Leveraging Semantic Relations in Code and Data to Enhance Taint Analysis of Embedded Systems

Jiaxu Zhao, Yuekang Li, Yanyan Zou*, Zhaohui Liang, Yang Xiao, Yeting Li
Bingwei Peng, Nanyu Zhong, Xinyi Wang, Wei Wang, Wei Huo*

Embedded Systems

- Embedded systems are typically installed on different types of IoT devices and network devices, the number of devices in use is expected to reach 27.1 billion by 2025[1].
- According to recent statistics[2], weekly attacks on IoT devices have increased by 41% per organization in the first two months of 2023 compared to 2022.

IPCamera | Firewall | VPN | Router | Switch | NAS

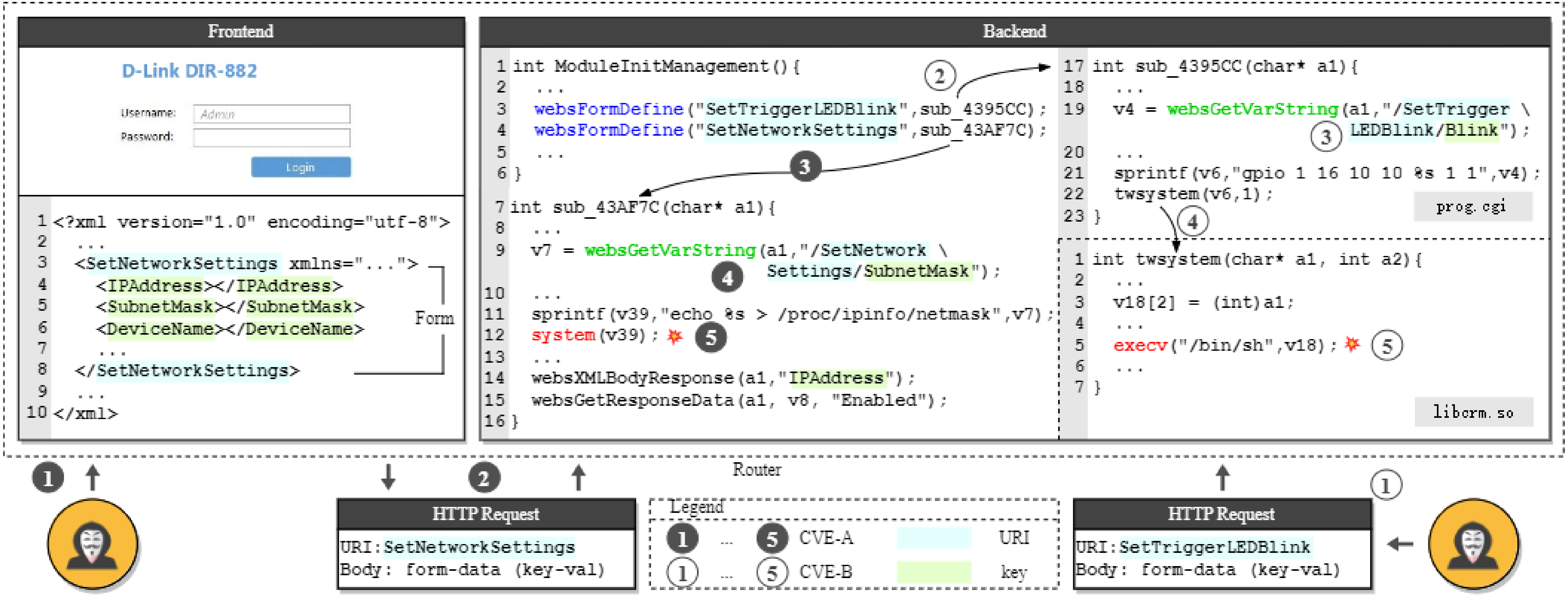
[1] https://iot-analytics.com/number-connected-iot-devices/

[2] https://blog.checkpoint.com/security/the-tipping-point-exploring-the-surge-in-iot-cyberattacks-plaguing-the-education-sector/

Existing Methods

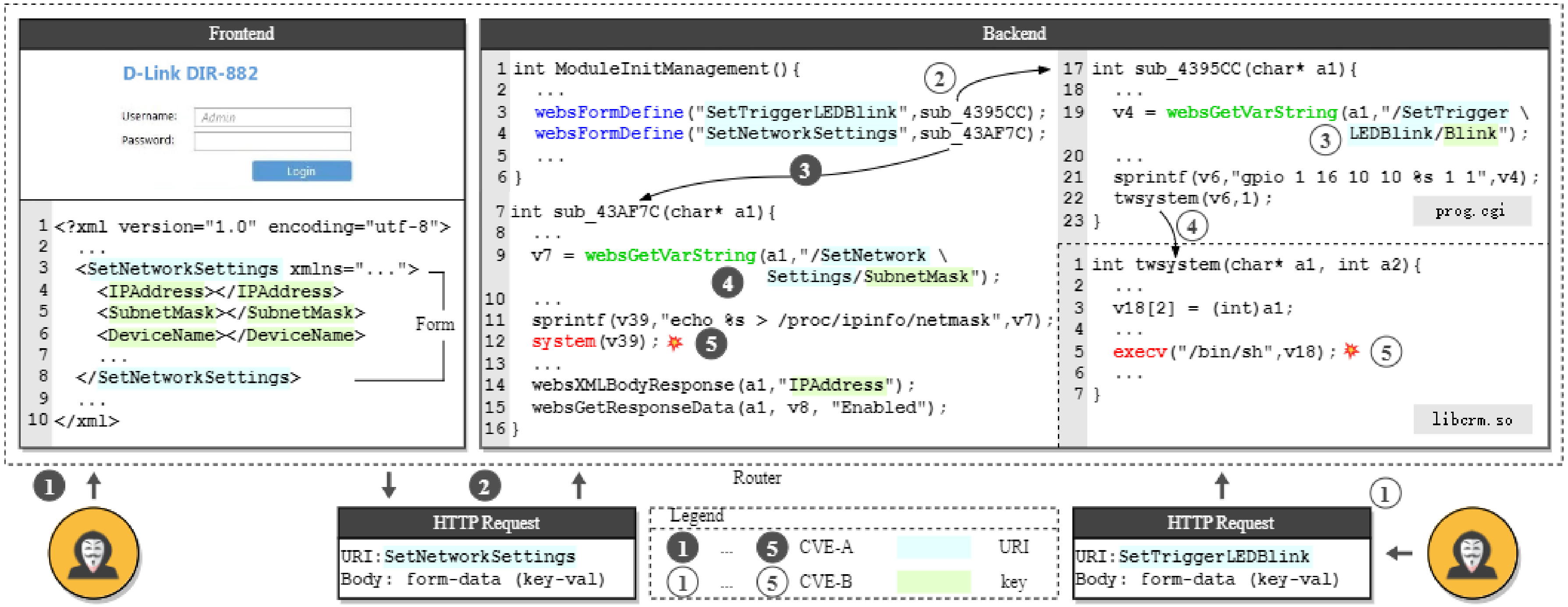
- Dynamic solutions
  - Black-box fuzzing and fuzzing based on emulator
  - Limited by code coverage and simulation success rates
  - Only focus on memory-related vulnerabilities
- Static methods
  - Taint analysis and Symbolic Execution
  - Static analysis is more applicable, but the problem of false positives and false negatives still needs to be solved!!!

Motivating Example



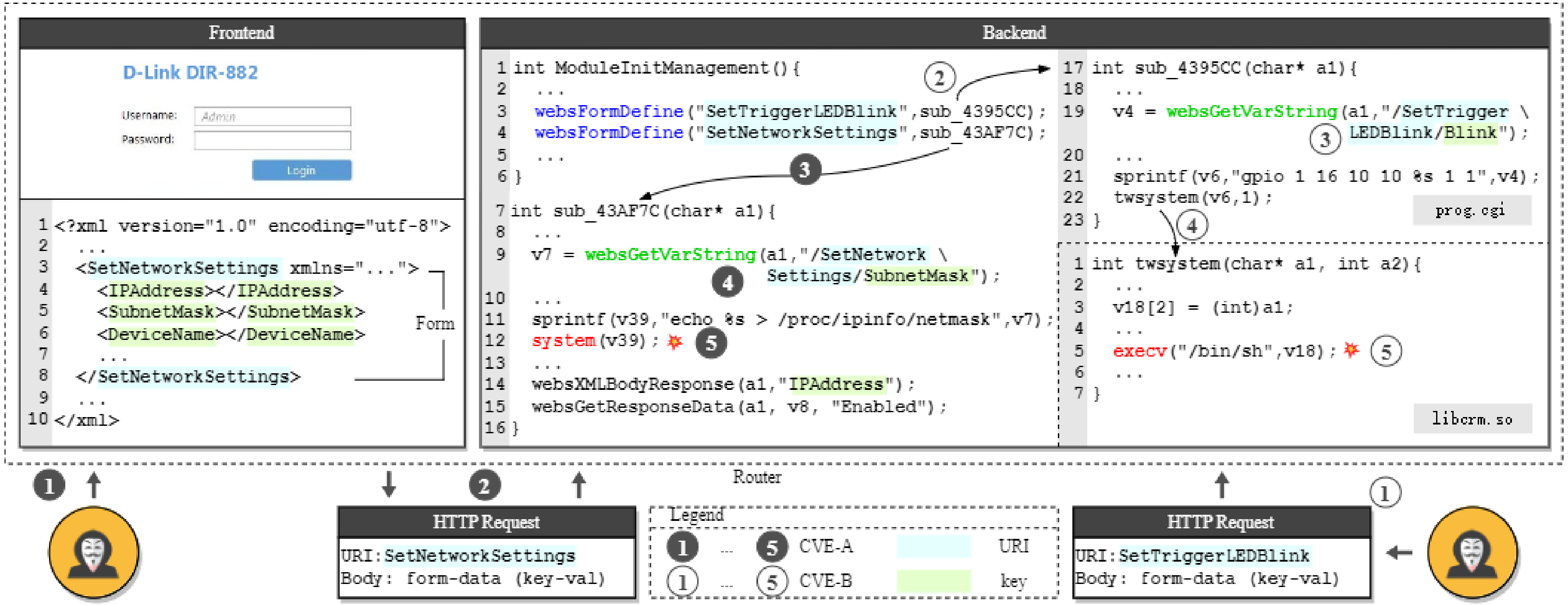
Motivating Example

- SaTC (sec21) fails to find the sources for both CVE-A and CVE-B
  - CVE-A: incomplete redefined rules for extracting source from the frontend
  - CVE-B: lack of support for extracting hidden data



Observation

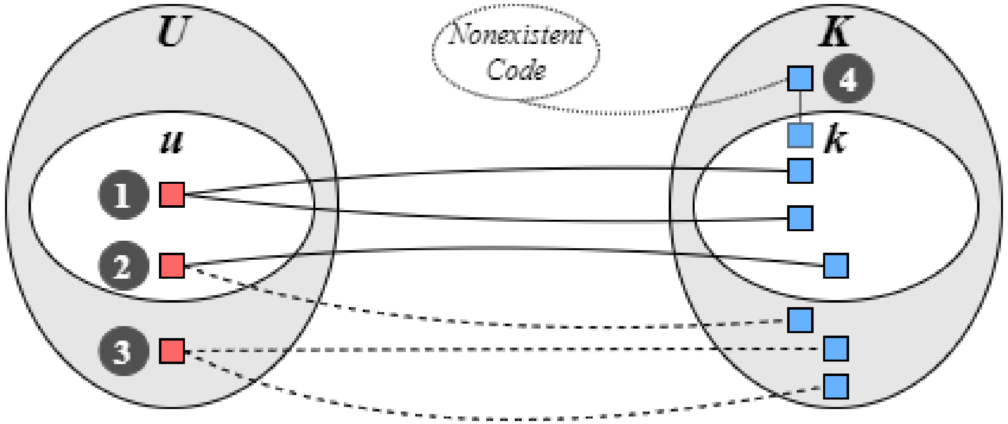
- User input entries can be categorized as URIs or keys.
- Some URIs and keys are non-hidden and some are hidden.
- Different URIs and keys shares same handing codes.



Findings

- Finding 1: User input entries can be categorized as URIs or keys. Identifying their corresponding handling codes and relationships can help to identify both hidden and non-hidden user input entries and locate the taint sources.

- relationships between URIs and keys
    - Non-hidden URIs and non-hidden keys
    - Non-hidden URIs and hidden keys
    - hidden URIs and hidden keys
    - the keys do not have any related URIs



Findings

- Finding 1: User input entries can be categorized as URIs or keys. Identifying their corresponding handling codes and relationships can help to identify both hidden and non-hidden user input entries and locate the taint sources.

- Challenge 1: the key problem in converting it into methodology is how to identify the backend URI and key handling codes.

- Intuitively, we can conclude some patterns to identify these codes to address this challenge.
- Some non-hidden URIs and keys —> corresponding handling codes —> other hidden and non-hidden URIs and keys

Findings

- the semantic information in the backend code can facilitate more precise pattern-based static analysis, such as inferring the purpose of a function.
  - key *time* —> key handing function *websGetVar* ✓
  - key *time* —> key handing function *strncmp* ✗

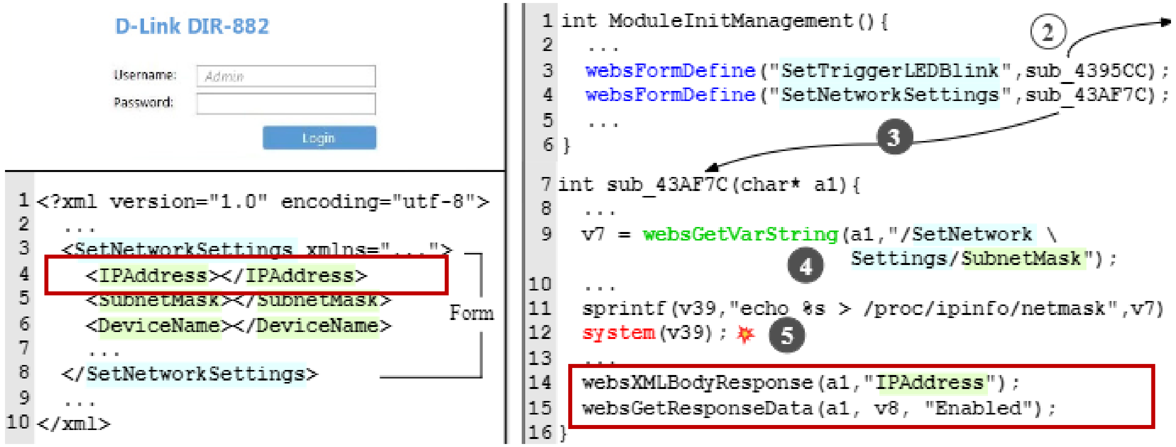
```
// non-hidden key
time_interval = websGetVar(wp,"time","00:00-06:30");
// hidden key
close_type = websGetVar(wp,"ledCloseType","allclose");
if(!strncmp(old_sched_led_type,"time",4)&&!strncmp(sched_led_type,
   "close",5)){
```

- Challenge 2: Effectively perform semantic-based analysis and combine it with pattern-based analysis is another challenge.

Findings

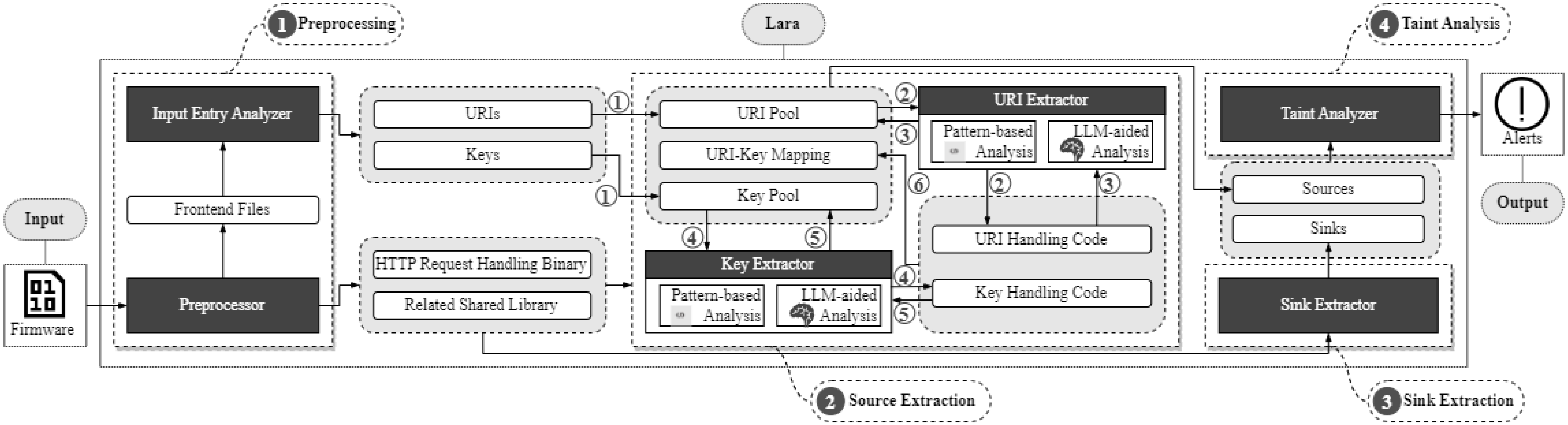
- Finding 2: Due to the better understanding of code semantics provided by LLM and the differences in false positive sources between LLM-aided analysis and pattern-based static analysis, LLM-aided analysis effectively enhances the identification of more accurate sources.
- False positives in pattern-based static analysis are caused by code patterns, while false positives in LLM-aided analysis are due to misleading code semantics, such as symbolic data.

- Pattern-based FP: websXMLBodyResponse
- LLM-aided FP: websGetResponseData



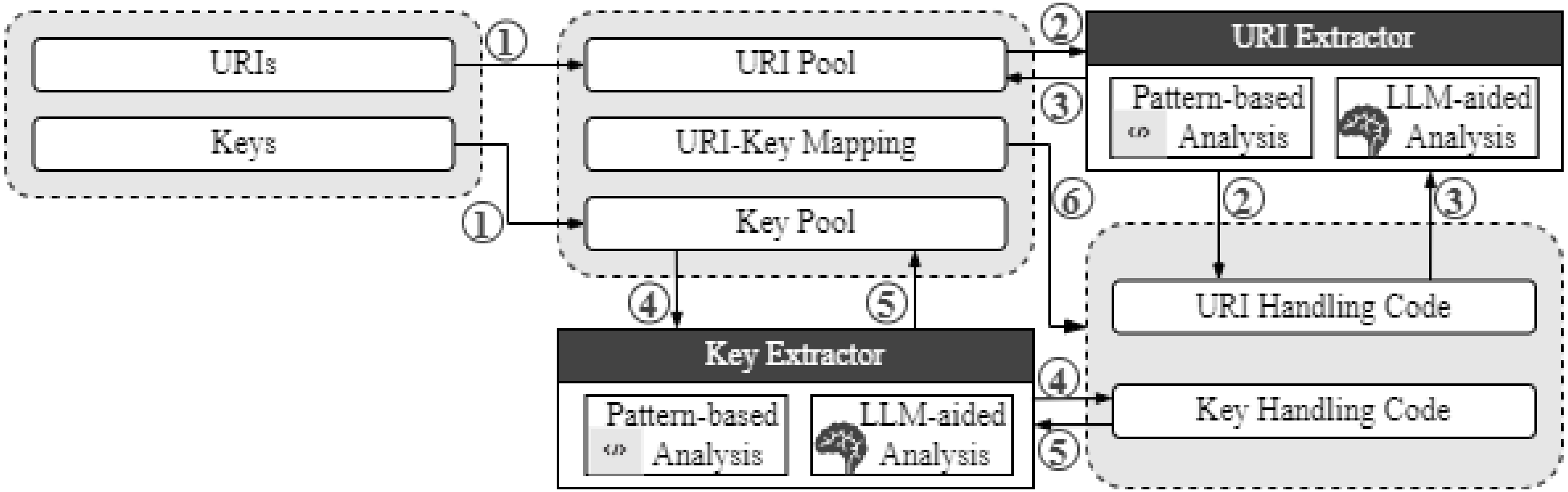
Lara | Architecture

- **Source Extraction**: Pattern-based Static Analysis && LLM-aided Analysis
- **Sink Extraction**
- **Taint Analysis**



Lara | Source Extraction

- ①-②: use the URIs extracted from the frontend to identify the URI handling code
- ③: enrich the URI pool with URIs not included or found in the front-end
- ①-④: uses the keys extracted from the frontend to identify the key handling code
- ⑤: enrich the key pool with keys not included or found in the front-end
- ⑥: uses the mapping between URIs and keys to filter out unreachable key handling code



Lara | Pattern-based Static Analysis

- We studied the backend programs of mainstream firmware and found that the code to bind URIs with URI handling codes can be categorized into three types.
- URI —> URI binding code —> URIs and URI handing functions

URI Binding Code Type 1 - Registration Function

```
int formDefineFirewall(){
  websFormDefine("BasicSettings",sub_44FFD0);
  websFormDefine("portForward",sub_44EFF0);
  websAspDefine("checkBridgeModeASP",sub_448F14);
}
```

URI Binding Code Type 2 - Constant Data

```
int HandleSocket(){
  for(j=0;;++j){
    v3 = ($0x15AD80)[2*j];
    ...
  }
}
.data:0x15AD80 DCD aGetStr // URI:getstr
.data:0x15AD84 DCD sub_AB16C // handling func
...
.data:0x15BDD8 DCD awifiCgi // URI:wifi.cgi
.data:0x15BDDC DCD sub_82A64 // handling func
...
.data:0x15C144 ALIGN 0x10
```

URI Binding Code Type 3 - Process Creation

```
int websCgiHandler(char *a1){
  cgiName = websGetCgiName(a1); // cigName:uri.cgi
  v36 = websLauchCgi(...,cgiName,data,...);
}
int websLaunchCgi(...){
  v24 = fork();
  if(!v24){
    if(execve(cgiName,a2,...)!=-1)
      ...
}
```

Lara | Pattern-based Static Analysis

- When the program parses the HTTP request body, the function that handles these data extracts the value corresponding to the key.
- To make sure the extracted key is associated with the HTTP request, Lara checks if the parameters of the key handling function and the URI handling function have an intersection.

- key —> key handing function

  —> corresponding keys for every URI

Key Handling Code

```
int formSetSchedLed(wp,path,query){
  // non-hidden key
  time_interval = websGetVar(wp,"time","00:00-06:30");
  // hidden key
  close_type = websGetVar(wp,"ledCloseType","allclose");
  if(!strncmp(old_sched_led_type,"time",4)&&!strncmp(sched_led_type,
     "close",5)){
    ResponseResultString = GetResponseResultString(v6);
    ...
    respCgiSendResp(v6,v7,"PWD_password",query);
  }
  get_mib_2cJSONString(root,"config","w15g.bss.wps.basic_have_config"
                       ,&wifi_buf_entry);
  ...
  return sub_1BB34("PWD_password",query);
}
```

Pattern-based Analysis → {websGetVar,respCgiSendResp,strncmp,sub_1BB34}

Lara | LLM-aided Analysis

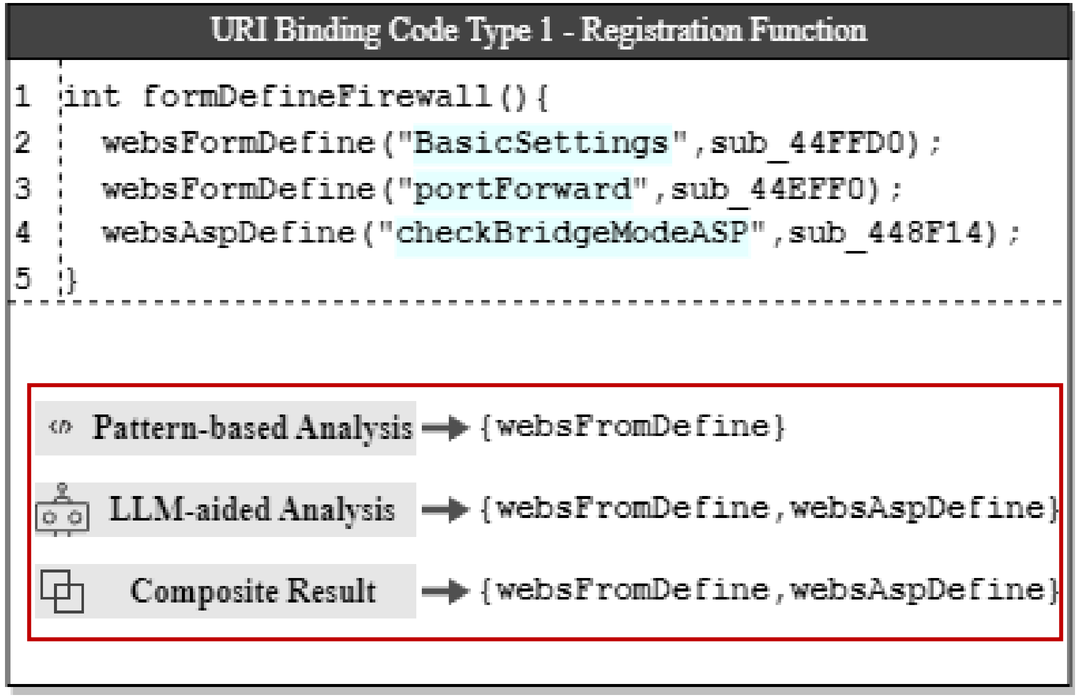
- We designed corresponding LLM interaction models for extracting URIs and keys based on the LMQL(PLDI2023), enabling LLM-aided analysis to produce valid results without manual interactions.
- Prompt Design
- Merge Operation

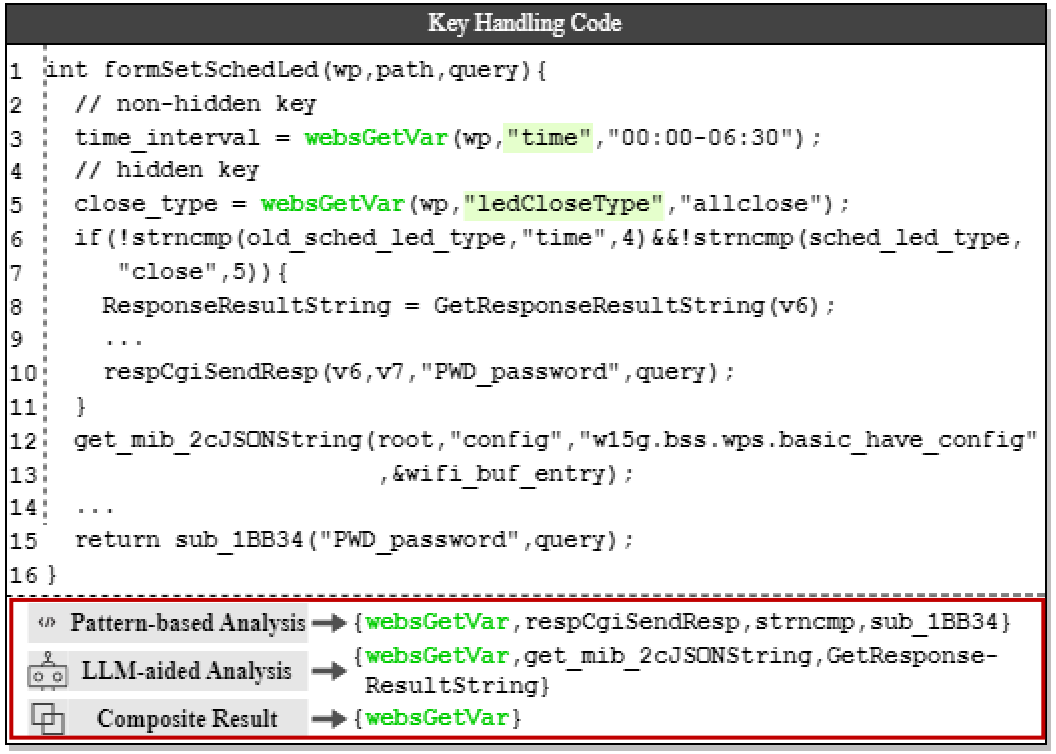
LLM interaction model for extracting URI Registration Function

```
query:
 [CoT Prompts]
  f = Locate_function(URI_reference_address)
 "Analyze the function f pseudocode and identify each called function"
 funcs = []
 "[FUNCs]"
 funcs += (FUNCs.list())
 result = []
 for func in funcs:
  "Whether the function func is with registration functionality?"
  "[A2]"
  if A2 == "YES":
   result.append(func)
from "GPT-3"
where
 FUNCs over OPTIONS.split(",") and
 STOPS_AT(A2, "YES")OR STOPS_AT(A2, "NO")
```

Lara | LLM-aided Analysis

- LLM-aided analysis can identify more URI registration functions for URI binding code type I.
- LLM contributes more on reducing FPs when extracting key handling functions.





Lara | Sink Extraction

- **Function-call sinks**
  - standard library function, like strcpy()、system().
  - wrapper function, like save_encrypted_data() from shared libraries wraps popen().
- **Non-function-call sinks**
  - direct variable operations, which may affect the contents of variable values, the positions of array reads, the number of controlled loop iterations, and so on.

```
int save_encrypted_data(char *a1, char *a2){
    memset(s, 0, 0x200);
    snprintf(s, 0x200, "echo -n %s | openssl ... -out %s", a1, a2);
    return popen(s, "r");
}
```

Lara | Taint Analysis

- **Source**: taint source is the variable to which the value corresponding to the key is assigned. This variable can be an argument or a return value of the function.
- **Taint Analysis:** Taint analysis begins at the URI handling function where the processing of the HTTP request body begins. And the taint analysis engine is based on IDAPython, including intra-function and inter-function propagation.
- **Vulnerability Detection:** Non-function-call sink models and dangerous standard library function sink models are predefined. Wrapper function models are extended based on the results of wrapper function extraction. Lara checks constraints when tainted data reaches a sink.

Evaluation

- **RQ1**: How is the performance of Lara comparing with the state-of-the-art tools?
- **RQ2**: How effective each part of Lara is for discovering vulnerabilities?
- **RQ3**: Can Lara discover previously unknown real-world vulnerabilities in firmware?

Evaluation | Dataset

- **Firmware Dataset:**

The dataset comprises 203 firmware samples from 21 vendors, including 10 different device types, covering 80 Routers, 37 APs, 20 Switches, 19 IPCameras, 17 Firewalls, 11 Range Extenders, 6 VPNs, 6 Modems, 4 Bridges, 3 NAS.

- **Known Vulnerability Dataset**

To have a fair ground truth, we collected all known buffer overflow vulnerabilities and command injection vulnerabilities that exist in these firmware samples from CVE records, while filtering out vulnerabilities without detailed information. we totally collected 646 known vulnerabilities.

Evaluation | Baselines

- Karonte focuses on vulnerabilities caused by interactions between multiple binaries.
- SaTC uses sources extracted from the frontend, and sinks are predefined functions.
- EmTaint utilizes on-demand alias analysis to enhance taint tracking.
- various variants of Lara.

Experiment Mode	Key-sensitive Taint Analysis	Key-pattern Analysis	URI-pattern Analysis	LLM-aided Analysis	Sink Extraction
LARA-Sink	✔	✗	✗	✗	✔
LARA-Key	✔	✔	✗	✗	✗
LARA-Pattern	✔	✔	✔	✗	✗
LARA-LLM	✔	✗	✗	✔	✗
LARA-Combined	✔	✔	✔	✔	✗
LARA	✔	✔	✔	✔	✔

RQ1: Comparison with the SOTA tools

- Compared with SaTC, Lara detected 556 more vulnerabilities and incurred 65.5% less time overhead.
- Compared with Karonte, Lara detected 602 more vulnerabilities and incurred 63.1% less time overhead.

Vendor	Samples	URI Type	#Vuln	LARA						SATC						KARONTE					
				TP	FP	FN	Prec.	Reca.	Time	TP	FP	FN	Prec.	Reca.	Time	TP	FP	FN	Prec.	Reca.	Time
Tenda	19	I&II	233 (51)	231 (51)	28	2	89.2%	99.1%	4,306	64 (1)	89	169	41.8%	27.5%	8,903	12	23	221	34.3%	5.2%	9,120
TOTOLink	11	II&III	134 (13)	134 (13)	32	0	80.7%	100.0%	736	8	23	126	25.8%	6.0%	5,280	0	5	134	0.0%	0.0%	4,667
H3C	13	I&I&III	88	88	9	0	90.7%	100.0%	894	0	22	88	0.0%	0.0%	6,240	0	11	88	0.0%	0.0%	6,240
DLink	14	I&II&III	67 (11)	62 (11)	13	5	82.7%	92.5%	1,603	0	42	67	0.0%	0.0%	6,720	8	22	59	26.7%	11.9%	5,903
TRENDnet	13	I&II&III	23 (2)	23 (2)	11	0	67.6%	100.0%	1,103	3	10	20	23.1%	13.0%	5,080	0	3	23	0.0%	0.0%	5,914
Wavlink	3	III	21	21	9	0	70.0%	100.0%	96	1	6	20	14.3%	4.8%	903	0	2	21	0.0%	0.0%	1,003
NetGear	34	I&II&III	16 (1)	16 (1)	15	0	51.6%	100.0%	8,799	1	11	15	8.3%	6.3%	16,011	11	15	5	42.3%	68.8%	14,080
Cisco	4	II&III	10	10	8	0	55.6%	100.0%	153	1	16	9	5.9%	10.0%	603	3	6	7	33.3%	30.0%	667
Linksys	13	I&II&III	9 (3)	9 (3)	0	0	100.0%	100.0%	768	0	15	9	0.0%	0.0%	2,301	0	3	9	0.0%	0.0%	3,005
DrayTek	9	I&II&III	9	9	5	0	64.3%	100.0%	1,203	0	0	9	0.0%	0.0%	3,908	0	0	9	0.0%	0.0%	4,320
TPLink	8	I	9 (1)	8 (0)	3	1	72.7%	88.9%	803	1	0	8	100.0%	11.1%	1,608	0	0	9	0.0%	0.0%	1,303
Zavio	1	III	9	9	1	0	90.0%	100.0%	82	0	2	9	0.0%	0.0%	480	0	0	9	0.0%	0.0%	480
Motorola	2	I	6 (2)	6 (2)	2	0	75.0%	100.0%	147	0	3	6	0.0%	0.0%	960	0	1	6	0.0%	0.0%	960
Belkin	3	II&III	5	5	0	0	100.0%	100.0%	283	3	4	2	42.9%	60.0%	803	2	1	3	66.7%	40.0%	966
SonicWall	2	II&III	4	4	1	0	80.0%	100.0%	182	0	0	4	0.0%	0.0%	960	0	0	4	0.0%	0.0%	960
ASUS	18	I&II	1	1	0	0	100.0%	100.0%	2,108	0	3	1	0.0%	0.0%	5,166	0	3	1	0.0%	0.0%	3,806
QNAP	5	III	1	1	0	0	100.0%	100.0%	603	0	3	1	0.0%	0.0%	1,608	0	0	1	0.0%	0.0%	1,303
Fortinet	1	II	1	1	2	0	33.3%	100.0%	20	0	0	0	0.0%	0.0%	480	0	0	0	0.0%	0.0%	480
Zyxel	25	I&III	0	0	0	0	0.0%	0.0%	3,260	0	0	0	0.0%	0.0%	9,605	0	0	0	0.0%	0.0%	7,221
Mercury	3	I&III	0	0	3	0	0.0%	0.0%	228	0	0	0	0.0%	0.0%	1,440	0	0	0	0.0%	0.0%	920
AXIS	2	III	0	0	0	0	0.0%	0.0%	51	0	0	0	0.0%	0.0%	452	0	0	0	0.0%	0.0%	960
Total	203	—	646 (84)	638 (83)	142	8	81.8%	98.8%	27,428	82 (1)	249	563	24.8%	12.7%	79,511	36	95	609	27.5%	5.6%	74,278

RQ1: Comparison with the SOTA tools

- Lara extracted a total of 27,781 URIs and 102,905 keys from the dataset with a precision of 99.8% and 99.7%, including 9,299 hidden URIs and 15,411 hidden keys.
- SaTC identified 5,201 URIs and 34,081 keys with a precision of 83.2% and 52.7%.

Tools	Vulnerability Detection				URI Registration Fun.			URI			Key Handling Fun.			Key		
	#Alert	TP	Prec.	Reca.	#	TP	Prec.	#	TP	Prec.	#	TP	Prec.	#	TP	Prec.
SATC	331	82	24.8%	12.7%	—	—	—	5,201	4,329	83.2%	—	—	—	34,081	17,959	52.7%
LARA-Sink	269	168	62.5%	26.0%	—	—	—	5,201	4,329	83.2%	—	—	—	34,081	17,959	52.7%
LARA-Key	782	498	63.7%	77.1%	—	—	—	—	—	—	1,032	383	37.1%	168,952	102,597	60.7%
LARA-Pattern	685	498	72.7%	77.1%	203	70	34.5%	24,969	24,969	100.0%	1,032	383	37.1%	155,664	102,433	65.8%
LARA-LLM	690	504	73.0%	78.0%	188	122	64.9%	27,823	27,714	99.6%	1,206	388	32.2%	133,266	102,611	77.0%
LARA-Combined	603	504	83.6%	78.0%	132	122	92.4%	27,781	27,714	99.8%	388	383	98.7%	102,905	102,597	99.7%
LARA	780	638	81.8%	98.8%	132	122	92.4%	27,781	27,714	99.8%	388	383	98.7%	102,905	102,597	99.7%

RQ1: Comparison with the SOTA tools

- With more sources and sinks from Lara, EmTaint could detect 245 more vulnerabilities.

Tools		TP	FP	FN	Prec.	Reca.
EMTAINT	Prototype	14	275	632	4.8%	2.2%
EMTAINT+LARA	EnhanceI	212	162	434	56.7%	32.8%
	EnhanceII	259	168	387	60.7%	40.1%
EMTAINT +SATC	EnhanceI	45	226	601	16.6%	7.0%
	EnhanceII	47	231	599	16.9%	7.3%

- FP Analysis for Lara: 105 false positives occurred due to LARA disregarding critical key operations between the source and sink, incorrect identification of sinks leads to 37 false positives.
- FN Analysis for Lara: 8 vulnerabilities were not detected due to either the complex inter-process communication (IPC) method or problems with the disassembly engines itself.

RQ2: Ablation Study

- Contribution of Source Extraction.
  - Lara-Combined (with the same sink) detected additional 422 vulnerabilities, thus achieving a 58.8% and 65.3% improvement in precision and recall, respectively.
  - This can be attributed to the fact that Lara-Combined extracted 23,385 more URIs and 84,638 more keys than SaTC while maintaining a higher precision rate.

Tools	Vulnerability Detection				URI Registration Fun.			URI			Key Handling Fun.			Key		
	#Alert	TP	Prec.	Reca.	#	TP	Prec.	#	TP	Prec.	#	TP	Prec.	#	TP	Prec.
SATC	331	82	24.8%	12.7%	—	—	—	5,201	4,329	83.2%	—	—	—	34,081	17,959	52.7%
LARA-Sink	269	168	62.5%	26.0%	—	—	—	5,201	4,329	83.2%	—	—	—	34,081	17,959	52.7%
LARA-Key	782	498	63.7%	77.1%	—	—	—	—	—	—	1,032	383	37.1%	168,952	102,597	60.7%
LARA-Pattern	685	498	72.7%	77.1%	203	70	34.5%	24,969	24,969	100.0%	1,032	383	37.1%	155,664	102,433	65.8%
LARA-LLM	690	504	73.0%	78.0%	188	122	64.9%	27,823	27,714	99.6%	1,206	388	32.2%	133,266	102,611	77.0%
LARA-Combined	603	504	83.6%	78.0%	132	122	92.4%	27,781	27,714	99.8%	388	383	98.7%	102,905	102,597	99.7%
LARA	780	638	81.8%	98.8%	132	122	92.4%	27,781	27,714	99.8%	388	383	98.7%	102,905	102,597	99.7%

RQ2: Ablation Study

- Contribution of LLM-aided Analysis
  - LLM-aided analysis contributes more on reducing the FPs than reducing the FNs.
  - Compared with Lara-Pattern, Lara-Combined reduces the FPR of vulnerability detection by 10.9%, the FPR of URI registration function identification by 57.9%, the FPR of key handling function identification by 61.6%, and the FPR of keys by 33.9%.
  - Meanwhile, Lara-Combined reduces the FNs by detecting 6 more vulnerabilities, 52 more URI registration functions, 2,745 more URIs, and 164 more keys.

Tools	Vulnerability Detection				URI Registration Fun.			URI			Key Handling Fun.			Key		
	#Alert	TP	Prec.	Reca.	#	TP	Prec.	#	TP	Prec.	#	TP	Prec.	#	TP	Prec.
SATC	331	82	24.8%	12.7%	—	—	—	5,201	4,329	83.2%	—	—	—	34,081	17,959	52.7%
LARA-Sink	269	168	62.5%	26.0%	—	—	—	5,201	4,329	83.2%	—	—	—	34,081	17,959	52.7%
LARA-Key	782	498	63.7%	77.1%	—	—	—	—	—	—	1,032	383	37.1%	168,952	102,597	60.7%
LARA-Pattern	685	498	72.7%	77.1%	203	70	34.5%	24,969	24,969	100.0%	1,032	383	37.1%	155,664	102,433	65.8%
LARA-LLM	690	504	73.0%	78.0%	188	122	64.9%	27,823	27,714	99.6%	1,206	388	32.2%	133,266	102,611	77.0%
LARA-Combined	603	504	83.6%	78.0%	132	122	92.4%	27,781	27,714	99.8%	388	383	98.7%	102,905	102,597	99.7%
LARA	780	638	81.8%	98.8%	132	122	92.4%	27,781	27,714	99.8%	388	383	98.7%	102,905	102,597	99.7%

RQ2: Ablation Study

- Contribution of Sink Extraction
  - Lara-Sink has identified 86 additional vulnerabilities in comparison to SaTC.
  - Lara has identified 134 more vulnerabilities compared to Lara-Combined.

Tools	Vulnerability Detection				URI Registration Fun.			URI			Key Handling Fun.			Key		
	#Alert	TP	Prec.	Reca.	#	TP	Prec.	#	TP	Prec.	#	TP	Prec.	#	TP	Prec.
SATC	331	82	24.8%	12.7%	—	—	—	5,201	4,329	83.2%	—	—	—	34,081	17,959	52.7%
LARA-Sink	269	168	62.5%	26.0%	—	—	—	5,201	4,329	83.2%	—	—	—	34,081	17,959	52.7%
LARA-Key	782	498	63.7%	77.1%	—	—	—	—	—	—	1,032	383	37.1%	168,952	102,597	60.7%
LARA-Pattern	685	498	72.7%	77.1%	203	70	34.5%	24,969	24,969	100.0%	1,032	383	37.1%	155,664	102,433	65.8%
LARA-LLM	690	504	73.0%	78.0%	188	122	64.9%	27,823	27,714	99.6%	1,206	388	32.2%	133,266	102,611	77.0%
LARA-Combined	603	504	83.6%	78.0%	132	122	92.4%	27,781	27,714	99.8%	388	383	98.7%	102,905	102,597	99.7%
LARA	780	638	81.8%	98.8%	132	122	92.4%	27,781	27,714	99.8%	388	383	98.7%	102,905	102,597	99.7%

RQ3: Real-world Vulnerabilities

- Lara uncovered 245 previously unknown vulnerabilities, and 162 of them have been assigned CVE IDs following responsible disclosure.
- 32 were due to hidden data, and 52 were caused by dangerous wrapper functions.

Vendor	#Series	#0-Day Vuln			#Hidden	#Wrapper Func.
		LARA	SATC	KARONTE		
Tenda	8	81	26	1	5	3
TOTOLInk	16	48	0	0	6	23
DLink	4	33	0	0	13	16
H3C	3	21	0	0	4	0
TRENDnet	5	15	7	4	1	3
Linksys	3	13	3	5	2	2
QNAP	3	12	0	0	0	0
Draytek	4	9	0	0	1	5
TPLink	4	4	0	0	0	0
ASUS	3	3	0	1	0	0
Cisco	2	3	0	0	0	0
Zavio	1	2	0	0	0	0
NetGear	1	1	0	0	0	0
Total	**57**	**245**	**36**	**11**	**32**	**52**

Summary

- Lara can capture sources with low false positive and false negative rates and thus can detect more vulnerabilities.

- Lara can significantly outperform the state-of-the-art IoT static analysis techniques by detecting more vulnerabilities with fewer false positives.

- We discovered 245 0-day vulnerabilities in 57 devices from 13 vendors, 162 of them have been assigned CVE IDs.

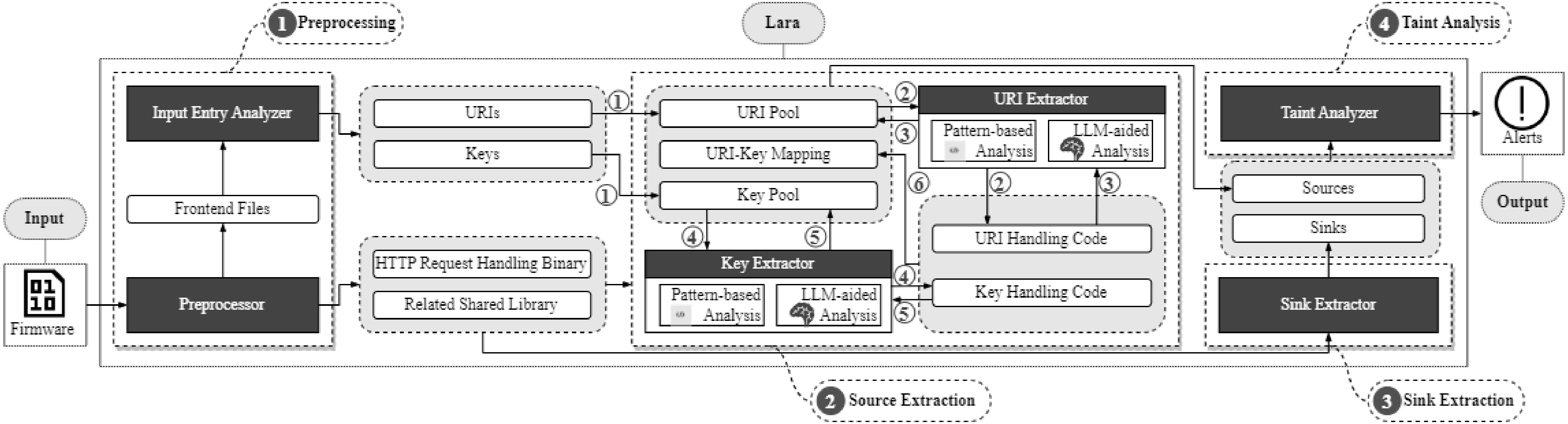
Code and Dataset:

https://sites.google.com/view/lara-data/

Q&A

Semantic Relations: Code and Code, Code and Data, Data and Data

More sources and more sinks make more vulnerabilities!



Thanks for your listening!

Contact me by email: **zhaojiaxu@iie.ac.cn**