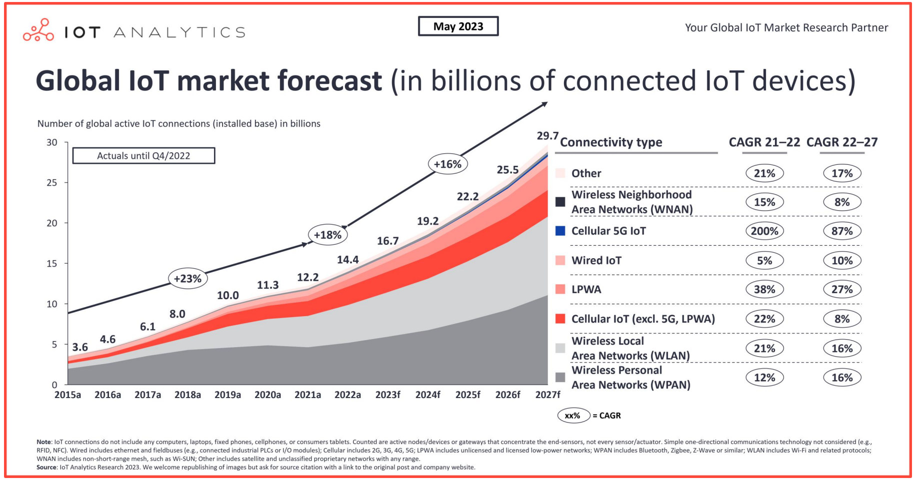
EAGLEYE: Exposing Hidden Web Interfaces in IoT Devices via Routing Analysis

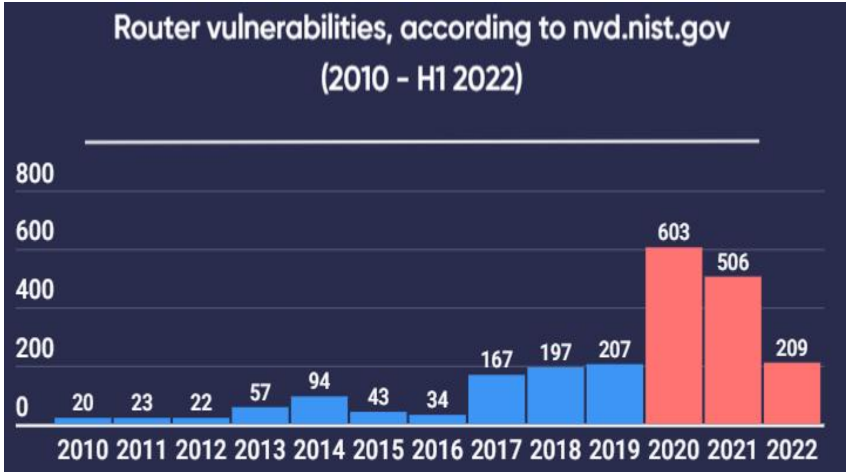
Hangtian Liu[*†§], Lei Zheng[†], Shuitao Gan[†§✉], Chao Zhang[†‡**], Zicong Gao[*],
Hongqi Zhang[*¶], Yishun Zeng[†], Zhiyuan Jiang[‖✉], Jiahai Yang[†]
[*]Information Engineering University [†]Institute for Network Sciences and Cyberspace (INSC), Tsinghua University
[‡]Zhongguancun Laboratory [§]Laboratory for Advanced Computing and Intelligence Engineering
[¶]Henan Key Laboratory of Information Security [‖]National University of Defense Technology
[**]JCSS, Tsinghua University (INSC) - Science City (Guangzhou) Digital Technology Group Co., Ltd.

Corresponding authors:ganshuitao@gmail.com, jzy@nudt.edu.cn.

IoT Devices Security

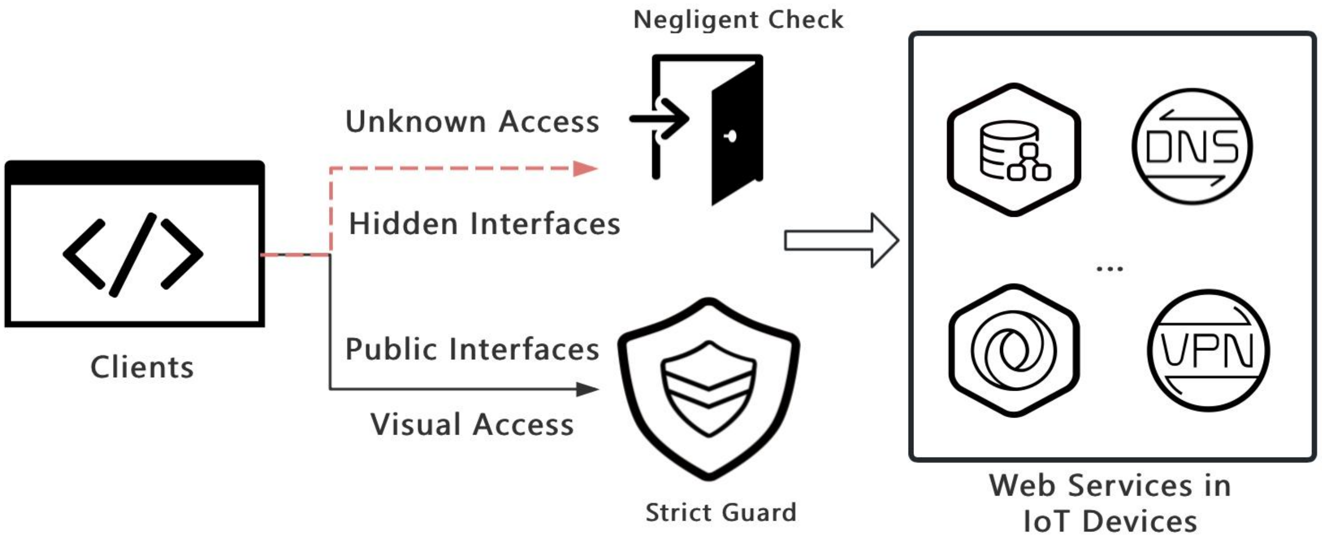
- The exponentially increase of IoT devices comes with growing threats.
  - Billions of IoT devices seamlessly connect humans, machines, and objects through network.
  - The vulnerabilities and attacks against IoT devices has grown significantly in recent years.





Hidden Interface

- The hidden interface issue is often overlooked due to its hidden nature.
- It leaves undisclosed access channels to attackers and is very likely to cause serious incidents.



What are hidden interfaces?

Definition

- Interface

  It refers to the gateway for clients to access specific functionalities or services of a device, which establishes the rules for client-device interaction, effectively serving as a mutual agreement on how to request particular functionalities or services.

- Public Interface

  Interfaces documented in the device manual are public interfaces. They are typically accessible through the entry portal.

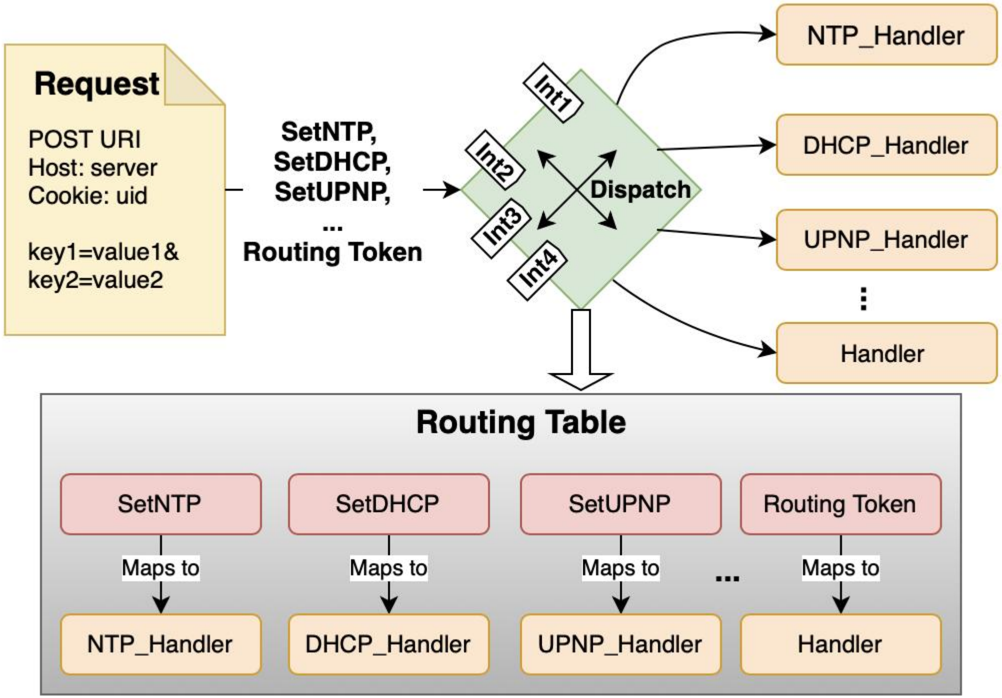
- Hidden Interface

  Interfaces not documented but still accessible to clients are termed hidden interfaces. They are not listed in the entry portal, preventing navigation to them.

Routing Mechanism

- **Routing token** is one special value in the request, which indicates the specific interface accessed and designates which handler should be called.

- **Routing table** reflects the mapping relationship between a routing token and the corresponding handler.



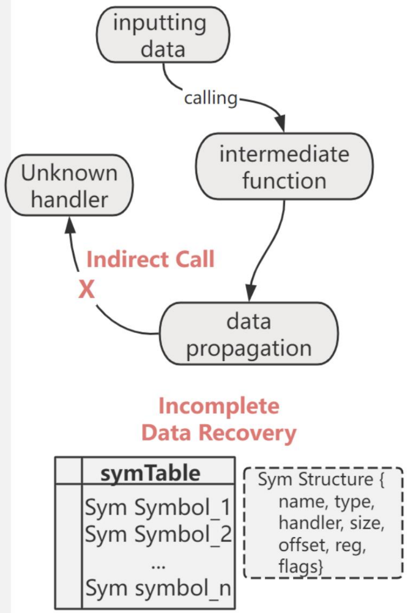
How to find hidden interfaces ?

How about Traditional Solutions?

- There is no obvious pattern of hidden interfaces, making it hard to statically search such interfaces (e.g., taint).

- There are no obvious consequences or feedback when the hidden interfaces are triggered, making it hard to dynamically test them (e.g., via fuzzing).

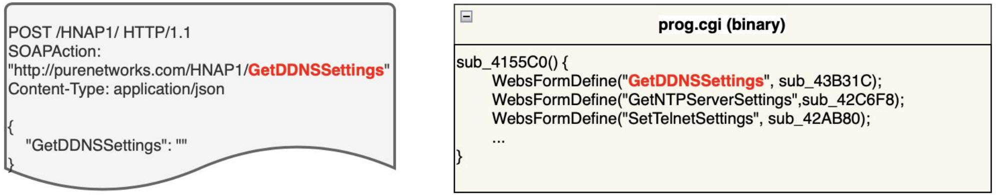
```
void callbackHandler(wp){
   // wp is the data structure of intermediate
representation for the  inputting request
   path=webGetVar(wp, "HTTP_SOAP_ACTION");
   error_if(path && path=='/');
   actionName=parsePath(path);
   fn=symLookup(actionName);
   if(fn){
      fn->handler(wp);
   }
   ....
}

Sym* symLookup(actionName){
   Global symTable  //symTable is a global variable
in running state
   sym=symTable[hash(actionName)];
   while(sym != Null){
      if(strcmp(sym->name, actionName)==0){
         return sym;
      }
      sym=sym->next;
   }
   return Null;
}
```



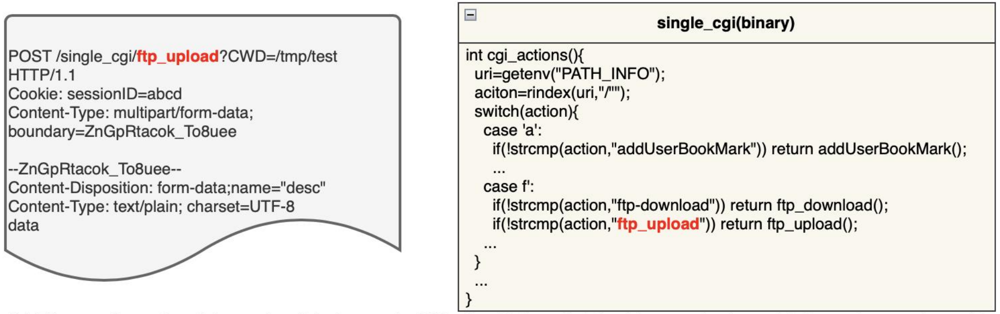
Data flow along the interface is hard to trace, due to some open problems like indirect calls and incomplete data recovery.

Observation

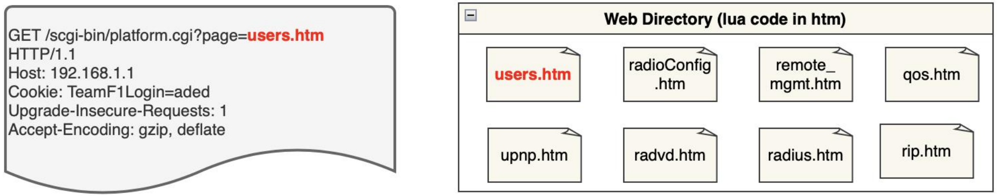
- Pay attention to code slices (firmware code) related to the routing token.
- Routing tokens share a similar pattern in terms of code semantics or formatting.
- Hidden interfaces function similarly to public counterparts and can be accessed with prior knowledge.



(a) The routing token "*GetDDNSSettings*" locates at "SOAPAction" header. The binary *prog.cgi* defines the routing table, which calls the function "*WebsFormDefine*" for mapping the routing token to the corresponding handler.



(b) The routing token "*ftp_upload*" is located within the URI path. The binary *single_cgi* defines the routing table, which uses the jump table (i.e., switch/cases) for mapping the routing token to the corresponding handler.



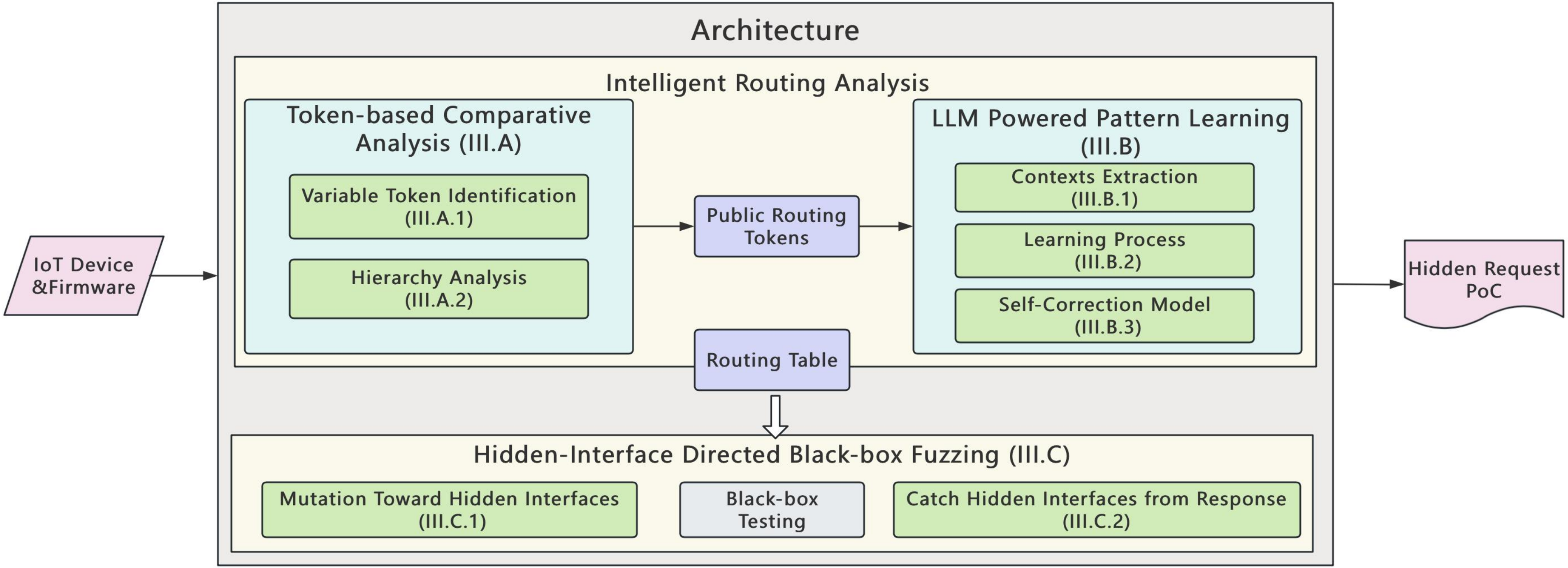
(c) The routing token "*users.htm*" locates at the query in the URL. Each interface corresponds to one HTML file, which contains LUA code to implement the corresponding handler for each specific interface.

Intuition

- First, we identify routing tokens from public requests.
- Then we extract contexts among public routing tokens in programs, learning and deducing their common pattern. Next, with the learned pattern, we try to extract more similar ones from firmware and obtain the maximum approximate set of the routing table.
- Finally, with the help of the routing table as a dictionary, we perform a directed black-box fuzzing to mutate the field of the routing token.

Our Solution: EAGLEYE

Overall Workflow



Locating Routing Tokens

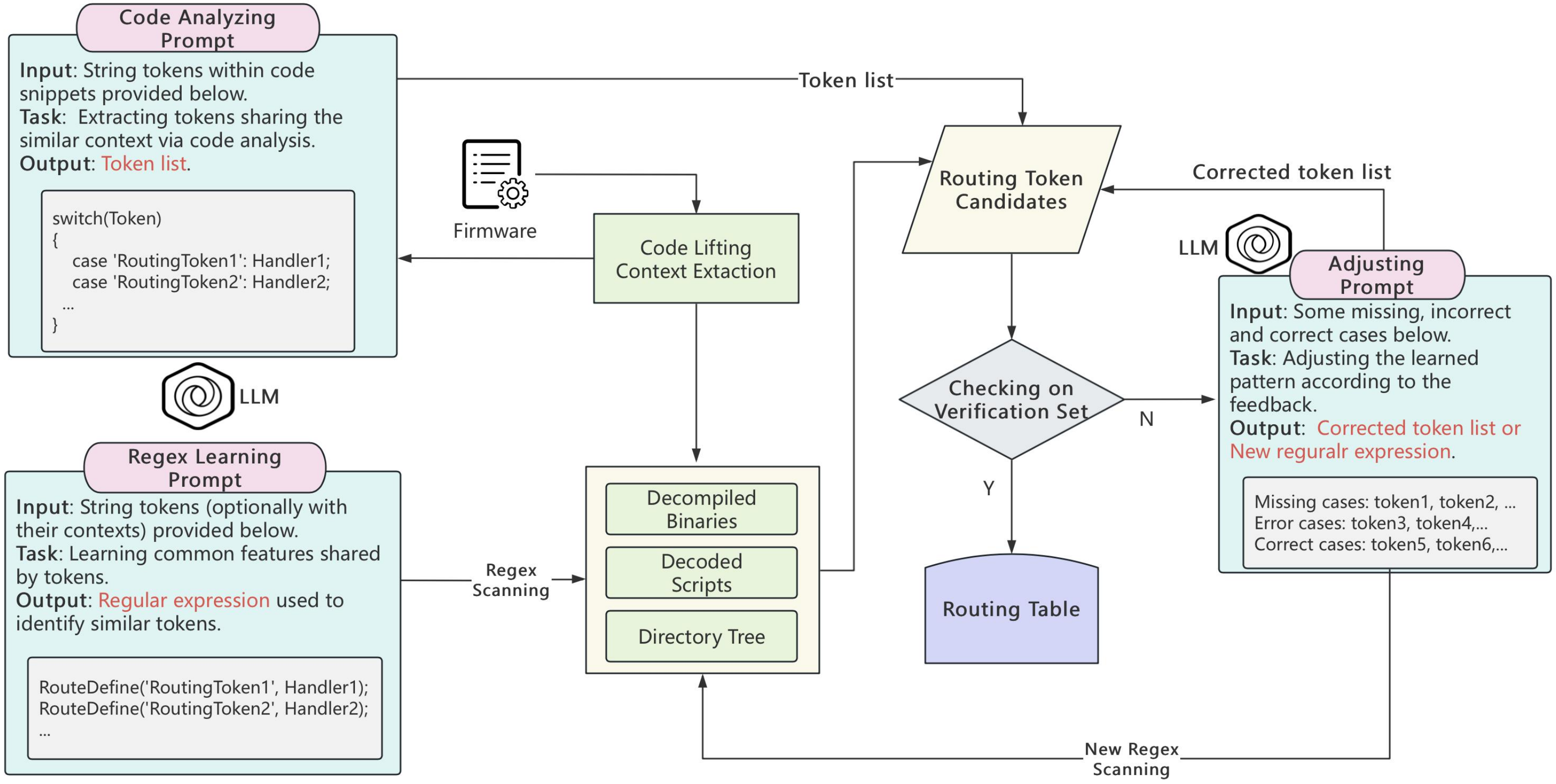
- Algorithm idea:
  - ➢ The routing token varies with the interface change, thus it can be identified by comparing requests of multiple interfaces.
  - ➢ Exclude tokens that cause interference, including session tokens, timestamp tokens, and normal tokens.
  - ➢ Recover the hierarchy among interfaces according to routing tokens' locations.

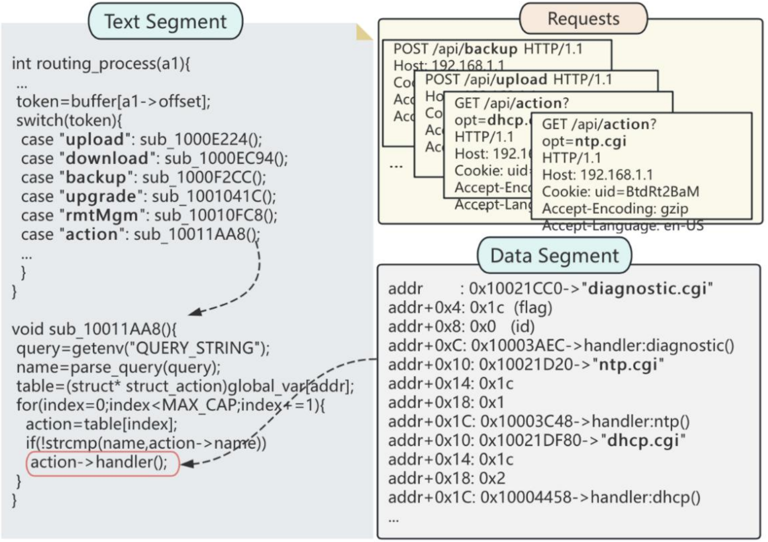
Algorithm 1: Token-based Comparative Analysis

Input: public requests *PubReqs*
Output: routing token set *RToken*, hierarchy *Hier* for multi-level routing tokens.

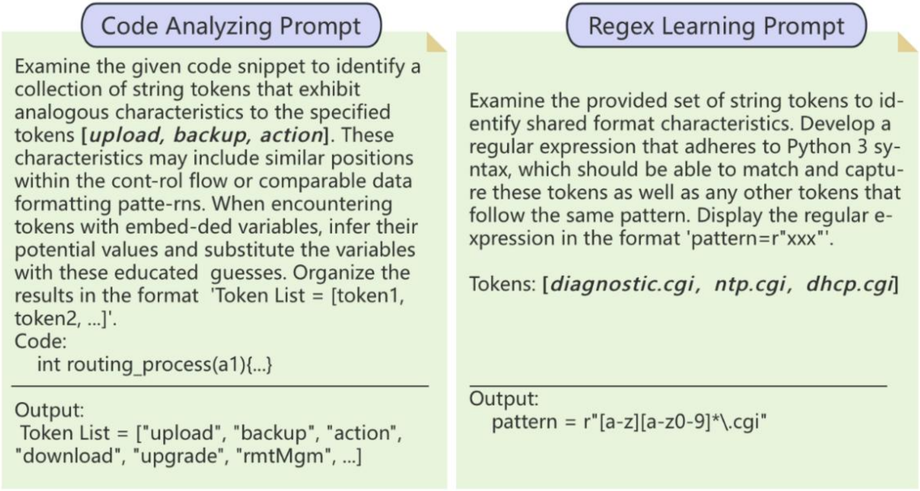
```
foreach req ∈ PubReqs do
    tokens ← Parse(req) /* Parsing request into
      tokens, which are tagged by their field
      locations. */
VTokens ← SelVarToken(tokens) /* Identify tokens
  varying with interfaces. */
RToken ← Filter(VTokens) /* Filter interference
  tokens. */
/* Analyze hierarchy for multi-level tokens */
if Size(RToken ≥ 2) then
    foreach field : token ∈ RToken do
        reqs ← SelReqs(token) /* Select requests
          involving the routing token. */
        subRToken ← Search(reqs) /* Search
          subordinate routing tokens. */
        if Size(subRToken) ≥ 1 then
            Hier ← Edge(field : token, subRToken)
```

Extracting Routing Table



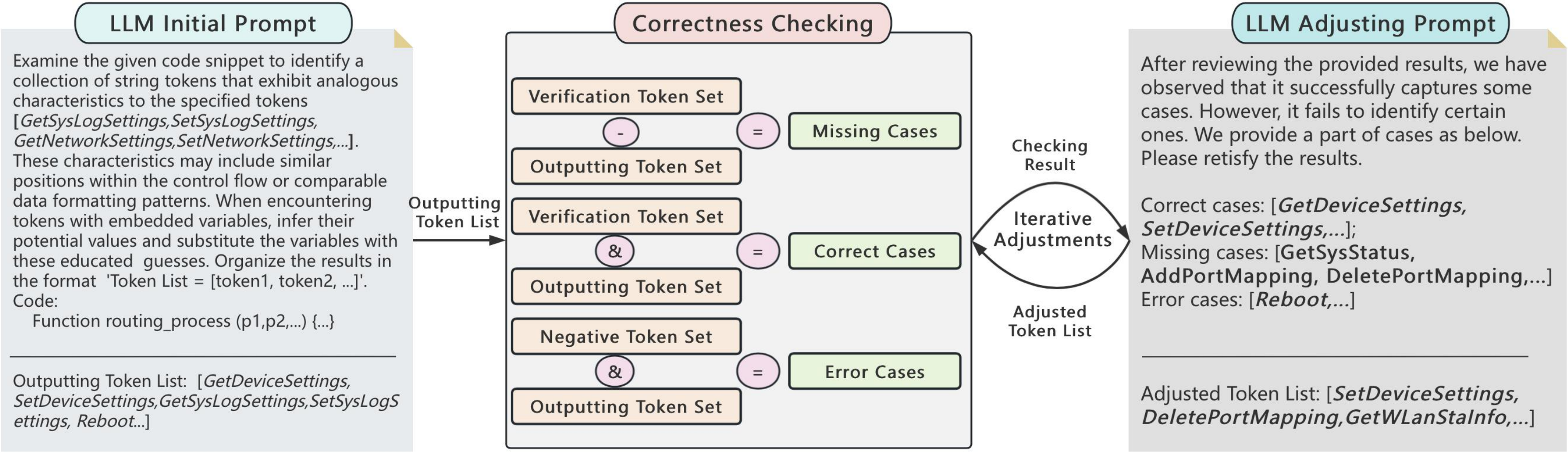


(a) An illustrative program demonstrating a specific routing process that incorporates two-level routing tokens, distinctly positioned within the URI and the query string.



(b) Two carefully designed prompts have been crafted to assist the LLM in identifying routing tokens effectively. These prompts work in tandem to maximize the discovery of potential routing tokens.

LLM Learning Demonstration



Self-Correction Demonstration

Black-box Fuzzing

- Algorithm idea:
  - ➢ Utilize public requests as templates and the routing table as a fuzzing dictionary.
  - ➢ Mutate the seed (i.e., request) by substituting the routing token from fuzzing dictionary.
  - ➢ Iteratively supplement necessary parameters from responses within the fuzzing campaign.
  - ➢ Catch hidden interfaces according to the validity of the response.

Algorithm 2: Hidden-Interface Directed Black-box Fuzzing

Input: Testing device $\mathcal{P}$ with black-box environment, routing table $RTable$

Output: Hidden interfaces PoC

```
SeedsPool ← ∅
ParasPool ← ∅
foreach rtoken ∈ RTable do /* Generate initial seeds using the routing table. */
    SeedsPool ← SeedsPool ∪ Generate(rtoken)
repeat
    seed ← Pop(SeedsPool)
    seed' ← Mutate(seed, ParasPool)
    res ← Interact(P, seed)
    if Validity(res) then /* Check the validity of response */
        PoC ← PoC ∪ seed
        continue
    param ← Distill(res) ;    /* Distill parameters from the response. */
    ParasPool ← ParasPool ∪ {param}
    if Augment(ParasPool) then /* Check if find any new parameter. */
        SeedsPool ← SeedsPool ∪ seed'
until SeedsPool ≡ ∅;
```

Evaluation for EAGLEYE

Testing Set

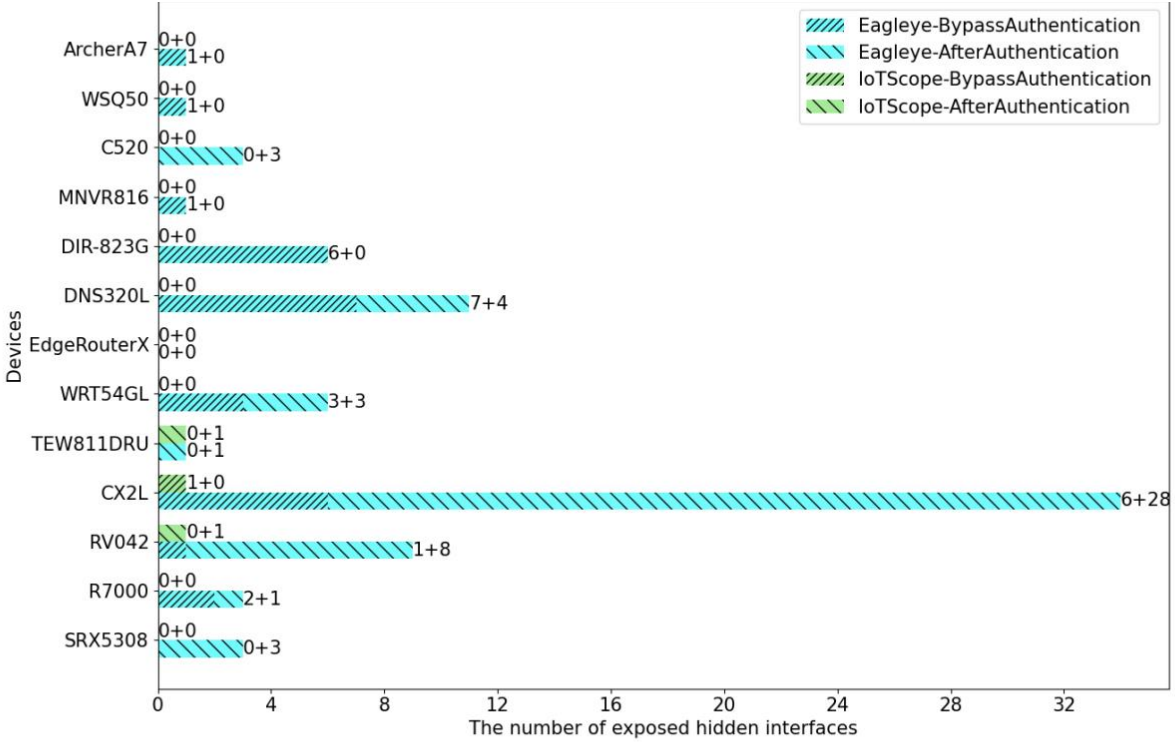
Vendor	Model	Version	Device Type	Web Type
Netgear	SRX5308	4.3.5-5	SSL VPN	Bin+LUA
	R7000	1.0.11.136	WiFi Router	Bin+HTM
Cisco	RV-042	4.2.3.14	VPN Router	Bin+HTM
Motorola	CX2L	1.0.1	WiFi Router	Bin
TrendNet	TEW-811DRU	1.0.10.0	Wifi Router	Bin+ASP
Linksys	WRT54GL	4.30.18	Wifi Router	Bin+ASP
Ubiquiti	EdgeRouterX	2.0.9	Ether Router	Bin+Python
DLink	DNS-320	1.11B01	NAS	Bin+PHP
	DIR-823G	V1.0.2B05	Wifi Router	Bin
Mercury	MNVR816	2.0.1.0.5	Video Recorder	Bin+LUA
ZTE	C520	2.1.6T3	IP Camera	Bin+LUA
Zyxel	WSQ50	V2.20	Wifi Router	Bin+LUA
TPLink	Archer A7	V5_1.2.1	Wifi Router	Bin+LUA

Overall Findings

Vendor	Model	#HINT	#B-Authen	#A-Authen
Netgear	SRX5308	3	0	3
	R7000	3	2	1
Cisco	RV-042	9	1	8
Motorola	CX2L	34	6	28
TrendNet	TEW-811DRU	1	0	1
Linksys	WRT54GL	6	3	3
Ubiquiti	EdgeRouterX	0	0	0
DLink	DNS-320	11	7	4
	DIR-823G	6	6	0
Mercury	MNVR816	1	1	0
ZTE	C520	3	0	3
Zyxel	WSQ50	1	1	0
TPLink	Archer A7	1	0	1
Total	-	79	27	52

#HINT=the number of hidden interfaces, #B-Authen=the number of hidden interfaces bypassing authentication, #A_x0002_Authen=the number of hidden interfaces after authentication.

Comparison with IoTScope



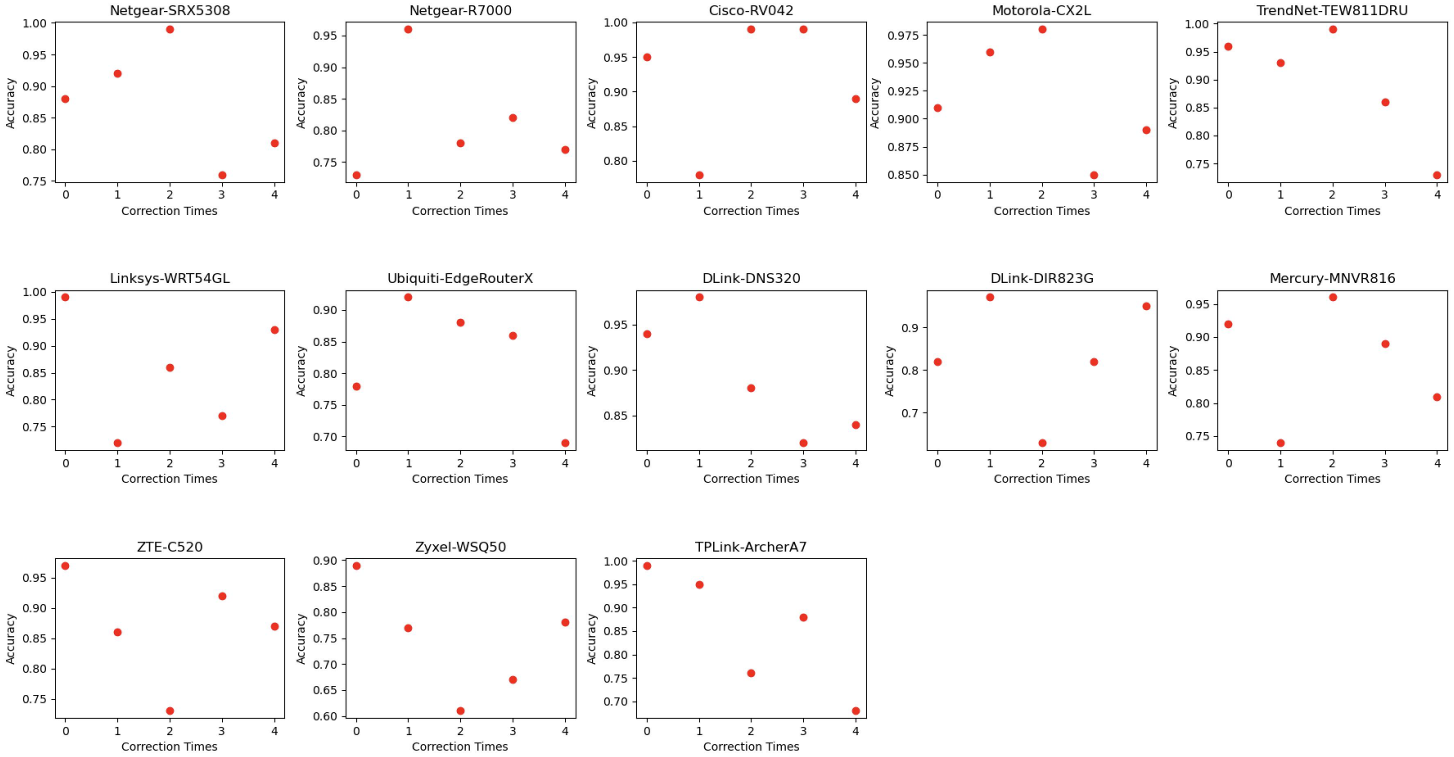
EAGLEYE outperforms IoTScope, exposing 25X more hidden interfaces.

Routing Analysis Effectiveness

Vendor	Model	Routing Token Identification				Routing Table Extraction	
		#VTF	#FTF	#LoC	#Hier	#Table	Layout
Netgear	SRX5308	4	1	Query	2	211	DIS
	R7000	5	3	URI	1	376	DIS
Cisco	RV-042	3	2	URI	1	197	DIS
Motorola	CX2L	4	3	Header	1	270	DIS& AGG
TrendNet	TEW-811DRU	3	1	URI	1	31	DIS
Linksys	WRT54GL	3	2	URI	1	101	DIS
Ubiquiti	EdgeRouterX	9	4	URI& Body	3	34	AGG& DIS
DLink	DNS-320	5	1	URI& Query& Body	2	137	DIS& AGG
	DIR-823G	4	2	URI& Header	1	166	DIS& AGG
Mercury	MNVR816	7	3	URI& Body	2	152	DIS
ZTE	C520	4	2	URI	1	58	DIS
Zyxel	WSQ50	3	1	URI& Body	2	45	DIS
TPLink	Archer A7	4	1	URI	2	28	DIS
Average	-	4.5	2.0	-	1.5	138.9	-

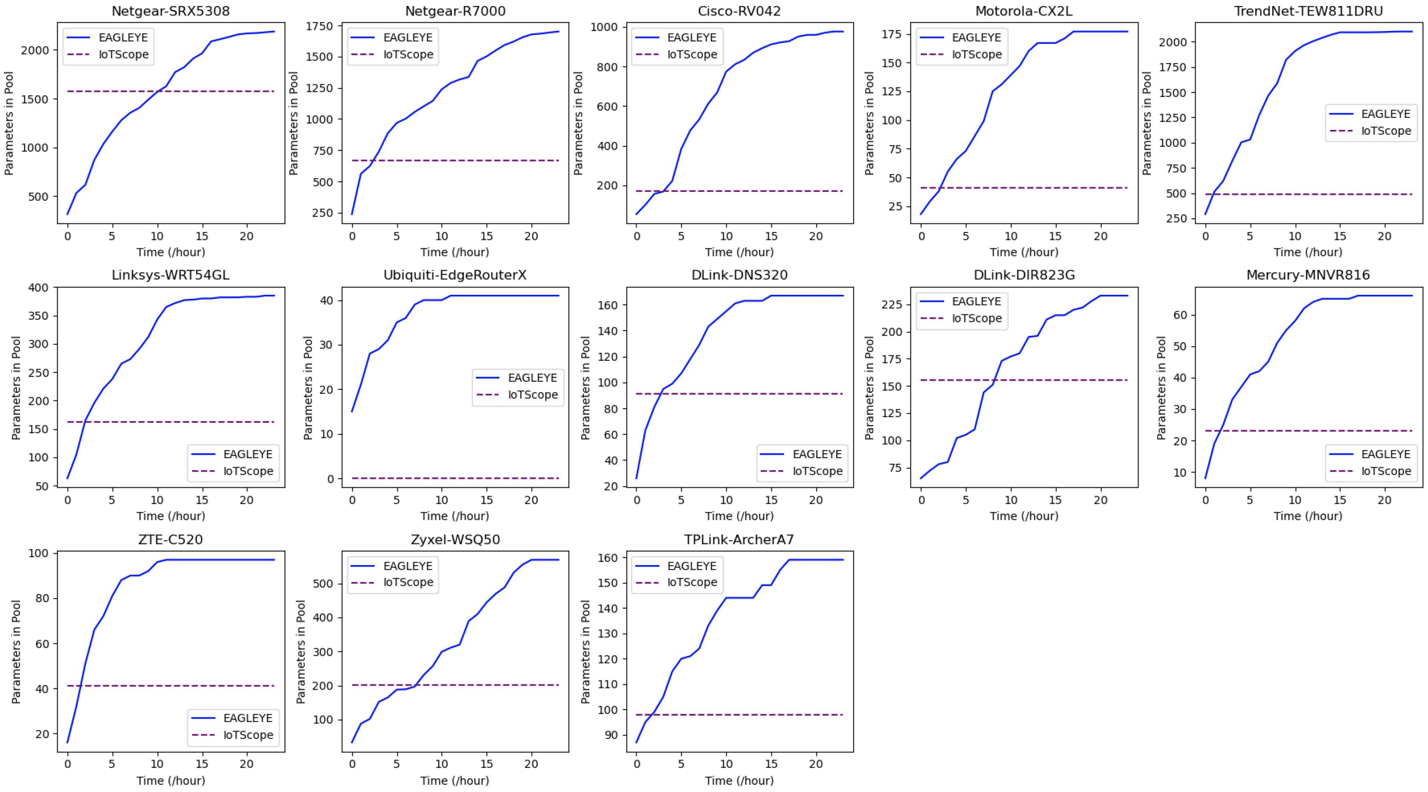
#VTF=the num_x0002_ber of variable token fields, #FTF=the number of filtered token fields, #LoC is where the routing token is located, #Hier=the max level of hierarchy for multi-level routing tokens, #Table=the size of the routing table, layout of routing table:DIS=Distributed, AGG=Aggregated.

Routing Analysis Effectiveness



Accuracy of the pattern learned by LLM varies with the number of corrections. Within limited adjustments, the LLM can learn the correct features among the routing table.

Black-box Fuzzing Effectiveness



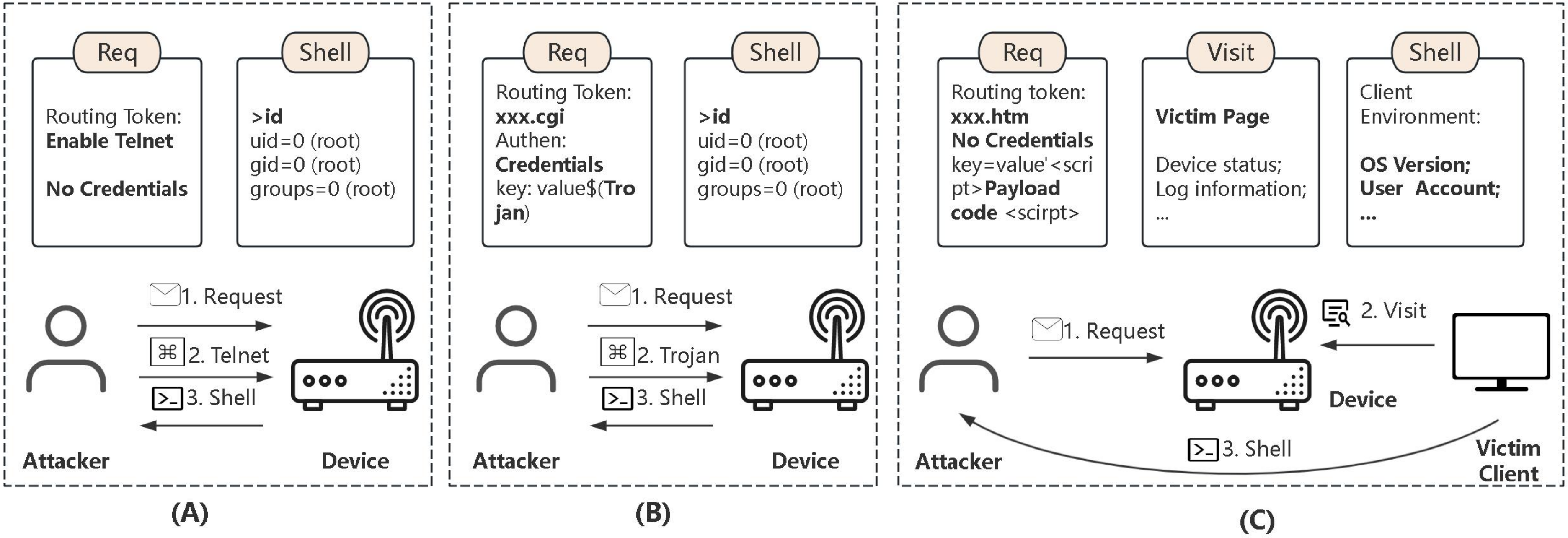
Compared with IoTScope, EAGLEYE can not only obtain more parameters but also the parameters are more accurate. To a certain extent, this trend reflects the continuous improvement of EAGLEYE's mutation quality with the continuous supplement of parameters from responses.

Vulnerabilities

Vendor	Model	#VUL	#CVE	#ID
Netgear	R7000	3	2	CVE-2024-1430, CVE-2024-1431
Cisco	RV-042	2	1	CVE-2024-20362
Motorola	CX2L	6	1	CVE-2024-25360
Linksys	WRT54GL	3	3	CVE-2024-1404, CVE-2024-1405, CVE-2024-1406
DLink	DNS-320L	5	0	-
	DIR-823G	6	0	-
Mercury	MNVR816	1	0	-
ZTE	C520	1	0	-
Zyxel	WSQ50	1	0	-
TPLink	Archer A7	1	0	-
Total	-	29	7	-

#VUL=the number of vulnerabilities, #CVE=the number of assigned CVEs, #ID=detailed CVE ID obtained.

Cases Study



Three typical vulnerabilities in discovered hidden interfaces. (A) A backdoor bypassing authentication: opens the telnet and gains a shell with the highest privilege. (B) A command injection escalating privilege: allows attackers with normal credentials to execute OS commands with the highest privilege. (C) An XSS attacking victim's clients: injects malicious code into one parameter saved in the web, and then the users who visit the victim pages will be infected.

Causes

- **Legacy Code:** Certain interfaces serving for development are left behind in the final product as hidden interfaces due to the incomplete separation of the development and production environment.
- **Permission Management Flaws**: Flaws in the permission management mechanism either fail to cor_x0002_rectly restrict access to privileged interfaces or inadvertently bring internal interfaces to light.
- **Security and Privacy Concerns**: Some vendors may choose not to detail the description of services unrelated to the user in the manual for security and privacy reasons, to prevent potential misuse.
- **Hidden Default Configurations:** The device is designed to work with default settings in most use cases, so vendors may not provide additional configuration options in the manual.

Summary

- We explain the significant problem of hidden web interfaces in IoT devices and give a series of clear definitions.
- We propose a novel solution EAGLEYE, which models the problem of exposing hidden interfaces as a searching process.
- We propose a noval approach, routing analysis, to intelligently learn the routing pattern among interfaces and direct the black-box fuzzing.
- We evaluated Eagleye on 13 commercial IoT devices, and successfully exposed 79 hidden interfaces, on which 29 unknown vulnerabilities including backdoor, command injection, XSS, and information leakage were found, and 7 have been assigned CVEs.

Thanks!