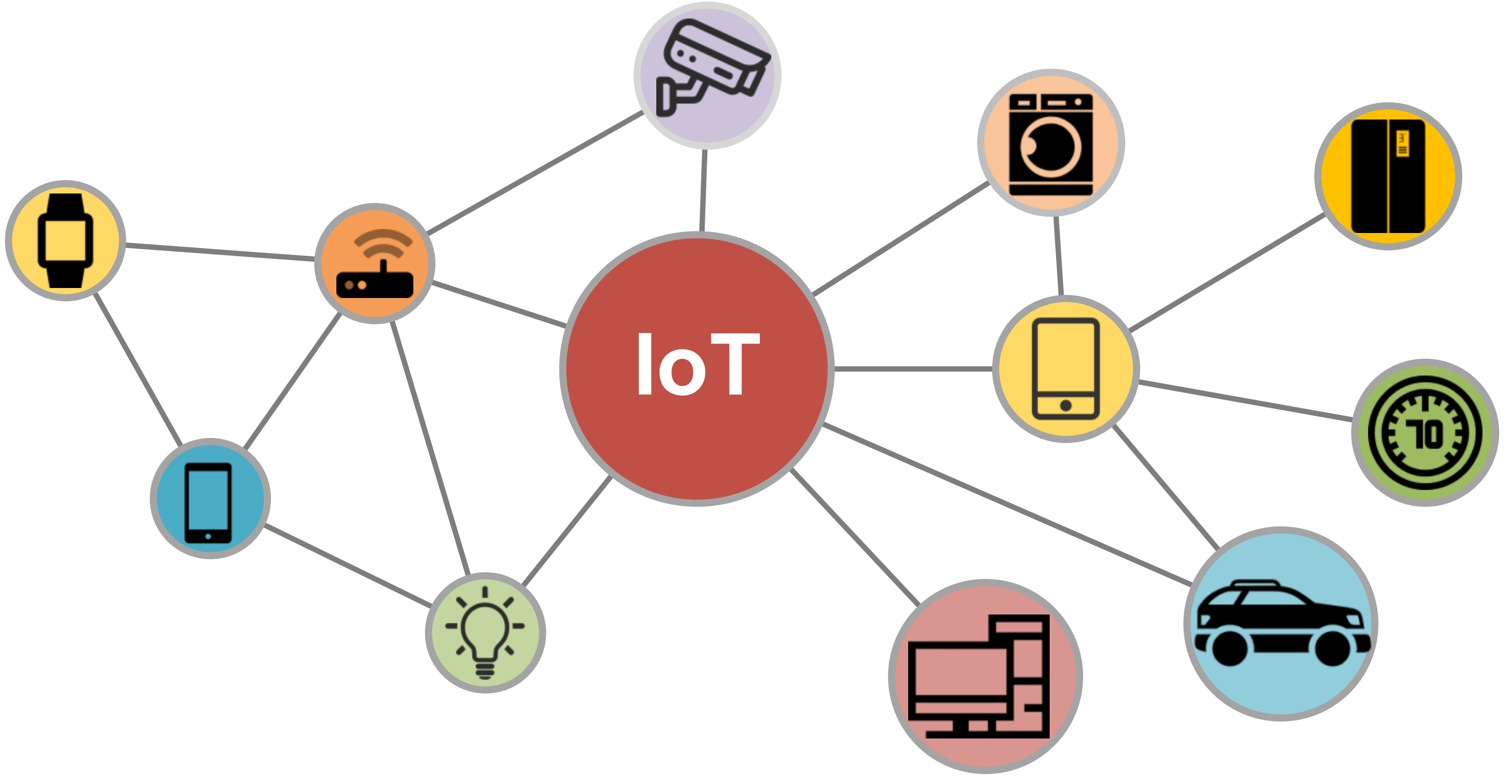
Sharing More and Checking Less: Leveraging Common Input Keywords to Detect Bugs in Embedded Systems

Libo Chen[*], Yanhao Wang[*], Quanpu Cai, Yunfan Zhan, Hong Hu,
Jiaqi Linghu, Qinsheng Hou, Chao Zhang, Haixin Duan, and Zhi Xue

Internet of Things

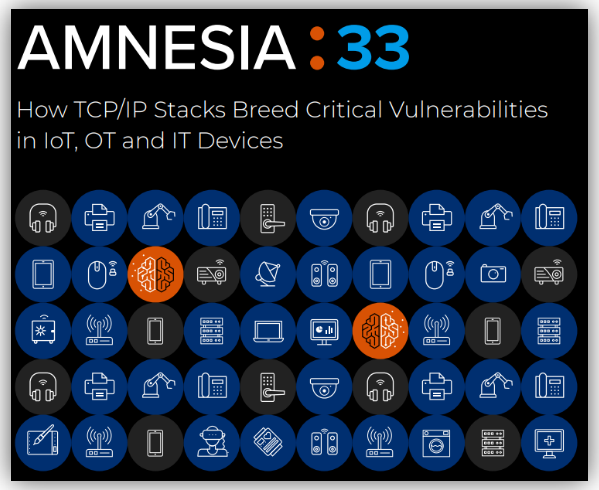
- 5.8 billion IoT endpoints are in use in 2020*
  - Examples: Smart Plugs, Smart Phones, Sensors, Game Consoles



*https://web-release.com/gartner-says-5-8-billion-enterprise-and-automotive-iot-endpoints-will-be-in-use-in-2020

Internet of Things

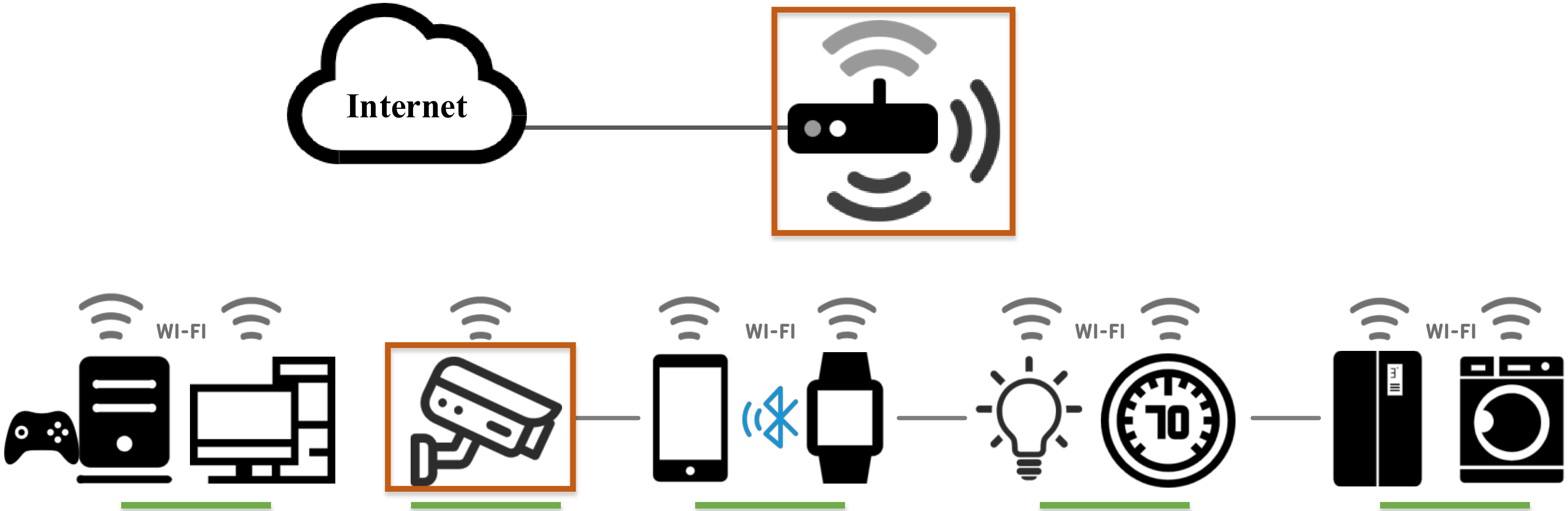
- 57% of IoT devices are vulnerable to medium or high severity attacks*
  - There are a large number of IoT devices
  - Lack of security in most IoT devices
  - Many IoT devices are connected to the network



*https://iotbusinessnews.com/download/white-papers/UNIT42-IoT-Threat-Report.pdf

Internet of Things

- **Wireless routers** and **web cameras** suffer more attacks
  - Web services and network services



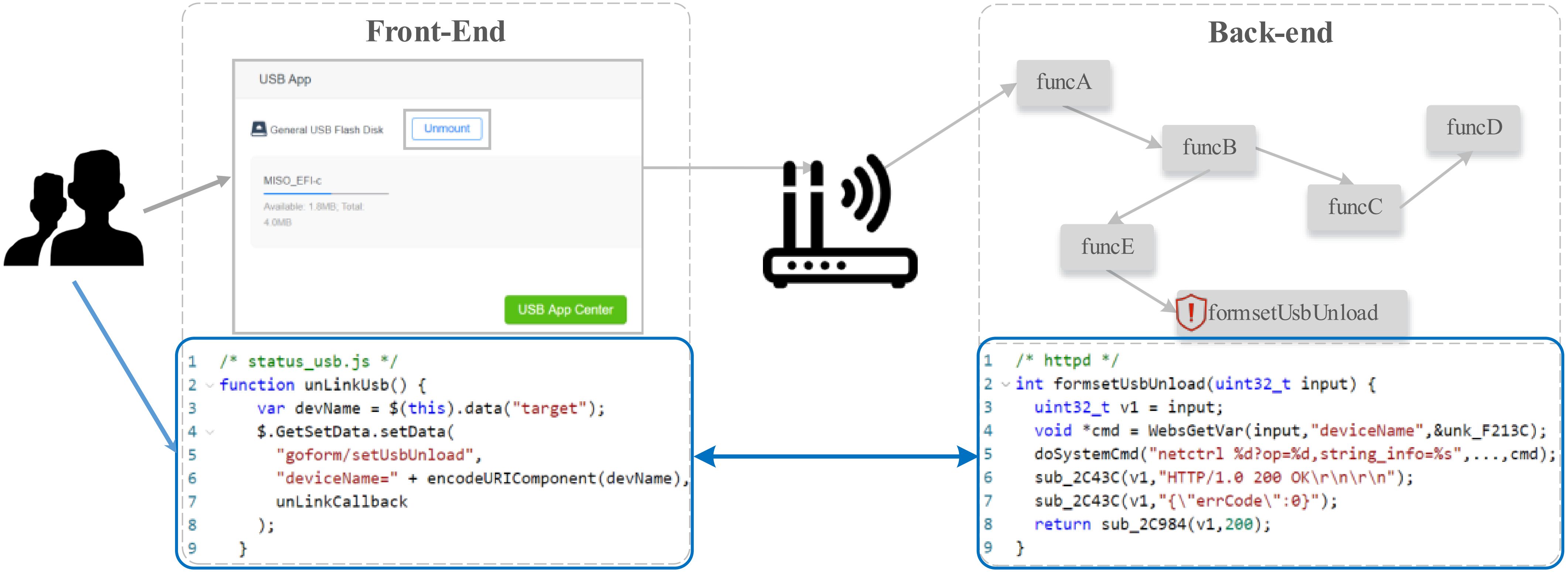
How to detect vulnerabilities in such IoT devices?

Existing Methods

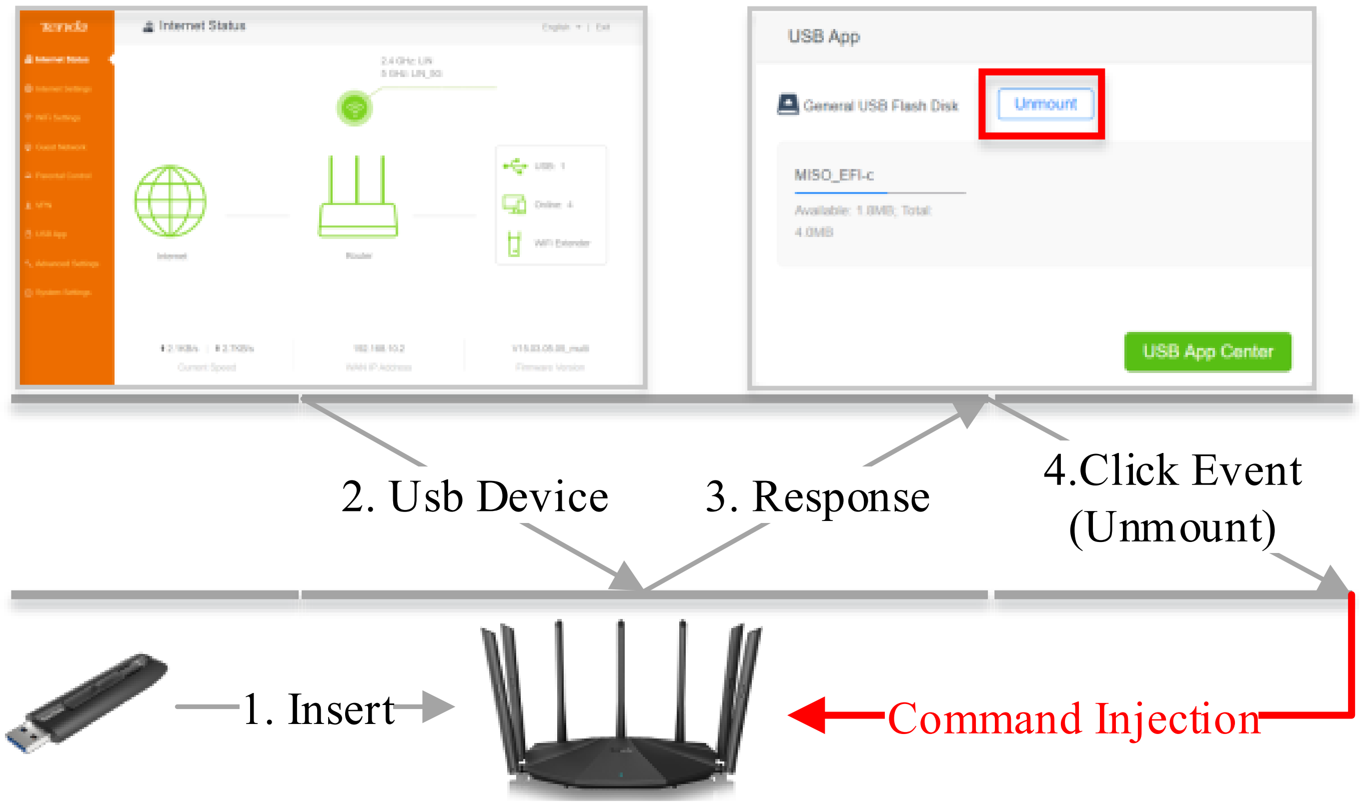
- Dynamic solutions
  - Fuzzing
    - Challenge: run firmware on the device or emulator, e.g., FIRMADYNE
    - Cons: unscalable, specific path condition
- Static methods
  - Symbolic Execution
    - Challenge: cross-binary analysis, e.g., KARONTE
    - Cons: heavyweight, path explosion

Our Solution

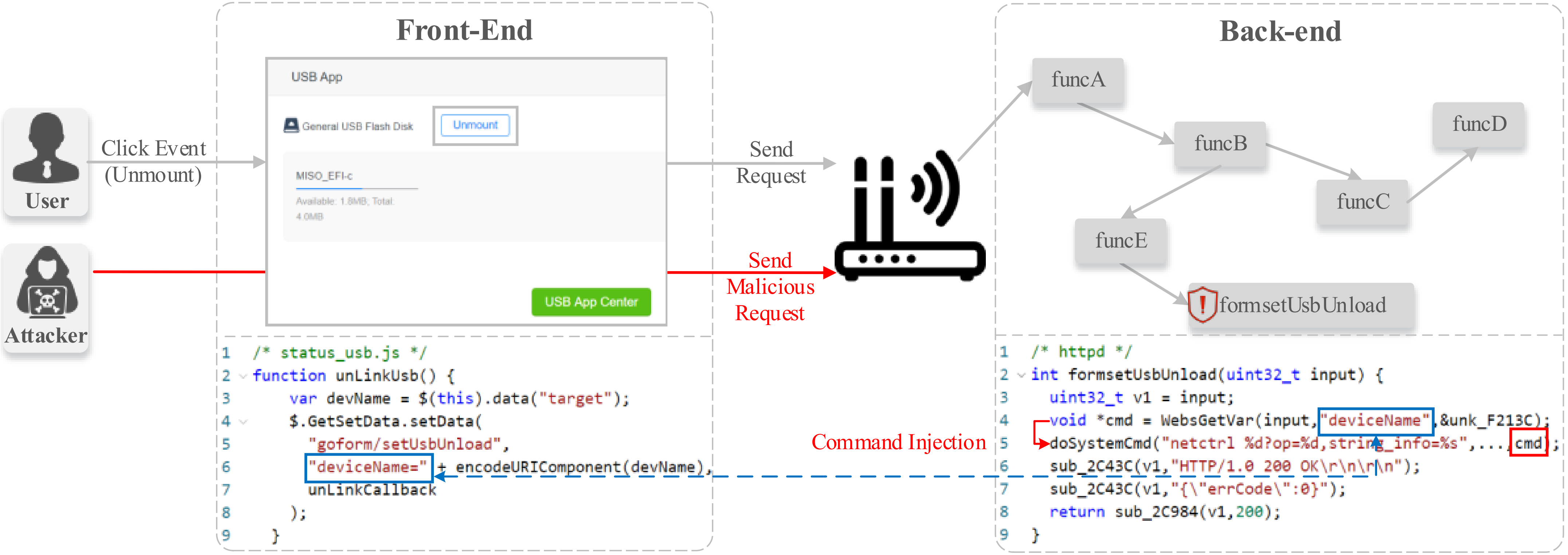
- Static Analysis
  - User-Input ⇨ (Front-End ⇔ Back-End) ⇨ Vulnerability Discovery



Motivating Example



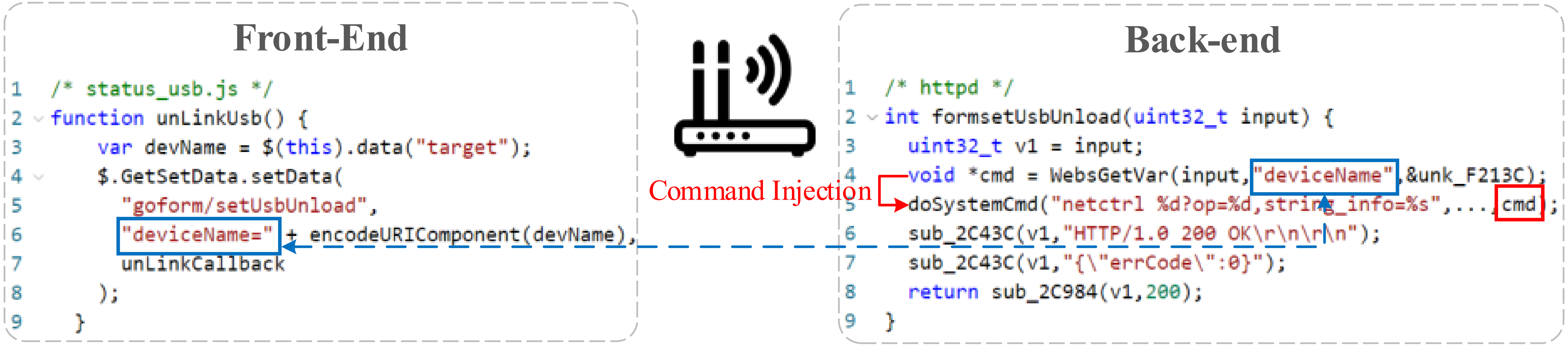
Motivating Example



Malicious Request: http://IP:Port/goform/setUsbUnload?deviceName=***evalCMD***

Intuition

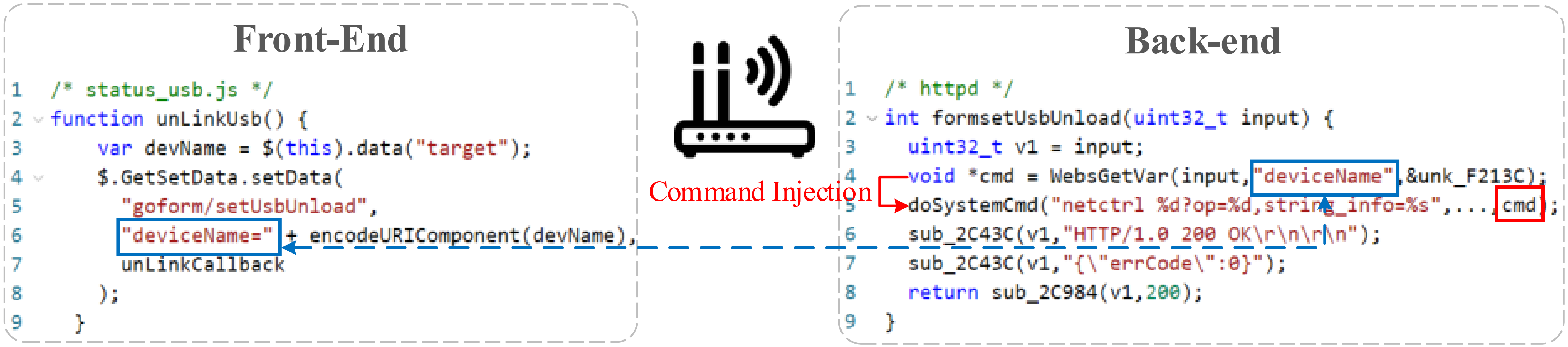
- The strings shown in the web interface are commonly used in both front-end files and back-end functions



- ◆In the front-end, the user-input is labeled with a character string
- ◆In the back-end, the same string is used to extract the user-input from the package

Intuition

- The strings shown in the web interface are commonly used in both front-end files and back-end functions
- Identifying these shared strings and discover the vulnerability from the reference points of the strings in the back-end



Intuition verification

- On average, **92.4%** of the keyword-value pairs captured in the front-end match those in the back-end

Vendor	Device Series	#Front-Str	#Back-allStrs	#Intersect	Verified	%
Tenda	AC9	101	49,288	86	70	81.4
Tenda	AC15	81	241,314	65	63	96.9
Tenda	AC18	81	119,537	66	57	86.4
Tenda	W20E	161	139,885	89	79	88.8
Netgear	R7000P	114	467,706	59	59	100
Netgear	XR300	135	517,254	76	72	94.7
Motorola	M2	133	83,911	31	31	100
D-Link	867	85	84,764	53	50	94.3
D-Link	882	100	522,317	86	81	94.1
TOTOLink	A950RG	69	53,931	31	27	87.1
Average	-	106	227,990	64	59	92.4

- Our intuition works for these common devices.

Challenge

- C1: Identifying keywords in the front-end
- C2: Locating the input handler in the back-end
- C3: Tracking the massive paths of user input to detect vulnerabilities

Front-End

```
/* status_usb.js */
function unLinkUsb() {
    var devName = $(this).data("target");
    $.GetSetData.setData(
      "goform/setUsbUnload",
      "deviceName=" + encodeURIComponent(devName),
      unLinkCallback
    );
  }
```

C1

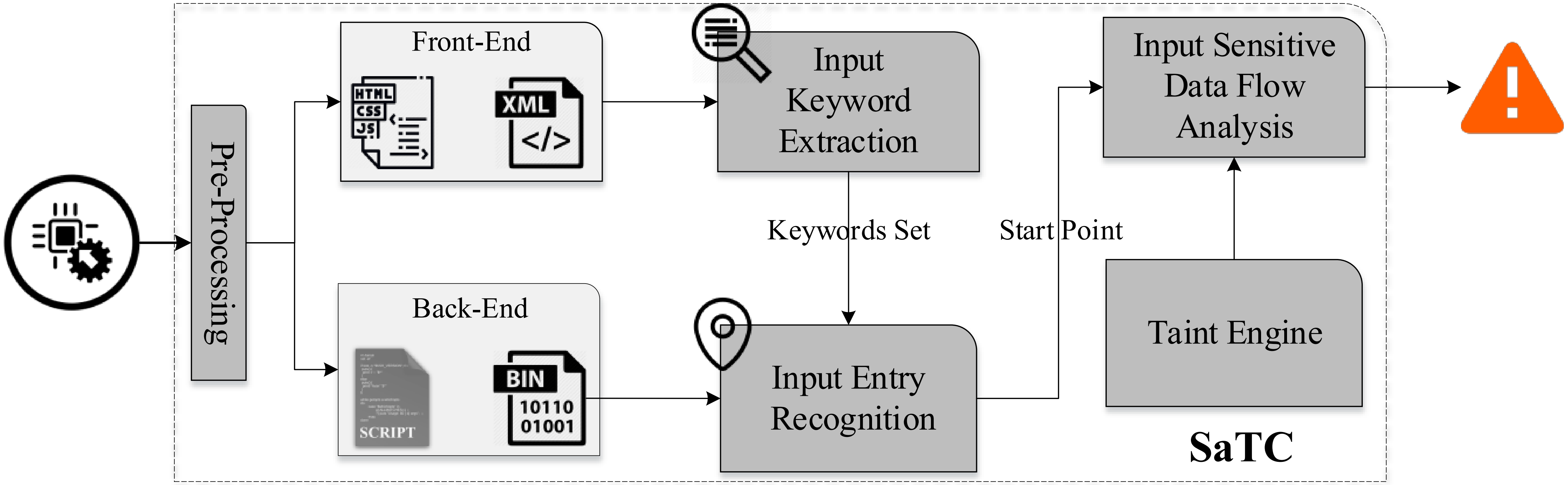
Back-end

```
/* httpd */
int formsetUsbUnload(uint32_t input) {
  uint32_t v1 = input;
  void *cmd = WebsGetVar(input,"deviceName",&unk_F213C);
  doSystemCmd("netctrl %d?op=%d,string_info=%s",...,cmd);
  sub_2C43C(v1,"HTTP/1.0 200 OK\r\n\r\n");
  sub_2C43C(v1,"{\"errCode\":0}");
  return sub_2C984(v1,200);
}
```

C2

C3

Architecture



Input Keyword Extraction

Input Keyword Extraction

- Strings Extraction (Front-end)
  - HTML
    - Use regular expressions
    - Extract the keywords from the values of the "id" and "name" attributes
  - HTTP Service
  - JavaScript
    - Use abstract syntax tree (AST)
    - Extract the value from AST node of which type is "Literal"
  - HTTP Service
  - XML
    - Use regular expressions
    - Extract the keywords from label name of XML node
  - HNAP, UPnP service

Input Keyword Extraction

- Strings Filter (Front-end)
  - Rules
    - Remove strings with special characters, such as !, @, $, etc.
    - Filter out short character strings
  - JavaScript File Filter
    - remove the character strings in share libraries, e.g., charting library
  - Common String Filter
    - remove the keywords referenced by many front-end files, e.g., Button

Input Keyword Extraction

- String Matching
  - Front-end: Strings Extraction ⇨ Strings Filter
  - Back-end: use GNU strings to extract strings from binaries

- Border binary identification
  - Treat the binaries with the maximum matched keywords as the border binary

Input Entry Recognition

Keyword Reference Locator

- The locator detects the location inside the border binary that references to the shared keyword

```
int sub_426B8() {
  Register_Handler("GetSambaCfg",formGetSambaConf);
  Register_Handler("setUsbUnload",formsetUsbUnload);
  Register_Handler("GetUsbCfg",formGetUsbCfg);
}

int formsetUsbUnload(uint32_t input) {
  uint32_t v1 = input;
  void *cmd = WebsGetVar(input,"deviceName",&unk_F213C);
  doSystemCmd("netctrl %d?op=%d,string_info=%s",...,cmd);
  sub_2C43C(v1,"HTTP/1.0 200 OK\r\n\r\n");
  sub_2C43C(v1,"{\"errCode\":0}");
  return sub_2C984(v1,200);
}
```

Implicit Entry Finder

- To find input entries in the back-end that do not have corresponding keywords in the front-end

```
int formSetSambaConf(uint32 user_input) {
  void *data=user_input;
  void *usbname;
  action=Extract(data,"action",&unk_F213C);
  passwd=Extract(data,"password","admin");
  premit=Extract(data,"premitEn","0");
  intport=Extract(data,"internetPort","21");
  usbname=Extract(data,"usbName",&unk_F213C);
  if (!strcmp(action,"del")) {
    doSystemCmd("cfm post netctrl %d?op=%d,string_info=%s",51,3,usbname);
  }
}
```

Cross-Process Entry Finder

- To locate the data-flow of user input interrupted at the process boundary, such as NVRAM and Environment variables

```
SetWebFilterSettings() {//in binary prog.cgi
  pcVar1=webGetVarString(wp,"/SetWebFilterSettings/WebFilterMethod");
  iVar2=webGetCount(wp,"/SetWebFilterSettings/WebFilterURLs/string#");
  i = 0;
  if (iVar2<=i) {
    /* NVRAM operations */
    nvram_safe_set("url_filter_mode",pcVar1);
    nvram_safe_set("url_filter_rule",tmpBuf);
  }
}
upload_url_filter_rules() {//in binary rc
  /* NVRAM operations */
  iVar1=nvram_get_int("url_filter_max_num");
  __s1=(char *)nvram_safe_get("url_filter_mode");
  __src=(char *)nvram_safe_get("url_filter_rule");
}
```

Input Sensitive Taint Analysis

Coarse-Grained Taint Engine

- Taint Source (Start point)
  - Mark taint sources based on the results of the input entry recognition
  - A taint source can be a return value or a parameter of a target function

```c
int formsetUsbUnload(uint32_t input) {
  uint32_t v1 = input;
  void *cmd = WebsGetVar(input,"deviceName",&unk_F213C);
  doSystemCmd("netctrl %d?op=%d,string_info=%s",...,cmd);
  sub_2C43C(v1,"HTTP/1.0 200 OK\r\n\r\n");
  sub_2C43C(v1,"{\"errCode\":0}");
  return sub_2C984(v1,200);
}
```

Coarse-Grained Taint Engine

- Taint Specification
  - ◆ Instruction Level
  - ◆ Function Call Handler
    - Summarizable function
    - General function
    - nested function

Disassembly Code

```c
void funcA(char *s1, char *s2) {
  int len = strlen(s1);
  for (int i=0; i<len; i++)
    switch (s1[i]=='/') {
      case '/': goto get;
      case ';': return;
      default: break;
    }
get:
  if (i+1 < n)
    strcpy(s2, s1[i]);
}

char* funcB(char *i,
            char *t,
            char *r) {
  char *target = funcC(i, t);
  if ( target)
    return target;
  else
    return r;
}
```

Taint Analysis

```
Taint Source:
T(s1[...])
Is_NestFunc(funcA)->False
StepInto(funcA, ...)

Has_Summary(strcpy)->True
Taint_Rule: T(src) => T(dst)
T(s1[...]) => T(s2[...])

Taint Source:
T(i[...])
Is_NestFunc(funcB)->True

Is_Pointer(retv)->True
Is_Used(retv)->True
T(i[...]) => T(retv[...])
```

Evaluation

- Q1: Can SaTC find real-world vulnerabilities?
- Q2: Can SaTC accurately detect the input keywords?
- Q3: How efficient and accurate is our taint analysis?

Evaluation

- Dataset
  - 6 vendors and 14 series
  - 39 firmware samples
  - Wireless router and web camera
  - ARM and MIPS

Vendor	Type	Series	#	SizeP	SizeUP	Arch
Netgear	Router	R/XR/WNR	19	38M	192M	ARM32
Tenda	Router	AC/G/W	9	12M	105M	ARM32
TOTOLink	Router	A/T	2	5M	60M	ARM32
D-Link	Router	DIR/DSR	5	8M	123M	MIPS32
Motorola	Router	C1/M2	2	12M	64M	MIPS32
Axis	Camera	P/Q	2	60M	700M	ARM32

Q1: real-world vulnerability

- Vulnerability
  - 33 new vulnerabilities
  - 30 of them are assigned CVE/CNVD/PSV numbers
- Services
  - HTTP
  - Universal Plug and Play (UPnP)
  - Home Network Administration Protocol (HNAP)

Vendo	Device Series	Type	Bug IDs	Ksrc	Service
Netgear	R7000/R7000P	BoF	PSV-2020-0267	HTML	HTTP
			CVE-2020-28373	XML	UPnP
	R6400v2	CI	CNVD-2020-15102	HTML+	HTTP
			CNVD-2020-28091	HTML+	HTTP
	XR300	CI	PSV-2020-0277	HTML	HTTP
Tenda	W20E	CI	CNVD-2019-22866	JS	HTTP
			CNVD-2019-22867	JS	HTTP
			CNVD-2019-22869	HTML	HTTP
		IAC	1 unassigned	JS	HTTP
	G1/G3	CI	CNVD-2020-46058	JS	HTTP
			CNVD-2020-46059	JS	HTTP
	AC15/AC18	CI	CNVD-2020-29725	JS	HTTP
			CNVD-2020-40766	JS	HTTP
			CNVD-2020-40767	JS	HTTP
			CNVD-2020-40768	JS	HTTP
TOTOLink	T10	CI	CNVD-2020-28089	JS	HTTP
	A950RG	CI	CNVD-2020-28090	JS	HTTP
			1 unassigned	JS	HTTP
D-Link	DIR 823G	IAC	CVE-2019-7388	JS	HTTP
			CVE-2019-7389	JS	HTTP
			CVE-2019-7390	JS	HTTP
			CVE-2019-8392	JS	HTTP
	DIR 878	CI	CVE-2019-8312	XML	HNAP
			CVE-2019-8314	XML	HNAP
			CVE-2019-8316	XML	HNAP
			CVE-2019-8317	XML	HNAP
			CVE-2019-8318	XML	HNAP
			CVE-2019-8319	XML	HNAP
	DIR 878 882	IAC	1 unassigned	JS	HTTP
		CI	CNVD-2020-23845	XML	HNAP
Motorola	C1 M2	CI	CVE-2019-9117	JS	HTTP
			CVE-2019-9118	JS	HTTP
			CVE-2019-9119	JS	HTTP
Total		3	33	3	3

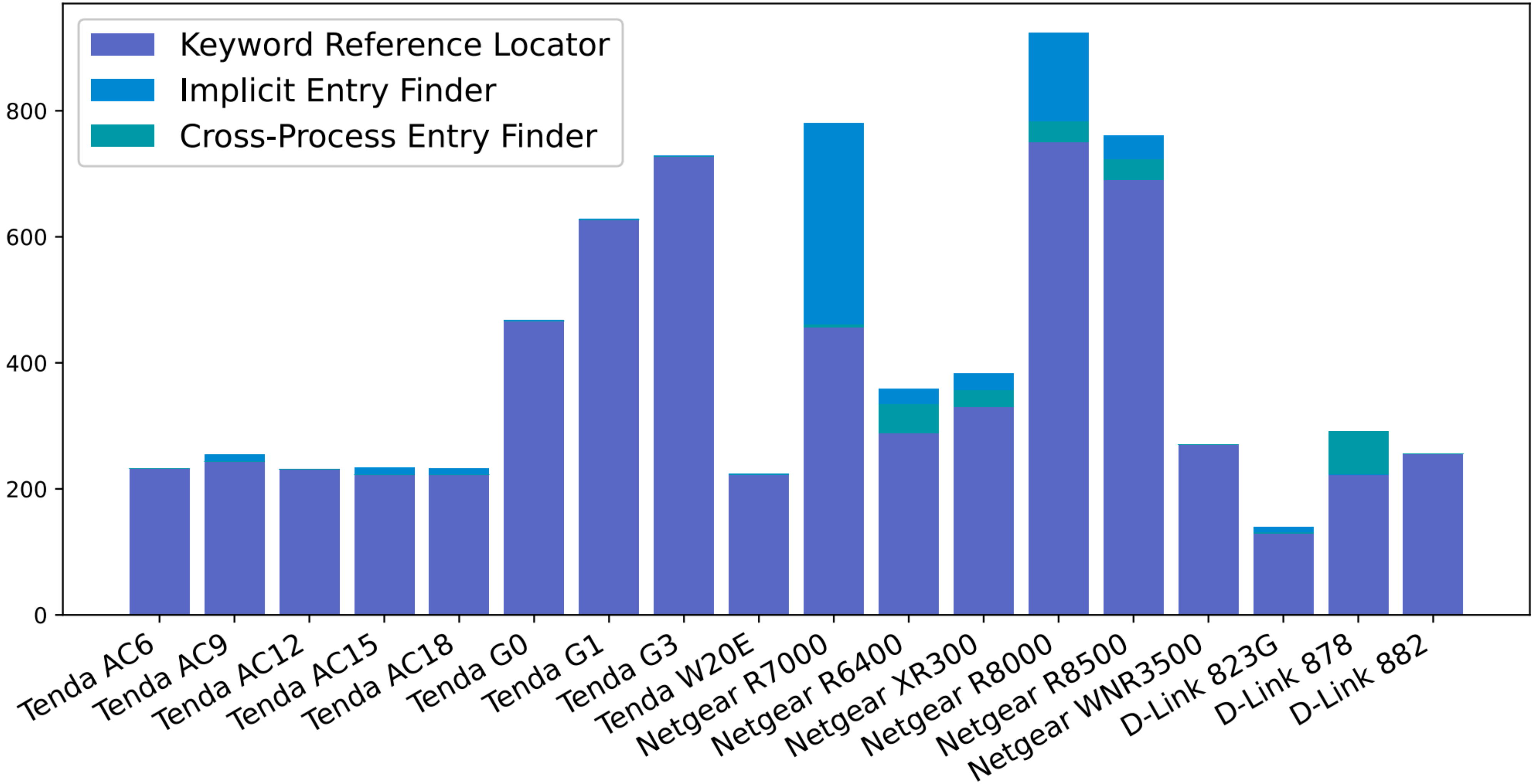
Q2: Keywords Extraction

- 20 out of 33 bugs are related to input keywords found in JavaScript files
- Eight are related to keywords in XML files
- Four of them rely on the keywords in HTML files

Q2: Keywords Extraction

Vendor	Series	Input	Keyword Extraction			Border Binary Recognition				Verification			
			str	fKey	time(s)	strAll	borderBin	borderKey	time(s)	vPar/tPar	%	vAct/tAct	%
Tenda	AC15	119	7,771	995	254	241,314	httpd	447	51	223/319	69.91	101/128	78.91
Tenda	AC18	119	7,663	984	145	119,537	httpd	447	57	222/319	69.59	101/128	78.91
Tenda	W20E	134	10,581	1,744	102	139,885	httpd	834	102	423/589	71.82	222/245	90.61
Tenda	G1	147	14,241	137	1,952	123,960	httpd	636	75	422/586	72.01	5/56	8.39
Tenda	G3	147	14,241	137	1,952	123,960	httpd	636	75	422/586	72.01	5/56	8.39
Netgear	XR300	864	18,889	4,232	683	517,254	httpd	1,226	1,280	330/1,014	32.54	11/211	5.21
Netgear	R6400	489	5,692	1,729	32	478,005	httpd	887	449	288/706	40.79	10/180	5.56
Netgear	R7000	610	9,421	2,304	167	330,087	httpd	1,132	452	456/920	49.57	0/211	0
Netgear	R7000P	607	8,670	2,257	67	467,706	httpd	1,121	579	455/919	49.51	0/201	0
D-Link	878	251	26,389	3,415	492	139,948	prog.cgi	735	170	223/735	45.44	140/520	26.92
D-Link	882	252	25,608	3,025	1,149	522,317	prog.cgi	878	670	256/416	61.54	91/461	19.74
D-Link	823G	110	10,200	2,544	370	48,005	goahead	255	78	27/167	16.17	24/87	27.59
TOTOLink	T10	59	6,217	869	231	51,898	system.so	64	24	35/41	85.37	20/23	86.96
TOTOLink	A950RG	73	7,520	1,267	303	53,931	system.so	180	31	53/66	80.3	35/114	30.7
Motorola	C1	105	12,347	2,133	315	90,652	prog.cgi	370	89	44/147	29.93	175/223	78.48
Motorola	M2	103	10,982	1,863	303	83,911	prog.cgi	333	93	38/137	27.74	143/196	72.96

Q3:Input Entry Recognition



Q3: False positives of taint analysis

- SaTC raised 101 alerts
- 46 of them are true positives
- Missing abstracts for the common functions, such as **atoi()**

```
void formDelVpnUsers(...)
{
  // reference point
  taint = websGetVar(wp, "vpnUserIndex", byte_E945C);
  strncpy(sUserIndexCopy2, taint, 0x3Fu);
  getVpnServerType(sServerType);
  for (pIndex = (unsigned int8 *)strtok_r((char *)sUserIndexCopy2, "\t", (char **)&pSavePtr); pIndex;
       pIndex = (unsigned int8 *)strtok_r(0, "\t", (char **)&pSavePtr)) {
    v6 = atoi((const char*)pIndex);//over-tainting -> v6
    get_item_in_list("vpn.ser.pptpuser", "&", v6 + 1, 1, sUserId);//over-tainting -> sUserId
    doSystemCmd("cfm post netctrl %s?op=%d,index=%s", (const char *)sServerType, 10, (const char *)sUserId);
  }
}
```

Summary

- We propose SaTC, a novel approach to detect security vulnerabilities in embedded systems
- Based on the insight that variable names are commonly shared between front-end files and back-end functions
- SaTC has successfully discovered 33 zero-day software bugs from 39 firmware samples, and 30 of them have been assigned CVE/CNVD/PSV IDs

Code and Dataset:

https://github.com/NSSL-SJTU/SaTC

Thank You

Questions?

Email:bob777@sjtu.edu.cn