces the number of paths by 89.4% on average
- 22 out of 30 samples merged more than 90% of the paths

Vendor & Model	Paths(BF)	Paths(AF)	Decreased Proportion
LINKSYS MR7350	69	23	66.67%
LINKSYS E9450	472	121	74.36%
LINKSYS EA4500	1838	143	92.22%
ASUS GT-AX6000	1010	63	93.76%
ASUS GT-AC2900	1062	88	91.71%
ASUS RT-AX56U	636	129	79.72%
Tenda AX-12	1673	72	95.70%
Tenda AX-3	9413	96	98.98%
Tenda AX-1803	5112	109	97.87%
Tenda AX-1806	4789	101	97.89%
Tenda W15E	11113	186	98.33%
TOTOLINK T8	7126	101	98.58%
TOTOLINK LR350	12688	75	99.41%
TOTOLINK A7000	13944	77	99.45%
TOTOLINK A8000	610	105	82.79%
D-LINK COVR-1203	686	19	97.23%
D-LINK COVR-1210	644	18	97.20%
Netgear RAX-10	1299	57	95.61%
Netgear RAX-30	171	11	93.57%
Netgear RAX-120	910	345	62.09%
Netgear MR-62	2008	214	89.34%
Trendnet TEW-829	231	32	86.15%
Trendnet TEW-823	167554	265	99.84%
Trendnet TEW-827	12160	116	99.05%
Trendnet TEW-818	20014	219	98.91%
Trendnet TEW-752	594	9	98.48%
TP-LINK AX3000	0	0	0.00%
TP-LINK XDR1850	2294	17	99.26%
TP-LINK XDR3060	2747	18	99.34%
TP-LINK XTR7880	2689	18	99.33%
Average	9518	95	89.40%

Summary

- We present a **lightweight reaching definition analysis solution** HermeScan to perform taint analysis on IoT firmware binaries

- HermeScan has discovered **87 zero-day vulnerabilities** in real-world devices, and **69** of them have been assigned **CVE IDs**

- We build **two sets of firmware samples** and comprehensively evaluate the performance of existing tools

Thanks! Questions?