Ph.D. Thesis Defense

Improving Large-Scale Vulnerability Analysis of IoT Devices with Heuristics and Binary Code Similarity

Dongkwan Kim
School of Electrical Engineering
KAIST

2021.12.07

Advisor: Yongdae Kim

Committee Members:
Prof. Yongdae Kim – Chair
Prof. Sang Kil Cha
Prof. Sooel Son
Prof. Shin Yoo
Prof. Insu Yun

WHY IS IOT SECURITY IMPORTANT?

New Mirai Variant and ZHtrap Botnet Malware Emerge in the Wild

March 16, 2021 Ravie Lakshmanan

Ir

Cyber Attacks on Industrial Sector: Lloyd's

e Internet of Things.

February

Whistleblower: Ubiquiti Breach "Catastrophic"

ent

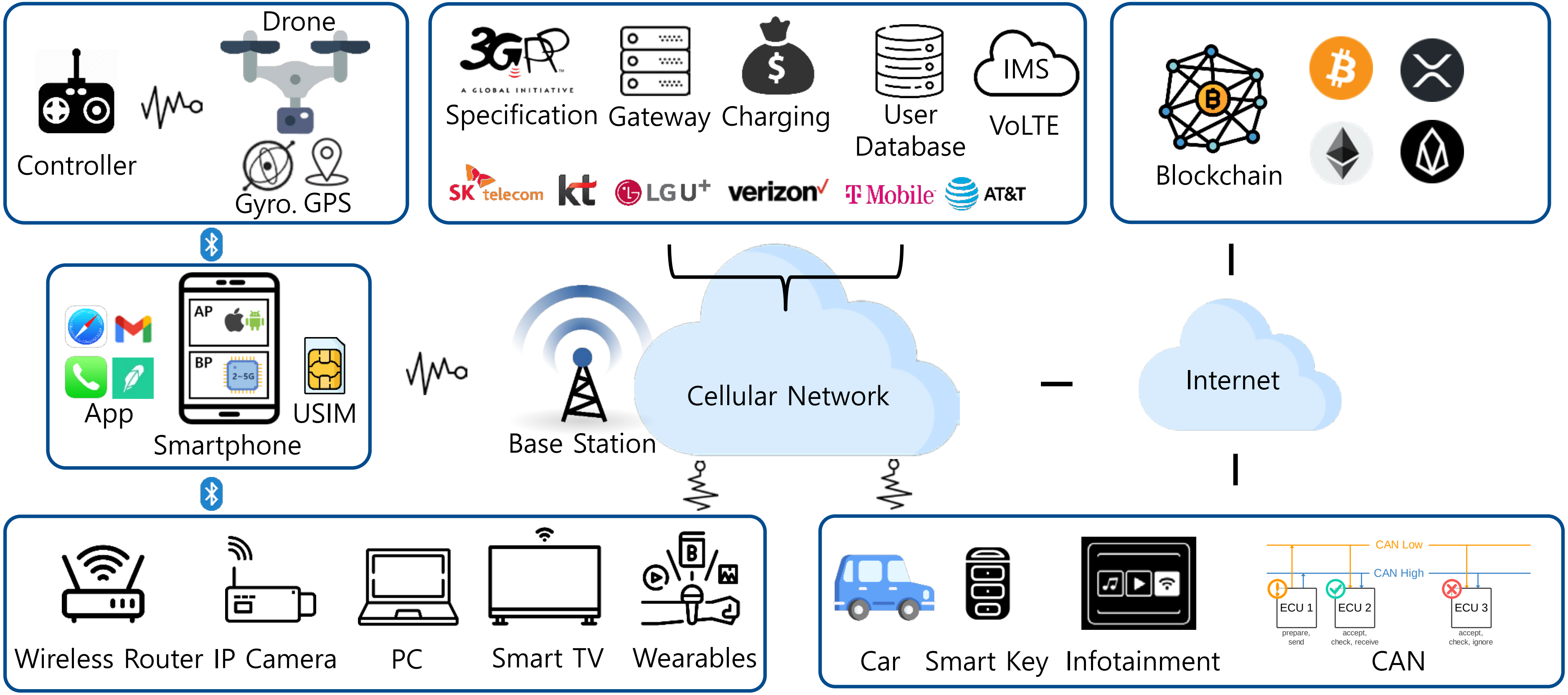
March 30, 2021 142 Comments

On Jan. 11, Ubiquiti Inc. [NYSE:UI] — a major vendor of cloud-enabled Internet of Things (IoT) devices

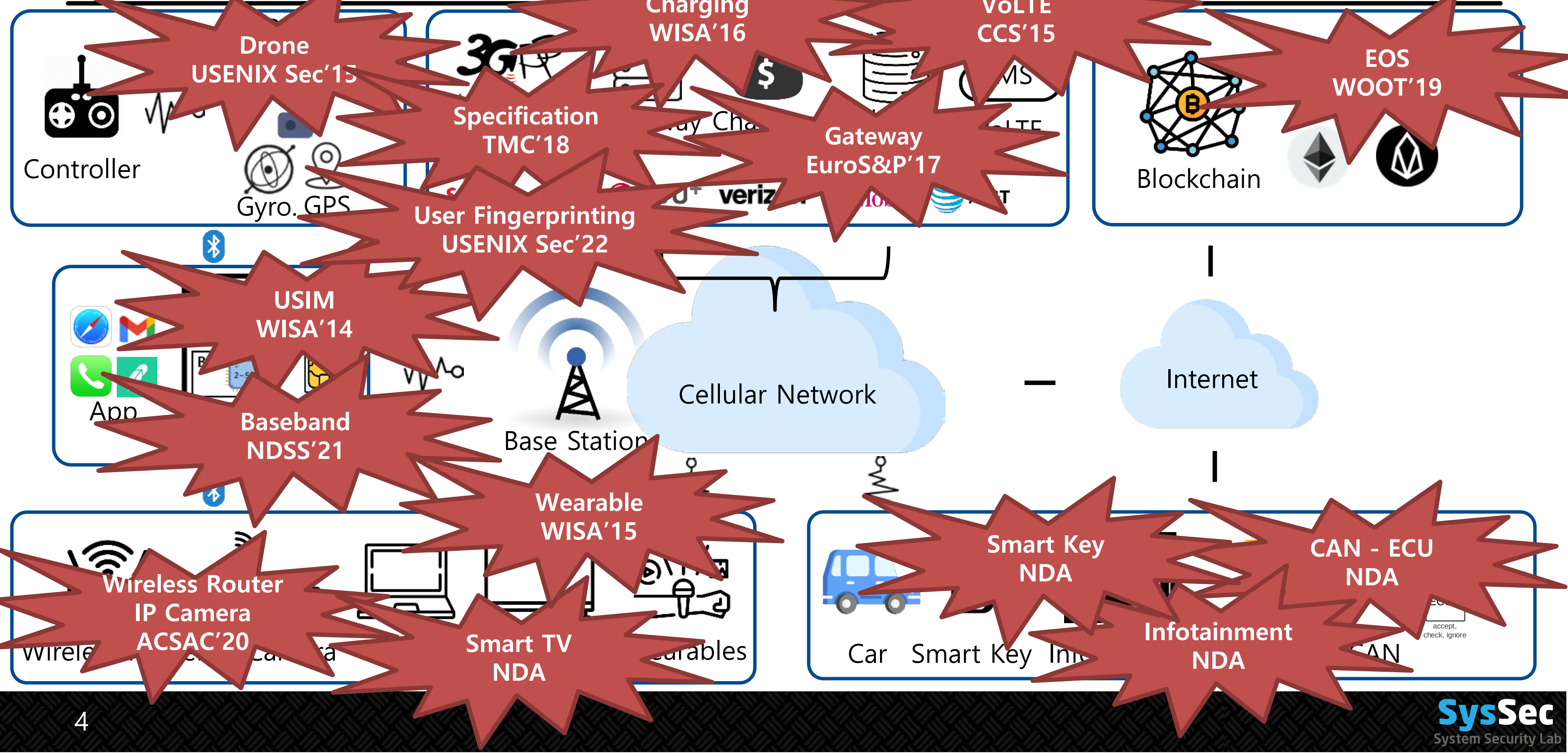
Your insecure Internet of Things devices are putting everyone at risk of attack

IoT devices are becoming more and more popular but many of the products people are installing don't come with adaquate security - and that's something cyber criminals can take advantage of.

IoT Ecosystem (In)Security



IoT Ecosystem (In)Security



SysSec
System Security Lab

Security Analysis of IoT Devices

- ❖ The number of IoT devices are rapidly increasing

→ **Scalability** is the key to analyzing threats in widespread devices

- ❖ Challenge: absence of development standards
  - – Opacity (Obscurity)
    - ▪ Vendors do not release implementation details
  - – Diversity
    - ▪ Complex hardware/implementation diversity

→ **Scaling up** the vulnerability analysis is **challenging**

IoT Analysis Procedure

❖ Firmware collection
- Physically obtaining numerous devices is infeasible
- Download firmware images from vendors websites

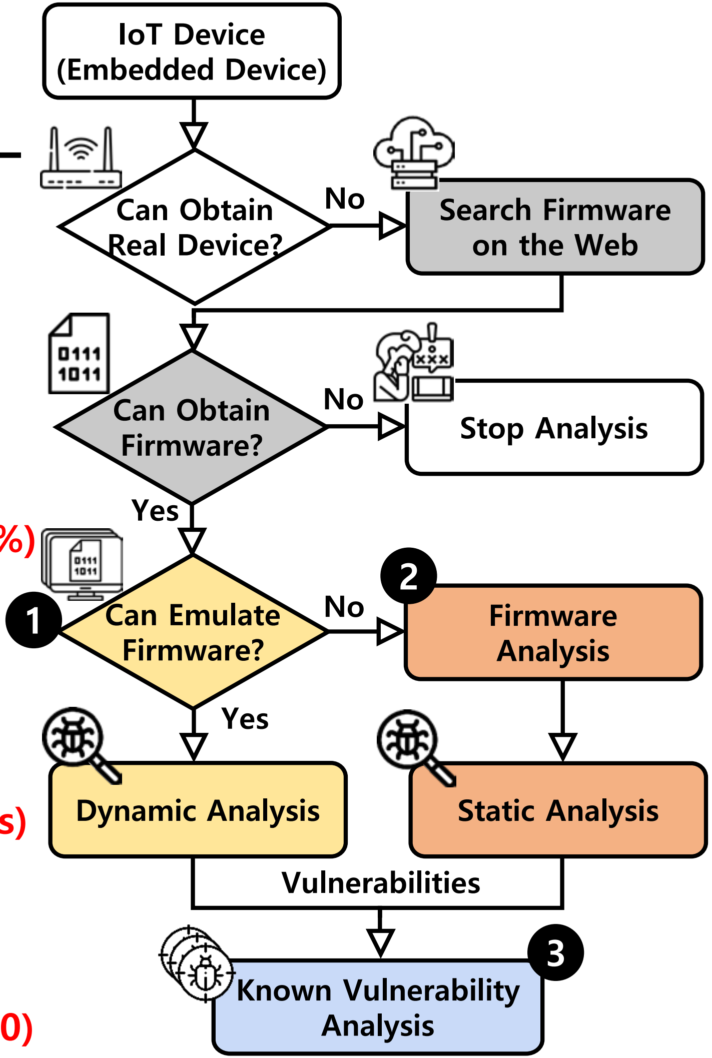
❶ Firmware emulation and dynamic analysis
- Build a virtual environment mimicking a real device
- Run automated pentesting (e.g., Metasploit)
- Run fuzzers (e.g., AFL) ➡ **Low emulation rate (16.3%)**

❷ Firmware and static analysis
- Analyze firmware structure and memory layout
- Identify target functions
- Run symbolic execution (e.g., angr)

➡ **Not scalable (≤ 10 images)**

❸ Known vulnerability analysis
- Build PoC exploits and run them (e.g., Metasploit)
- Build signatures and search them (e.g., BCSA)

➡ **A few studies (≤ 10)**



Motivating Observation

- Existing academic studies focused on developing **novel/fresh** approaches
  - ➔ **Such approaches often disregard/ignore heuristics**

- Opacity (Obscurity)
  - Vendors do not release implementation details
  - ➔ **Conducting empirical analysis and developing "dirty" heuristics are inevitable**

- Diversity
  - Complex hardware/implementation diversity
  - ➔ **Systematizing the developed heuristics is necessary**

Research Statement

Although heuristics seem to be trivial and not novel, developing/systematizing “dirty” heuristics is necessary to enable large-scale vulnerability analysis of IoT devices

Target Device Categories

❖ Select two device categories having different characteristics

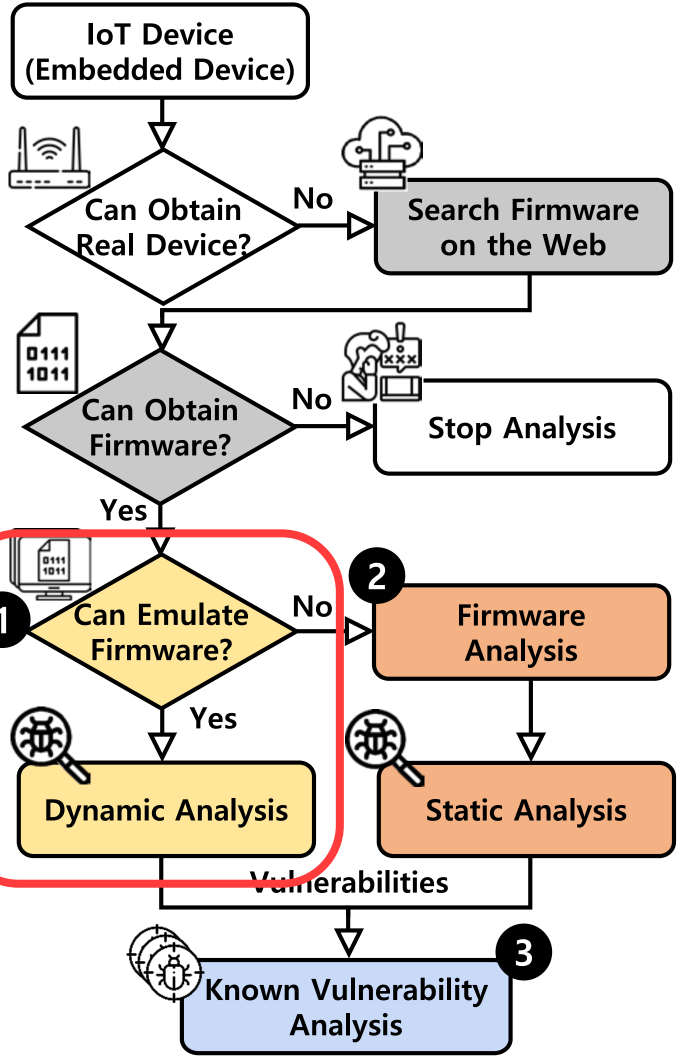
Properties	Wireless Routers, IP Cameras	Smartphone Baseband
# of Vendors	Numerous	A Few (Oligopoly)
Operating System	General Purpose OS (Linux)	No OS Abstraction
Firmware Structure	**Well-Known**	**Unknown**
# of Files in Firmware	Multiple Files	Monolithic
Functionality	Simple	Complex (Real-Time)
# of Peripherals	A Few	Multiple
Emulation	Feasible	Nearly Infeasible

Analysis Roadmap

1. Firmware Emulation Problem
   – Low emulation rate (16.3%)
   ➔ **Wireless routers, IP cameras**

2. Firmware Analysis Problem
   – Not scalable (≤ 10 images)
   ➔ **Smartphone baseband**

3. Known Vulnerability Analysis Problem
   – A few studies (≤ 10)
   ➔ **Both device categories**



Developing Heuristics for Firmware Emulation: Case study of Linux-based IoT Devices

Enabling Large-scale Emulation of IoT Firmware
with Heuristic Workarounds
IEEE Security & Privacy

Dongkwan Kim, Eunsoo Kim, Mingeun Kim
Yeongjin Jang, Yongdae Kim

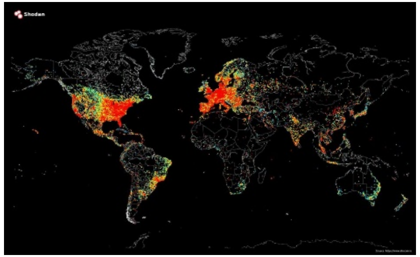
Extension of FirmAE
ACSAC 2020

(In)Security of Linux-Based IoT Devices

- **34.2 billion** embedded devices will be in use in 2025*
  - Wireless routers, IP cameras, …

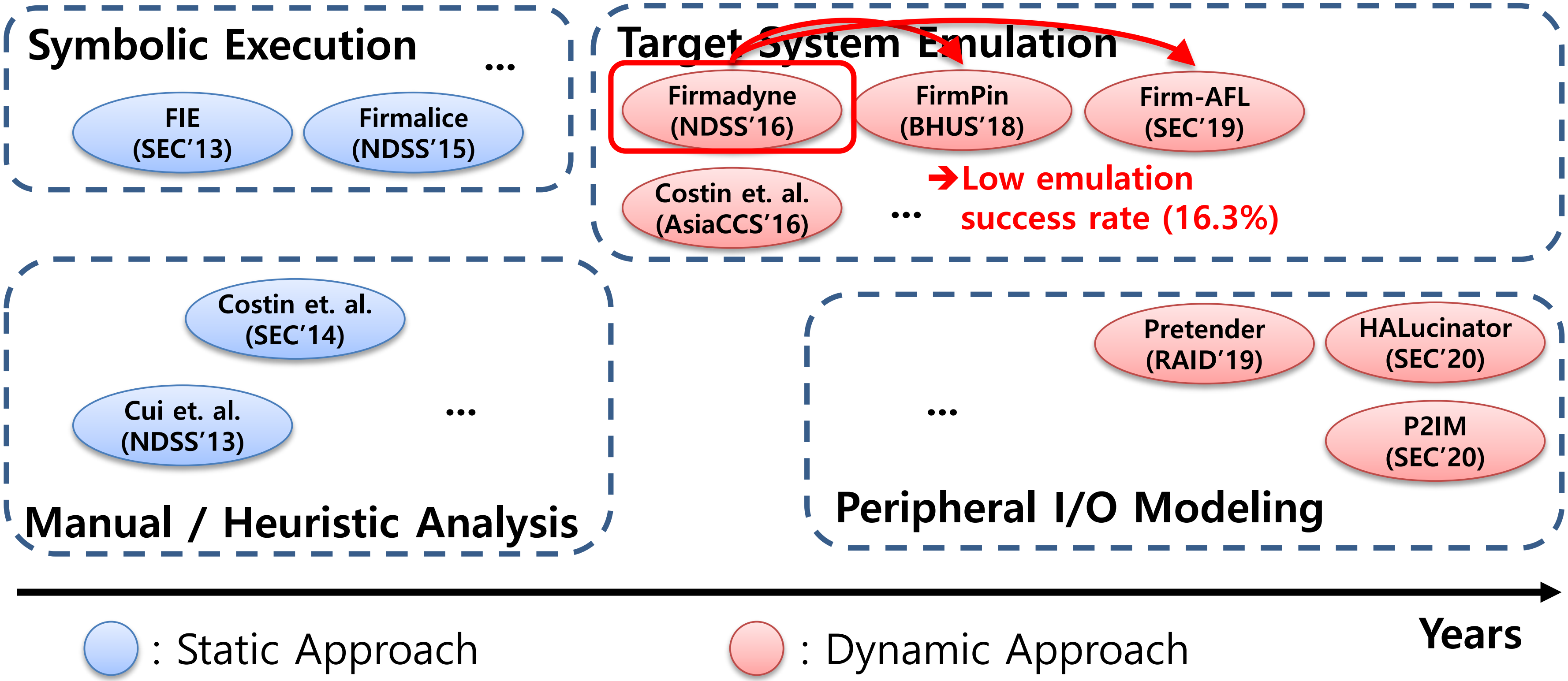
- Many **botnets** target IoT devices
  - Mirai (Aug. 2016)
  - Satori (Dec. 2017)
  - Crypto (May. 2018)
  - ECHOBOT (Dec. 2019)
  - New Mirai variant (July 2020, 2021~)
  - ➔ DDoS attacks: DynDNS (2016), GitHub (2018), …

- **Exposed to the Internet**, especially **web interfaces**
  - Shodan, ZoomEye
  - Over 30 exploits in Mirai variants



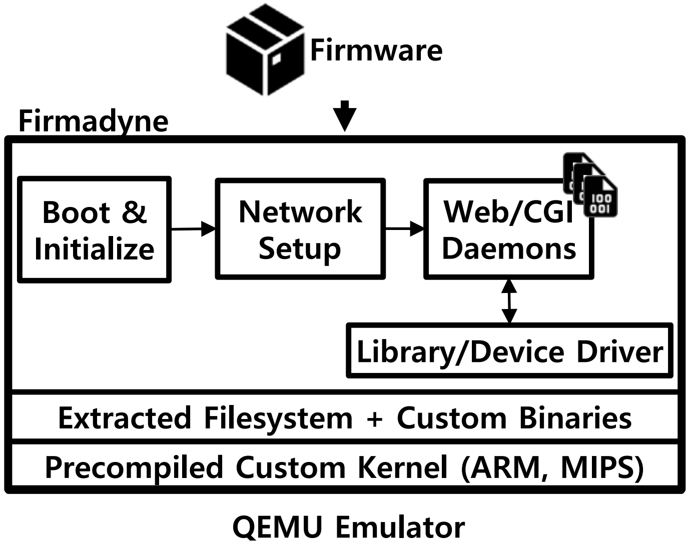
*https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/

Existing Analysis Approaches



Firmadyne: state-of-the-art firmware emulator

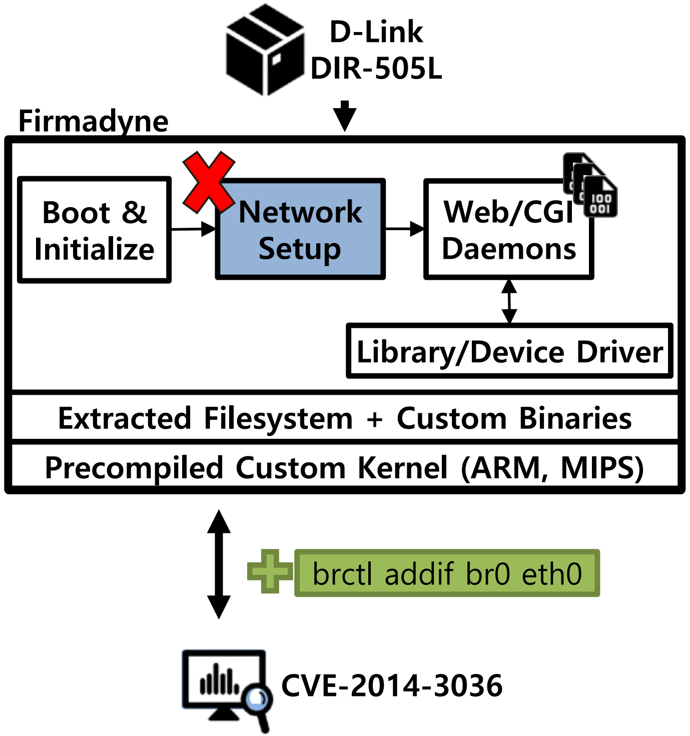
- Custom kernel and library
  - Hook system calls
  - Mimic NVRAM-related functions
    - *NVRAM: flash memory
- Emulating target firmware twice
  - Collect useful logs (IP address, device name)
  - Configure the system with the logs



Firmadyne can emulate only 183 of 1,124 (16.3%) firmware images for web services

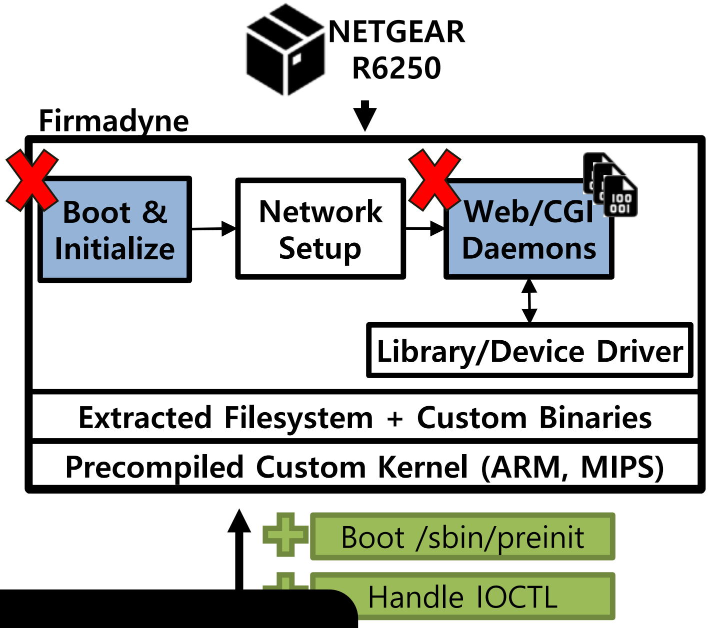
Motivating example: CVE-2014-3936

- Target
  - D-Link DIR-505L
- Symptom
  - Fails to configure network interface
- Possible causes
  - Access to unsupported peripherals
  - Retrieve unknown/improper values
- How to address
  - Forcibly set up the network interface



Motivating example: CVE-2017-5521

- Target
  - NETGEAR R6250
- Symptom
  - Fails to boot and run the web service
- Possible causes
  - Incorrect init program
  - Missing kernel module to handle IOCTL
- How to address
  - Set the correct
  - Add an IOCTL



-2017-5521

Simple heuristics are effective!

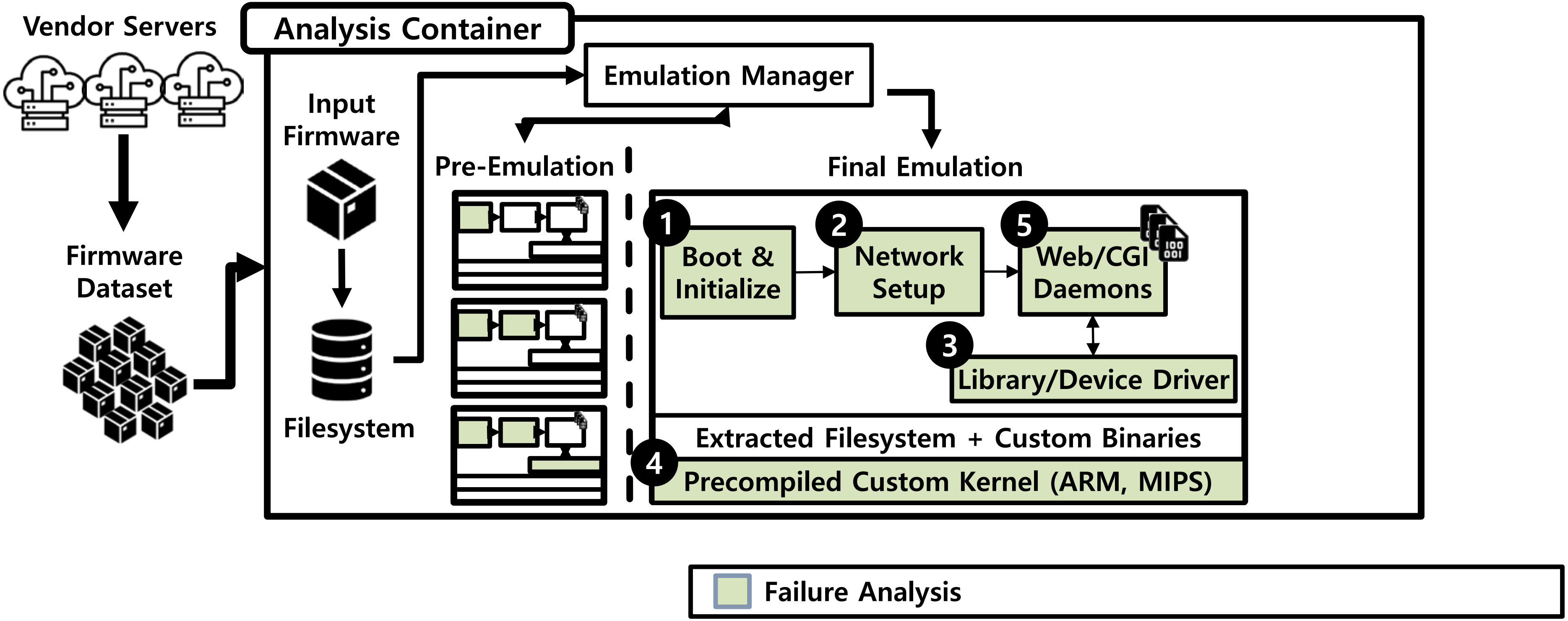
Our approach

- Goal
  - Run admin web services for dynamic security analysis

- Requirements
  - Emulated system should be reachable from the host
  - Web services should be available

- Approach
  - Investigate failure cases of Firmadyne
  - Develop heuristics to satisfy the emulation requirements

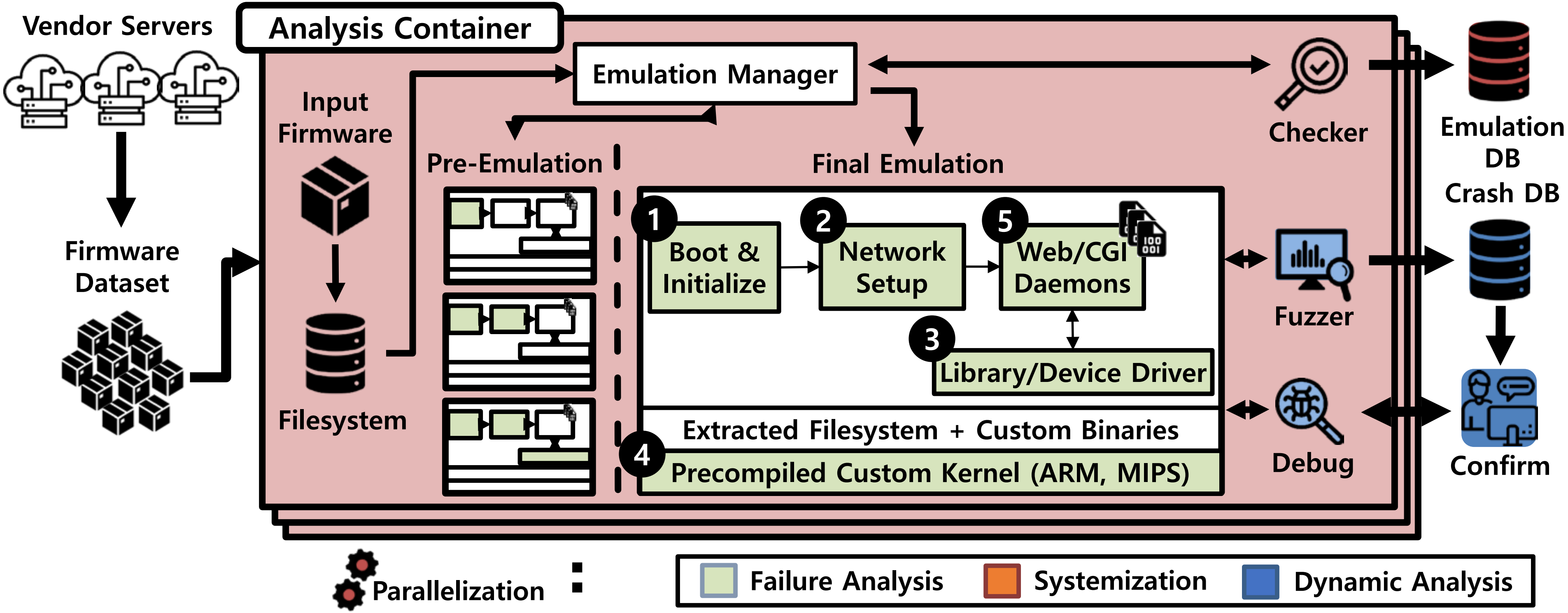
FirmAE overview



Examples of Developed Heuristics

Where	Problem	Heuristics
Boot	Missing files or directories	Extract path strings and create them (e.g., /var, /etc)
Library for Virtualization	Unknown configuration values	Search filesystem and original kernel (e.g., /etc/nvram.default)
Network	No network interface	Forcibly configure a default interface (e.g., eth0, 192.168.0.1)
Programs	Unexecuted web server	Forcibly run the server (e.g., run httpd)

FirmAE overview



Emulation Results (vs Firmadyne)

Wireless Routers

IP Cameras

Dataset	Vendor	Images	Firmadyne Web	FirmAE Web
AnalysisSet (**Outdated**)	D-Link	179	54 (30.17%)	167 (93.30%)
	NETGEAR	73	5 (6.85%)	59 (80.82%)
	TP-Link	274	30 (10.95%)	257 (93.80%)
Sub Total		**526**	**89 (16.92%)**	**483 (91.83%)**
LatestSet (**Latest**)	D-Link	58	17 (29.31%)	48 (82.76%)
	TP-Link	69	10 (14.49%)	54 (78.26%)
	NETGEAR	101	7 (6.93%)	79 (78.22%)
	TRENDnet	106	23 (21.70%)	63 (59.43%)
	ASUS	107	25 (23.36%)	62 (57.94%)
	Belkin	37	2 (5.41%)	22 (59.46%)
	Linksys	55	8 (14.55%)	44 (80.00%)
	Zyxel	20	0 (0%)	10 (50.00%)
Sub Total		**553**	**92 (16.64%)**	**382 (69.08%)**
CamSet (**Latest**)	D-Link	26	0 (0%)	17 (65.38%)
	TP-Link	6	0 (0%)	0 (0%)
	TRENDnet	13	2 (15.38%)	10 (76.92%)
Sub Total		**45**	**2 (4.44%)**	**27 (60.00%)**
Total		**1124**	**183 (16.28%)**	**892 (79.36%)**

x5

*Latest firmware images are checked as of Dec. 2018

Dynamic Analysis Results

- Dynamic security analysis
  - Known vulnerabilities
    - RouterSploit (set of known exploits)
    - **14 (Firmadyne) → 320 (FirmAE)**

Description	Total Vulns (Devices)
Information Leak	**8 (157)**
Command Injection	**23 (112)**
Authentication Bypass	**2 (5)**
Buffer Overflow	**5 (7)**

  - New vulnerabilities
    - RouterSploit + Simple custom fuzzer
    - **23 vulns from 95 latest devices (affecting 6 vendors)**

Conclusion and Lessons Learned

- Existing approaches build a generic firmware emulator without detailed analysis
  ➔ Low emulation rate

- Effectiveness of empirical analysis and heuristics
  - Successfully emulate firmware images (16.28% ➔ 79.36%)
  - Successfully transfer heuristics (old version ➔ latest version, routers ➔ IP cameras)
  - Help security analysis (known vulns: 14 ➔ 320, new vulns: 23)

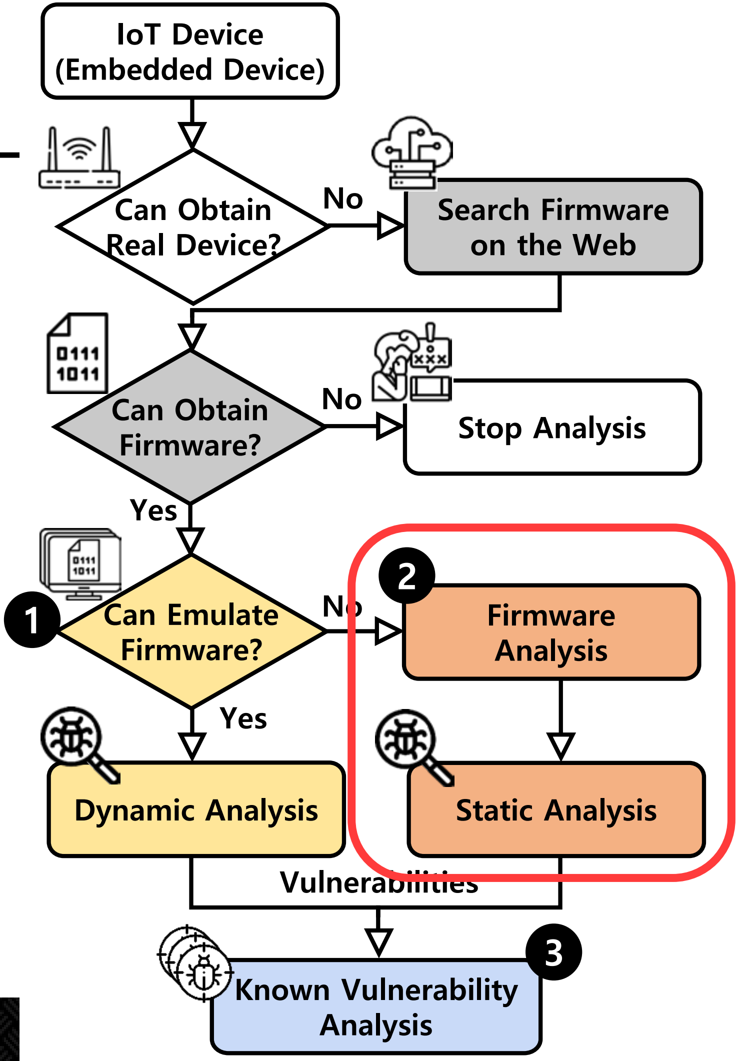
- Lessons learned
  - Developing/Systematizing heuristics are effective and necessary
  - Many IoT devices share similar code bases

Analysis Roadmap

1. Firmware Emulation Problem
   – Successful emulation (16.28% ➔ 79.36%)
   ➔ Wireless routers, IP cameras

2. Firmware Analysis Problem
   – Not scalable (≤ 10 images)
   ➔ **Smartphone baseband**

3. Known Vulnerability Analysis Problem
   – A few studies (≤ 10)
   ➔ **Both device categories**



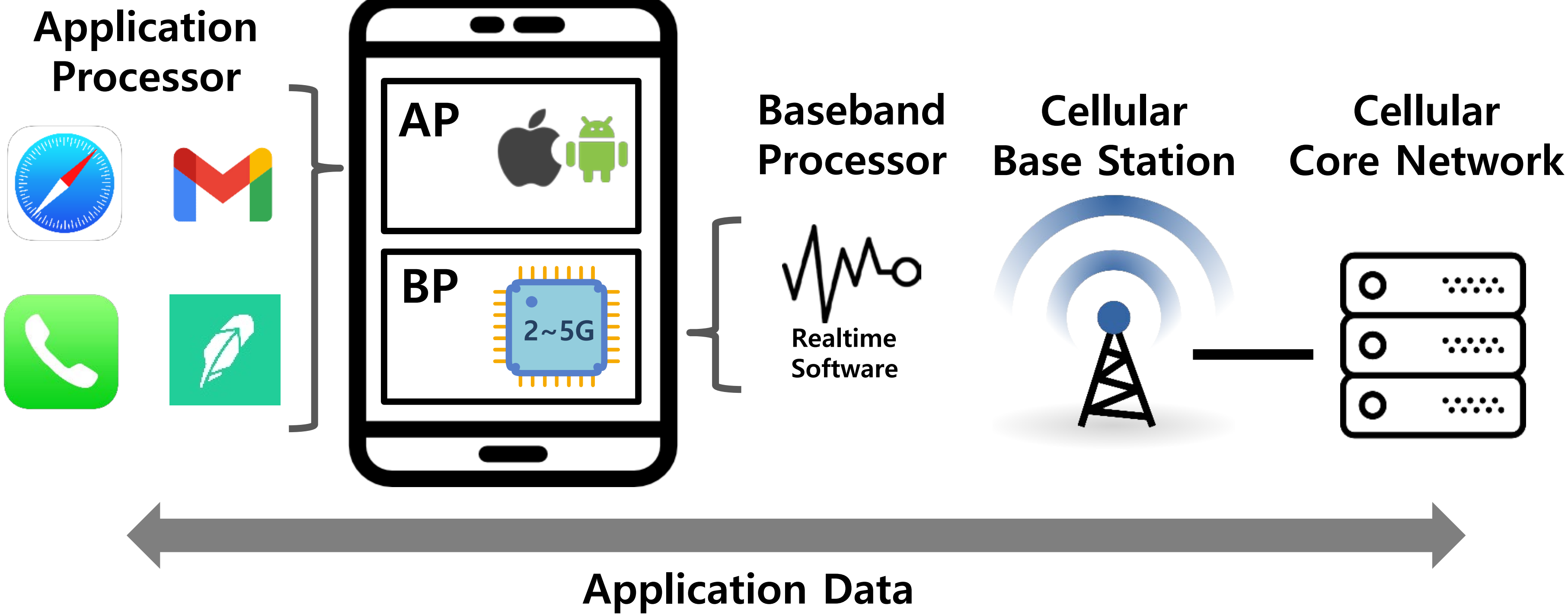
Developing Heuristics for Firmware Analysis: Case study of Smartphone Baseband

BaseSpec: Comparative Analysis of
Baseband Software and Cellular Specifications for L3 Protocols
NDSS 2021

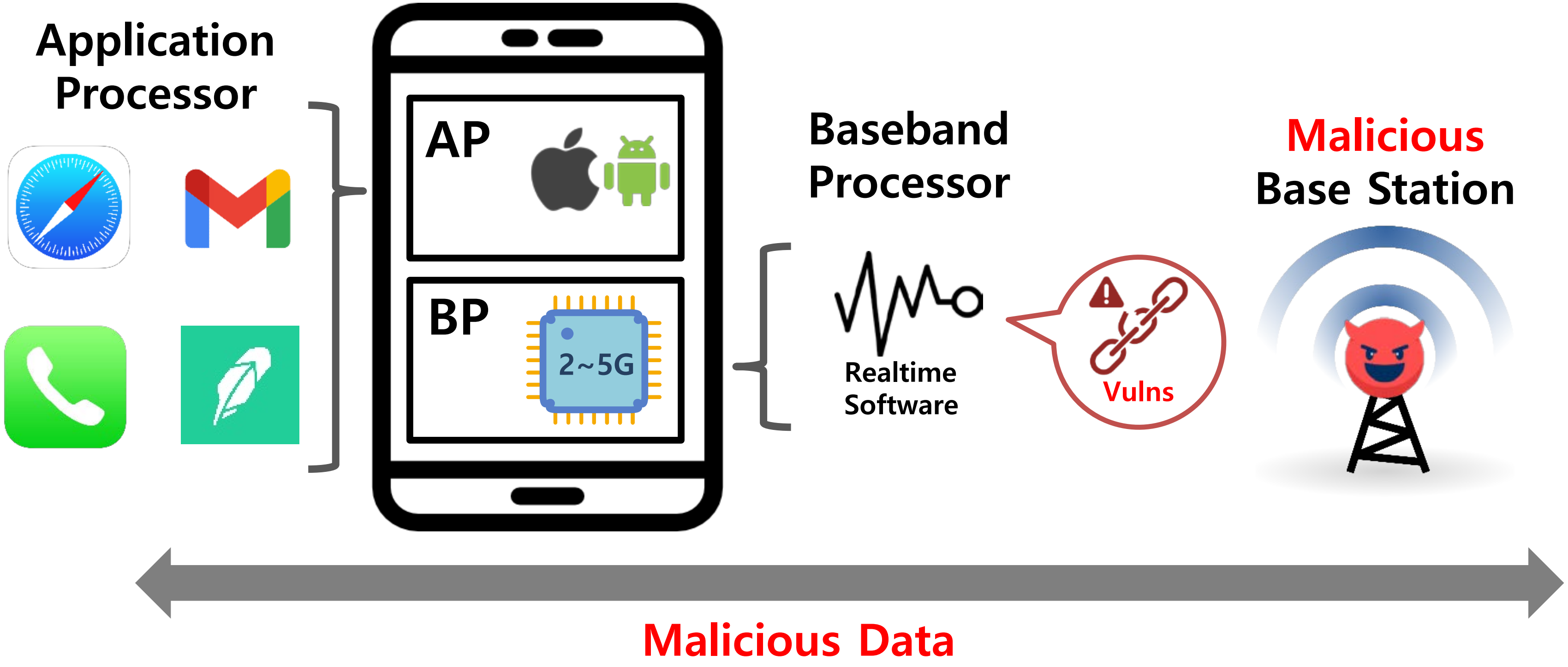
Eunsoo Kim*, **Dongkwan Kim***, CheolJun Park,
Insu Yun, Yongdae Kim

*: co-first author

Why Cellular Baseband?

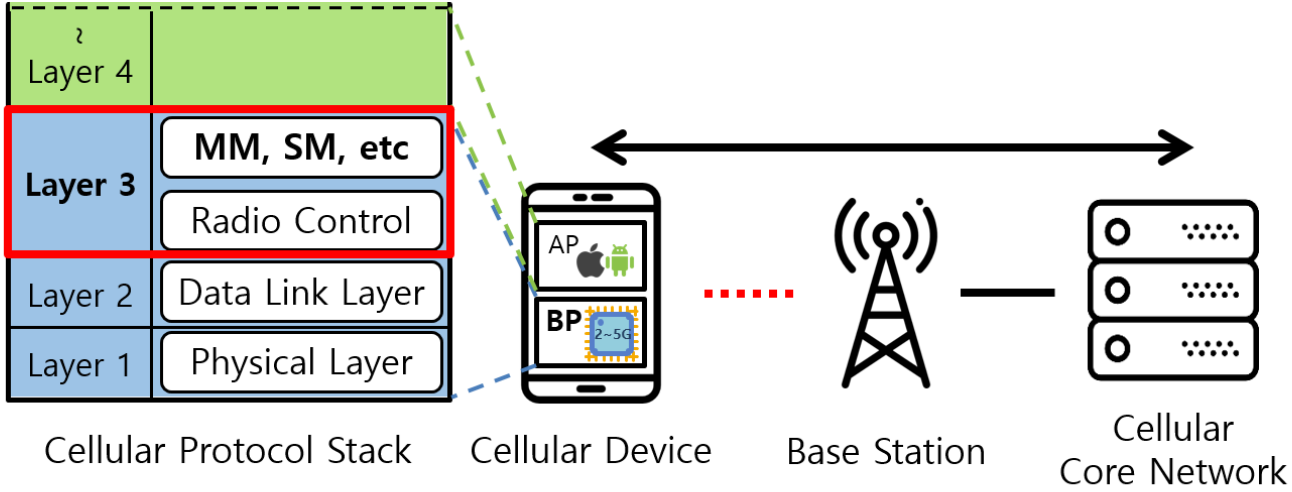


Why Baseband?

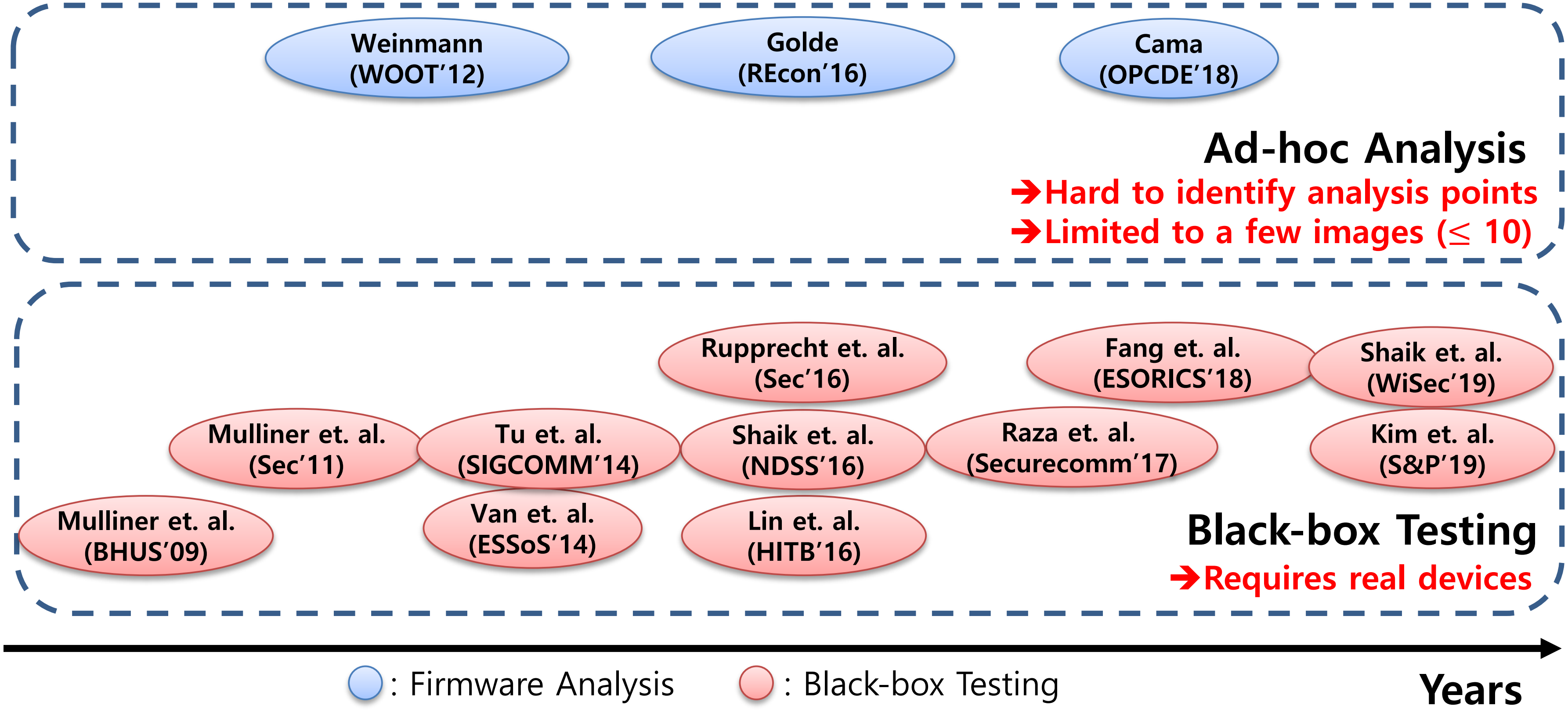


Baseband Manages Cellular Protocols

- Similar to OSI Model
- **Layer 3 (L3)** manages core procedures
  - Call Control, Mobility or Session Management, …
- Multiple **vulnerabilities** have been found in L3



Existing Analysis Approaches



Challenges in Baseband Firmware Analysis

- ❖ **Numerous** functions (over 90K) in a single firmware image (over 30MB)
- ❖ **Non-trivial, Real-time** operations (e.g., mobility, session, call, interrupts, …)
- ❖ Vendors **do not release** implementation details

➔ How can we analyze firmware structure?

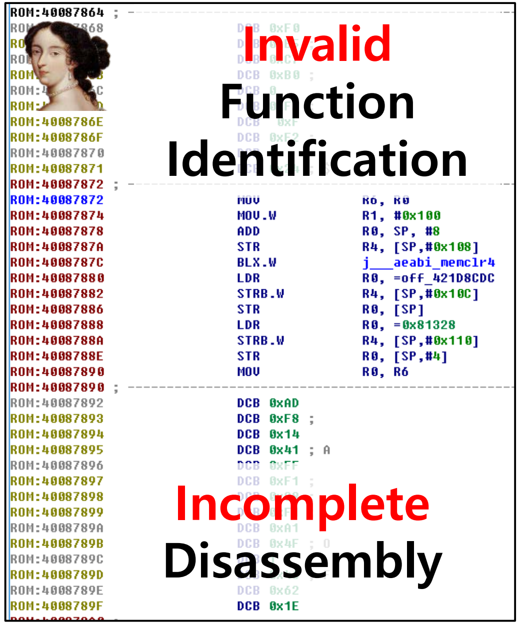
- ❖ **Diverse** firmware versions and device models

➔ How can we scale up the analysis?

Motivating Example: IDA Pro Analysis

- IDA Pro (state-of-the-art tool) fails to identify functions
  - Initial: **only 450** functions
  - Actual: over 90,000 functions

- Problems
  - IDA **cannot support** ARM memory layout setup
    - Memory layout should be set first
  - IDA **cannot analyze** indirect calls
    - Interrupt tables, function pointers, …

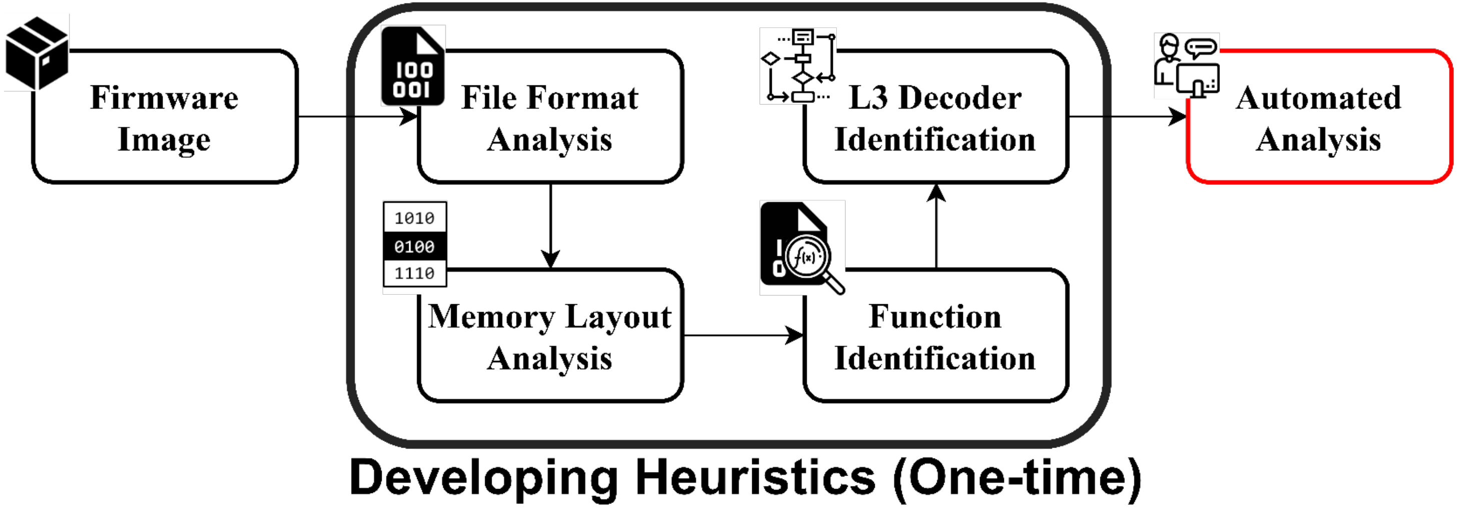


How to improve the performance?

Our Approach

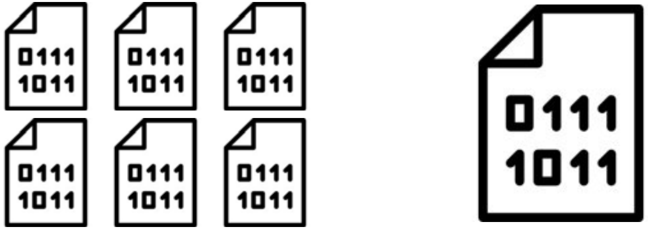
- Goal
  - Identify target functions for further security analysis
    - L3 decoder functions
- Requirements
  - Should load firmware into a correct memory layout
  - Should identify functions in firmware correctly
  - Should detect target functions among the identified functions
- Approach
  - Investigate firmware manually
  - Develop heuristics to satisfy the firmware analysis requirements

Firmware Analysis Overview



Analyzing Firmware File Formats

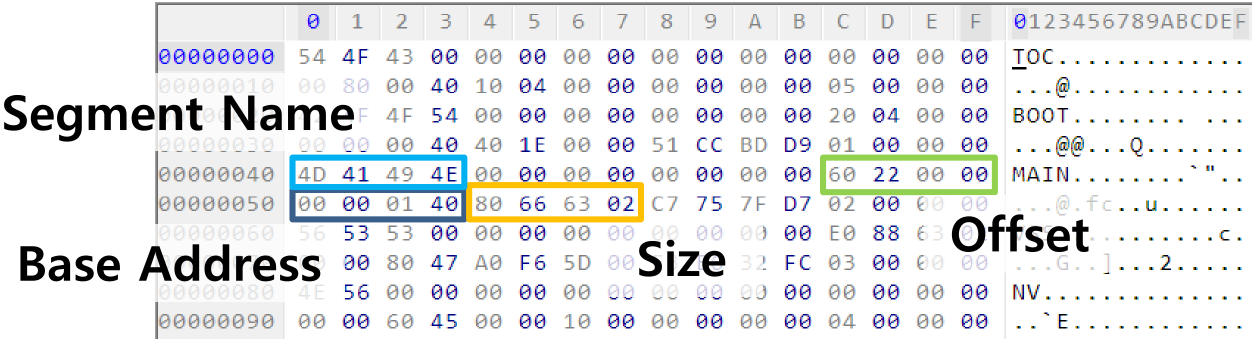
- Baseband firmware
  - Downloaded from a 3rd party website
  - Single binary **over 30 MB**
  - **Unknown format**



Typical Firmware
(Multiple Binaries)

Baseband Firmware
(Single Binary)

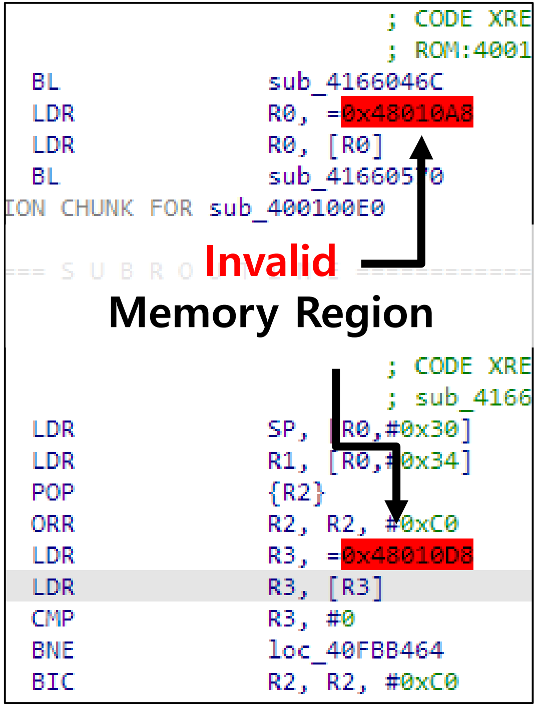
- Leverage binary analysis's heuristic knowledge
  - 4-byte integers often represent a base address, size, or offset



Memory Layout Analysis

- Observation
  - Access **invalid** memory regions
- Possible causes
  - Partial firmware
  - **Special memory layout setup**

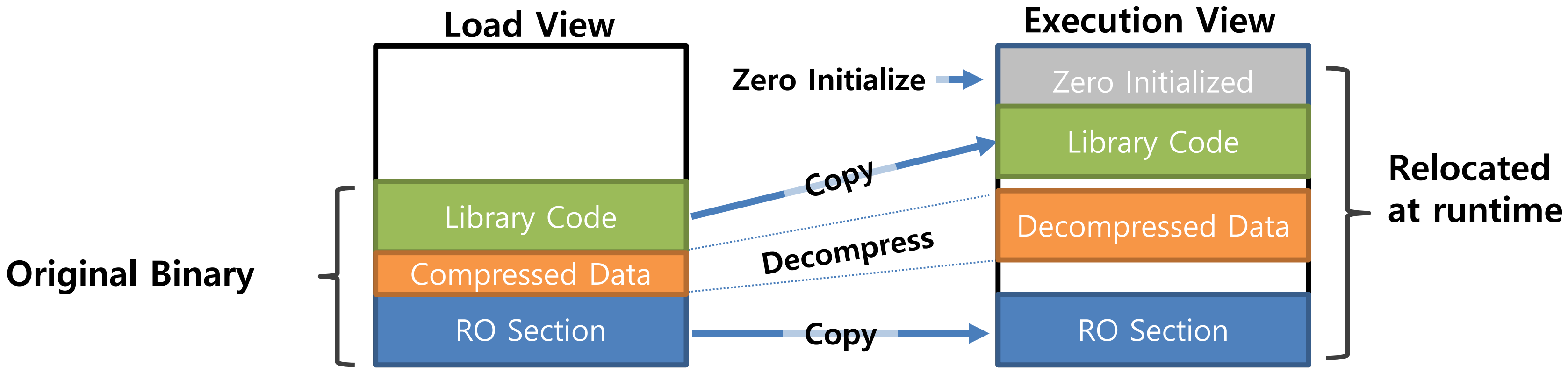
➔ Eventually figured out "**scatter-loading**"



Scatter-Loading

- Runtime feature in ARM-based embedded devices
- Memory regions are relocated at runtime
  - Copy, Decompress, Zero-initialize memory regions

➔ **None of existing binary analysis approaches considered scatter-loading**



Scatter-Loading Heuristics

- ❖ Observation
  - – A memory layout is defined in source files
  - ➔ Linker inserts **pre-defined table**

```
LOAD 0x8000
{
    EXEC_ROM +0     { *(+RO) }
    RAM    +0x1000 { *(+RW,+ZI) }
    HEAP   +0x2000 EMPTY 0x100 {}
    STACK +0x3000 EMPTY 0x400 {}
}
```

- ❖ Approach
  - – Find scatter-loading table
  - – Detect scatter-loading functions
  - – Emulate scatter-loading operations

Src Dst Size Function

```
__scatter_table DCD unk_4246B454, unk_44260000, dword_4, __scatterload_copy
                DCD sub_4246B458, sub_4000000, unk_1D8F4, __scatterload_copy
                DCD unk_42488D4C, unk_4800C30, off_28, __scatterload_copy
                DCD off_42488D74, dword_4800C58, dword_3A4, __scatterload_decompress
                DCD unk_42488E44, unk_100000, loc_12600, __scatterload_decompress
                DCD unk_42489094, unk_45700040, 0x1AACC0, __scatterload_decompress
                DCD unk_424ABD28, word_46100000, loc_58, __scatterload_copy
                DCD unk_424ABD80, unk_46800000, loc_1030C, __scatterload_decompress
                DCD loc_424ABF90, sub_47400000, unk_10FE68, __scatterload_copy
                DCD loc_425BBDF8, dword_47700000, unk_1020, __scatterload_copy
                DCD unk_425BCE18, dword_4B700000, dword_4, __scatterload_copy
                DCD unk_425BCE1C, dword_4B700004, 0x889A0, __scatterload_copy
```

➔ **Applicable** to other ARM-based embedded devices

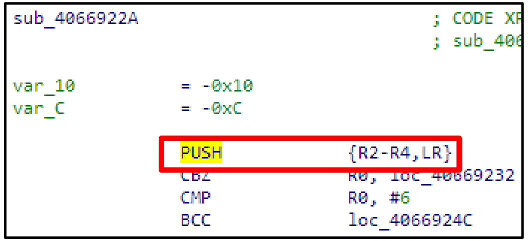
Function Boundary Identification

- Baseband is a complex embedded system
  - Numerous indirect calls
    - Interrupt tables, function pointers, …
  - ARM mode (32-bit) + THUMB mode (16-bit)
  - Data appears "in" the code section
  - ➔ **More difficult** than traditional function identification

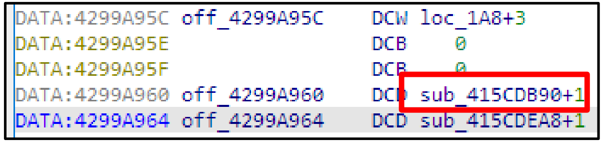
- Existing approaches
  - ByteWeight (SEC'14), Shin et. al. (SEC'15), Andriesse et. al. (SEC'16, SP'17), …
  - ➔ Most approaches **do not consider** **ARM/THUMB co-existing** binaries
  - ➔ State-of-the-art tool (IDA) **cannot analyze** **(450 among 90K funcs are detected)**

Function Identification Heuristics

- ❖ Identify frequent function prologues
  - – Linear sweep as proposed in Andriesse et. al. (SEC'16)
  - – Functions often start with a "PUSH" instruction
  - – Analyze PUSH instruction
    - ▪ Different byte code in ARM/Thumb mode
    - ▪ Should contain LR register
    - ▪ Should not contain SP, PC register
    - ▪ Should contain temporary registers (e.g., R2-R4)



- ❖ Analyze Thumb-mode function pointers
  - – Thumb mode function call (pointer+1)
  - – Find thumb mode function pointer



Function Identification Heuristics

- Utilize debug information (logging messages)
  - Developers often include debug information
  - Analyze customized debug structure

Magic Value

Debug msg

Filename

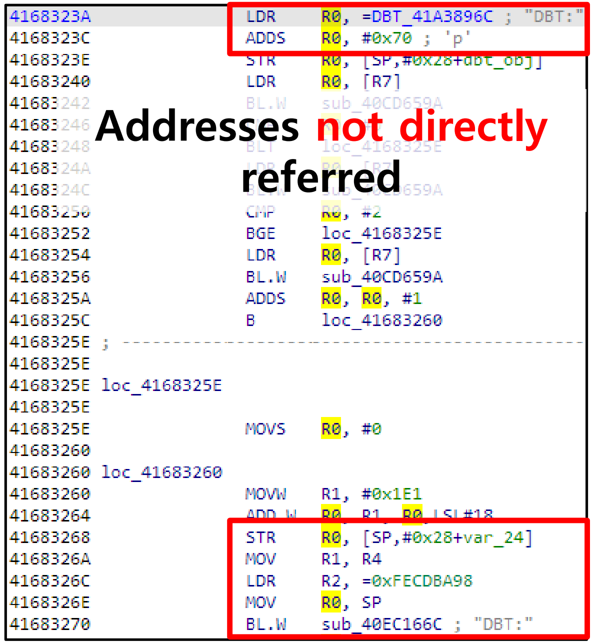
```
DCB "DBT:"                    ; DATA XREF: NAS_Parsing+74↑o
                              ; sub_4047B972:off_4047BA28↑o
DCD 4
DCD 1
DCD 0xFECDBA98
DCD aWarnDecodeErro           ; "Warn>Decode Error: 0x%x"
DCD 0xC28
DCD asc_4156E7E0              ; "../../../CALPSS/LteL3/LteSae/SAEMM/Code"...
```

- An instruction candidate has an operand that refers to debug information
  - ➔ **Should be a part of a function**

```
CBZ             R4, loc_4168327C
LDR             R0, =DBT_41A3896C ; "DBT:"
ADDS            R0, #0x70 ; 'p'
STR             R0, [SP,#0x28+var_28]
```

Identifying L3 Decoder Functions

- Utilize debug information (logging messages)
  - Commonly used in analyzing stripped binaries
- Search target keywords
  - "Decode", "L3", "EMM", …
- Implement simple slicer
  - Debug information is **not directly** referred
    - Cache optimization
  - Slice forward to compute correct addresses
  - Analyze target keywords



➔ **Identify functions of interest (i.e., L3 decoder functions)**

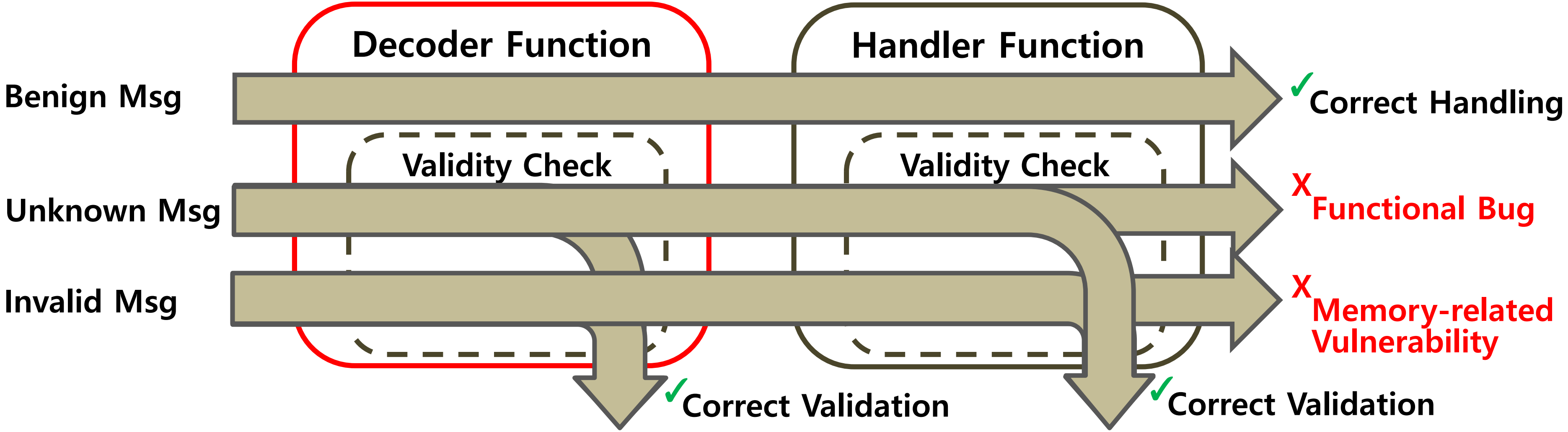
Evaluation

- Implemented as an IDA Pro Plugin
  - Function boundary identification
  - L3 decoder function identification

	Model	Build Date	Firmware Size (MB)	# of Funcs in Default IDA Pro	# of Funcs after Applying Heuristics	L3 Decoder Address
Latest 8 Images	Model 1	May/2020	44	452	91043	0x4113ed5a
	Model 2	May/2020	44	3601	89989	0x4117e646
	Model 3	May/2020	43.8	446	89893	0x4114ca72
	...					
	Model 9	Apr/2020	37	386	66663	0x4100b0b4
Oldest 8 Images	Model 1	Apr/2019	43.4	457	89789	0x411c03aa
	Model 2	Feb/2019	43.3	450	88209	0x4127b8ca
	Model 3	Feb/2019	43.1	450	80268	0x4124810e
	...					
	Model 9	Apr/2016	36.8	377	61714	0x41019c00

Security Analysis

❖ Analyze manually from the detected decoder functions



➔ 5 functional bugs, 4 memory-related bugs (2 RCEs) affecting 33 messages

Conclusion and Lessons Learned

- Existing approaches
  - Black-box testing ➔ **Need physical devices**
  - Ad-hoc firmware analysis ➔ **Not scalable (≤ 10 images)**

- Effectiveness of empirical analysis and developing/systematizing heuristics
  - Successfully identify function boundaries (595 ➔ 73,874 on avg., 124 times)
  - Successfully detect target functions (0 false positive)
  - Help security analysis (9 new bugs, including 2 RCEs)

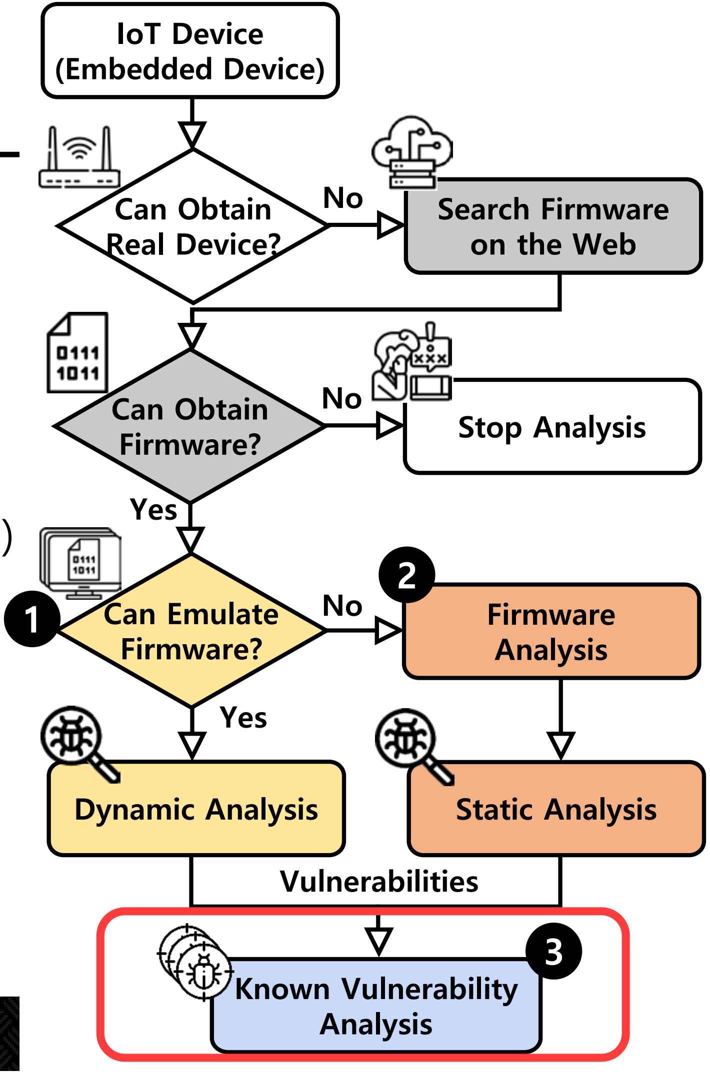
- Lessons learned
  - Developing/Systematizing Heuristics are effective and necessary
  - Baseband devices within a vendor share similar code

Analysis Roadmap

1. Firmware Emulation Problem
   – Successful emulation (16.28% ➔ 79.36%)
   ➔ Wireless routers, IP cameras

2. Firmware Analysis Problem
   – Successful analysis (595 funcs ➔ 73,874 funcs)
   ➔ Smartphone baseband

3. Known Vulnerability Analysis Problem
   – A few studies (≤ 10)
   ➔ **Both device categories**



Finding Known (Similar) Vulnerability in IoT Devices with BCSA

Revisiting Binary Code Similarity Analysis using Interpretable Feature Engineering and Lessons Learned

IEEE Transactions on Software Engineering (major revision, under review)

Dongkwan Kim, Eunsoo Kim,
Sang Kil Cha, Sooel Son, and Yongdae Kim

Known Vulnerability Issues in IoT Devices

- Example vulnerability: CVE-2018-10106
  - Permission bypass in "cgibin" reveals users' private key
  - Parameter can be over-written with a newline character (0x0a)

```
response = self.http_request(
    method="POST",
    path="/getcfg.php?A=A%0a_POST_SERVICES%3dDEVICE.ACCOUNT%0aAUTHORIZED_GROUP%3d1",
    headers=headers
)
```

  - Still appears in newer device versions (D-Link)
    - CVE-2018-10106, CVE-2019-17506, CVE-2019-20213, CVE-2020-9376
  - Appears in different venders (TRENDnet)
    - CVE-2018-7034

- Potential reasons
  - Improper version/update management
  - Copy and paste buggy code

Known Vulnerability Analysis

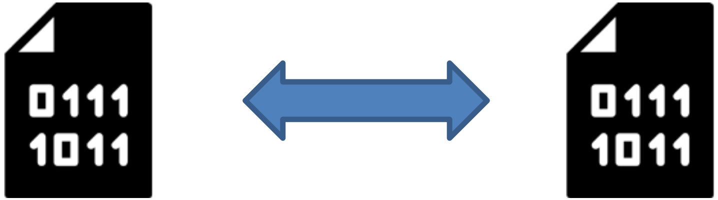
- Dynamic analysis
  - – Build PoC exploits and run them
  - ☞ Require successful emulation
    - ☞ Architecture challenges (e.g., ARM, MIPS, PowerPC, Hexagon, …)
    - ☞ Dependency issues in peripherals (e.g., Camera, LED, MMIO access, …)
  - ☞ Require time for emulation and testing

Increasing Scalability
Preserving Low False Positive Rate

- Static analysis
  - – Match known signatures
  - – Leverage Binary code similarity analysis (BCSA)
  - ➔ **Apply BCSA to find same/similar vulnerabilities in newer devices**

Binary Code Similarity Analysis

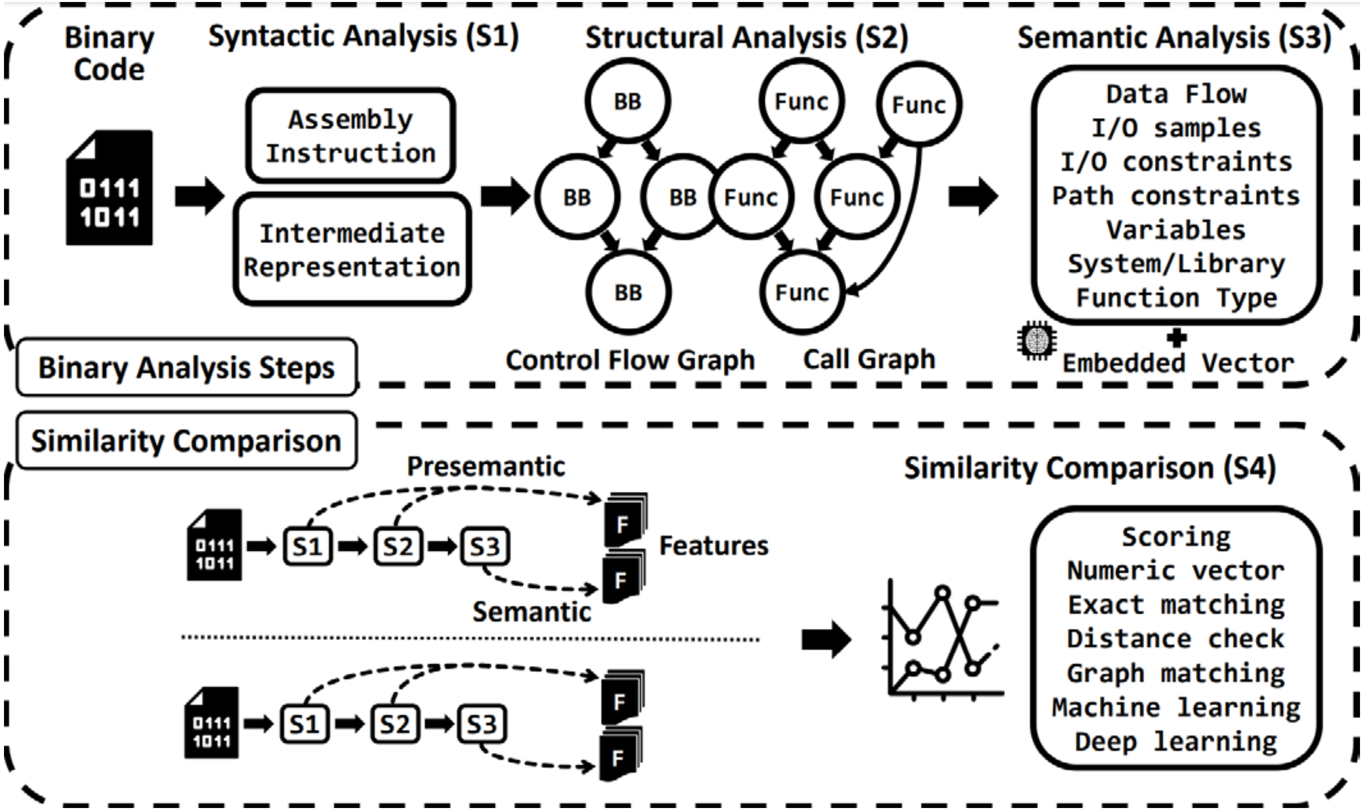
- Binary code similarity analysis (BCSA)



Known Binary Code A | **Unknown Binary Code B**

- Popular tasks
  - Malware detection
  - Plagiarism detection
  - Authorship identification
  - **Vulnerability discovery**
- Target
  - Architecture (e.g., x86 -> ARM)
  - Compiler (e.g., gcc -> clang)
  - Optimization (e.g., O1 -> O3)
  - Obfuscation (e.g., LLVM-Obfuscator)

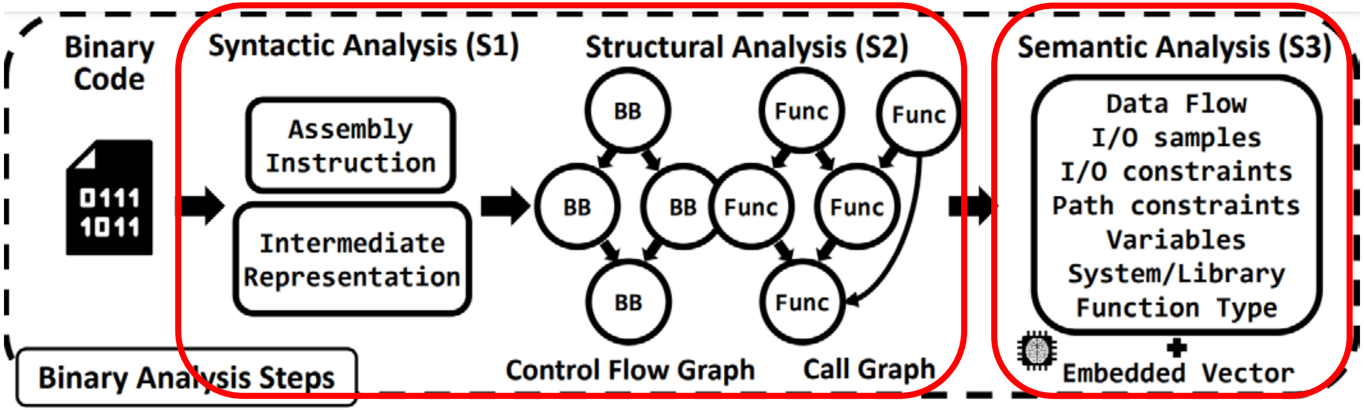
BCSA Workflow



Similarity Score: 0.0 ~ 1.0

SysSec
System Security Lab

BCSA Workflow



Pre-Semantic Features | **Semantic Features**

❖ **Numeric features**
- BB-level: # of instructions, …
- CFG-level: # of basic blocks, …
- CG-level: # of callers, …

❖ **Non-Numeric features**
- Raw bytes: N-gram, …
- Instructions: Assembly, IR, …
- Functions: Name, …

❖ **Semantic features**
- Symbolic constraints
- Runtime behavior (memory values, …)
- Program slices (data flow, …)
- Embedded vector (machine learning)
- …

*Revisiting Binary Code Similarity Analysis using Interpretable Feature Engineering and Lessons Learned

2014	2015	2016	2017	2018	2019	2020
TEDEM	Multi-k-MH	DiscovRE	GitZ	FirmUp	InnerEye	DeepBinDiff
Tracy		Esh	Xmatch	SANER18	Asm2Vec	ImOpt
CoP		Genius	Gemini	BinGo-E	SAFE	ACCESS20
LoPD		BinGo	BinSim	WSB	BAR19i	Patcheko
BLEX		MockingBird	IMF-SIM	BinMatch	BAR19ii	
BinClone		Kam1n0	CACompare	MASES18	FuncNet	
		BinDNN	BinSign	Zeek		
			BinSequence	BinArm		
			ASE17	αDiff		
				VulSeeker		

Studied 43 papers in 27 venues

BCSA Features in Previous Literature

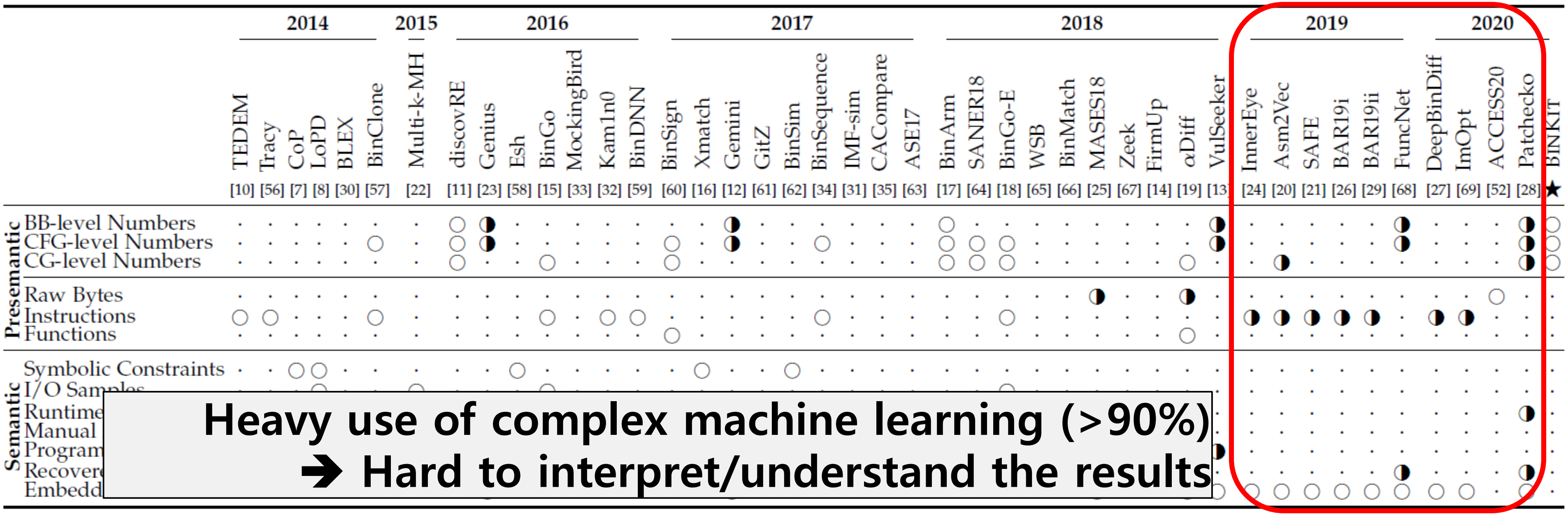
		2014						2015	2016							2017									2018										2019						2020				
		TEDEM	Tracy	CoP	LoPD	BLEX	BinClone	Multi-k-MH	discovRE	Genius	Esh	BinGo	MockingBird	Kam1n0	BinDNN	BinSign	Xmatch	Gemini	GitZ	BinSim	BinSequence	IMF-sim	CACompare	ASE17	BinArm	SANER18	BinGo-E	WSB	BinMatch	MASES18	Zeek	FirmUp	αDiff	VulSeeker	InnerEye	Asm2Vec	SAFE	BAR19i	BAR19ii	FuncNet	DeepBinDiff	ImOpt	ACCESS20	Patchecko	BinKit
		[10]	[56]	[7]	[8]	[30]	[57]	[22]	[11]	[23]	[58]	[15]	[33]	[32]	[59]	[60]	[16]	[12]	[61]	[62]	[34]	[31]	[35]	[63]	[17]	[64]	[18]	[65]	[66]	[25]	[67]	[14]	[19]	[13]	[24]	[20]	[21]	[26]	[29]	[68]	[27]	[69]	[52]	[28]	★
Presemantic	BB-level Numbers	·	·	·	·	·	·	·	○	◑	·	·	·	·	·	·	·	◑	·	·	·	·	·	·	○	·	·	·	·	·	·	·	·	◑	·	·	·	·	·	◑	·	·	·	◑	○
	CFG-level Numbers	·	·	·	·	·	○	·	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	○
	CG-level Numbers	·	·	·	·	·	·	·	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	○
	Raw Bytes	·	·	·	·	·	·	·	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	·
	Instructions	○	○	·	·	·	○	·	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	·
	Functions	·	·	·	·	·	·	·	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	·
Semantic	Symbolic Constraints	·	·	○	○	·	·	·	·	·	○	·	·	·	·	·	○	·	·	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
	I/O Samples	·	·	·	○	·	·	○	·	·	·	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
	Runtime Behavior	·	·	·	·	○	·	·	·	·	·	·	○	·	·	·	·	·	·	○	·	○	○	○	·	·	○	○	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	◑	·
	Manual Annotation	·	·	·	·	·	·	·	·	·	·	○	·	·	·	○	·	·	·	○	·	·	·	·	·	·	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
	Program Slices, PDG	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○	·	·	·	·	·	·	·	·	·	·	·	○	○	·	◑	·	·	·	·	·	·	·	·	·	·	·
	Recovered Variables	·	·	·	·	·	·	·	○	◑	·	·	·	·	·	○	·	◑	·	·	·	·	·	·	○	○	○	·	·	·	·	·	·	·	·	·	·	·	·	◑	·	·	·	◑	·
	Embedded Vector	·	·	·	·	·	·	·	·	○	·	·	·	·	·	·	·	○	·	·	·	·	·	·	·	·	·	·	·	○	·	·	○	○	○	○	○	○	○	○	○	○	·	○	·

Heavy use of complex semantic features (>84%)
➔ No clear justification

◑: used with machine learning

*Revisiting Binary Code Similarity Analysis using Interpretable Feature Engineering and Lessons Learned

BCSA Features in Previous Literature



◑: used with machine learning

*Revisiting Binary Code Similarity Analysis using Interpretable Feature Engineering and Lessons Learned

BCSA Dataset in Previous Literature

		Source*		Architecture*								Optimization*					Compiler†*													Extra				Info.		
Year	Tool [Paper]	Binaries	Firmware	x86	x64	arm	aarch64	mips	mips64	mipseb	mips64eb	O0	O1	O2	O3	Os	GCC 3	GCC 4	GCC 5	GCC 6	GCC 7	GCC 8	Clang 3	Clang 4	Clang 5	Clang 6	Clang 7	etc.	Total #	Noinline	PIE	LTO	Obfus.	Code	Dataset	IDA
2014	TEDEM [10]	14	·	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○
	Tracy [56]	(115)	·	·	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○	·	○
	CoP [7]	(214)	·	·	·	·	·	·	·	·	·	○	○	○	○	○	·	1	·	·	·	·	·	·	·	·	·	1	2	·	·	·	○	·	·	○
	LoPD [8]	48	·	○	·	·	·	·	·	·	·	○	○	○	○	○	·	1	·	·	·	·	·	·	·	·	·	1	2	·	·	○	○	·	·	·
	BLEX [30]	1,140	·	·	○	·	·	·	·	·	·	○	○	○	○	·	·	1	·	·	·	·	1	·	·	·	·	1	3	·	·	·	·	·	·	○
	BinClone [57]	90	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○	·	○
2015	Multi-k-MH [22]	60	6	○	·	○	·	○	·	·	·	○	○	○	○	·	·	2	·	·	·	·	1	·	·	·	·	·	3	·	·	·	·	·	·	○
2016	discovRE [11]	593	3	○	·	○	·	○	·	·	·	○	○	○	○	○	·	1	·	·	·	·	1	·	·	·	·	2	4	○	·	·	·	○	◑	○
	Genius [23]	(7,848)	8,128	○	·	○	·	○	·	·	·	○	○	○	○	·	·	2	·	·	·	·	1	·	·	·	·	·	3	·	·	·	·	·	·	○
	Esh [58]	(833)	·	·	○	·	·	·	·	·	·	·	·	·	·	·	·	3	·	·	·	·	2	·	·	·	·	2	7	·	·	·	·	○	○	○
	BinGo [15]	(5,143)	·	○	○	○	·	·	·	·	·	○	○	○	○	·	·	3	·	·	·	·	1	·	·	·	·	1	5	·	·	·	·	·	·	○
	MockingBird [33]	80	·	○	·	○	·	○	·	·	·	○	·	○	○	·	·	1	·	·	·	·	1	·	·	·	·	·	2	·	·	·	·	·	·	○
	Kam1n0 [32]	96	·	○	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○	·	○
	BinDNN [59]	2,064	·	○	○	○	·	·	·	·	·	○	○	○	○	·	·	1	·	·	·	·	·	·	·	·	·	1	2	·	·	·	·	·	·	○
2017	BinSign [60]	(31)	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	2	2	·	·	·	○	·	·	○
	Xmatch [16]	72	1	○	·	·	·	○	·	·	·	·	·	·	·	·	·	2	·	·	·	·	1	·	·	·	·	·	3	○	·	·	·	·	·	○
	Gemini [12]	18,269	8,128	○	·	○	·	○	·	·	·	○	○	○	○	·	·	·	1	·	·	·	·	·	·	·	·	·	1	·	·	·	·	·	·	○
	GitZ [61]	44	·	·	○	·	○	·	·	·	·	○	○	○	○	○	·	3	·	·	·	·	2	1	·	·	·	·	6	·	·	·	·	·	·	·
	BinSim [62]	1062	·	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○	·	·	○
	BinSequence [34]	(1,718)	·	·	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○
	IMF-sim [31]	1,140	·	·	○	·	·	·	·	·	·	○	·	○	○	·	·	1	·	·	·	·	1	·	·	·	·	1	3	·	·	·	○	·	·	·
	CACompare [35]	72	·	○	·	○	·	○	·	·	·	○	·	○	○	·	·	1	·	·	·	·	1	·	·	·	·	·	2	·	·	·	·	·	·	○
	ASE17 [63]	55	·	○	○	·	·	·	·	·	·	○	·	○	○	·	·	1	·	·	·	·	1	·	·	·	·	·	2	·	·	·	○	·	·	·
2018	BinArm [17]	·	2,628	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○
	SANER18 [64]	7	·	○	○	·	·	·	·	·	·	·	·	·	·	·	·	1	1	1	·	·	1	·	·	·	·	1	5	·	·	·	·	·	·	○
	BinGo-E [18]	(5,145)	·	○	○	○	·	·	·	·	·	○	○	○	○	·	·	3	·	·	·	·	1	·	·	·	·	1	5	·	·	·	·	·	·	○
	WSB [65]	(173)	·	·	·	·	·	·	·	·	·	·	○	○	○	·	·	1	·	·	·	·	1	·	·	·	·	·	2	·	·	·	○	·	·	·
	BinMatch [66]	80	·	○	·	·	·	·	·	·	·	○	·	○	○	·	·	1	·	·	·	·	1	·	·	·	·	·	2	·	·	·	○	·	·	○
	MASES18 [25]	47	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○	·	·	·
	Zeek [67]	(20,680)	·	○	·	○	○	·	·	·	·	○	○	○	○	○	·	3	·	·	·	·	4	1	·	·	·	2	10	·	·	·	·	·	·	·
	FirmUp [14]	·	2,000	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○
	αDiff [19]	(69,989)	2	○	○	○	·	·	·	·	·	○	○	○	○	·	·	2	1	·	·	·	2	·	·	·	·	·	5	·	·	·	·	◑	◑	○
	VulSeeker [13]	(10,512)	4,643	○	○	○	○	○	○	·	·	○	○	○	○	·	·	1	1	·	·	·	·	·	·	·	·	·	2	·	·	·	·	○	·	○
2019	InnerEye [24]	(844)	·	·	○	○	·	·	·	·	·	·	○	○	○	·	·	·	·	·	·	·	·	·	·	1	·	·	1	·	·	·	·	◑	◑	·
	Asm2Vec [20]	68	·	·	○	·	·	·	·	·	·	○	○	○	○	·	·	1	1	·	·	·	2	·	·	·	·	·	4	·	·	·	○	○	·	○
	SAFE [21]	(5001)	·	·	○	○	·	·	·	·	·	○	○	○	○	·	1	3	1	1	1	·	2	1	1	1	·	·	12	·	·	·	·	○	◑	○
	BAR19i [26]	(804)	·	·	○	○	·	·	·	·	·	·	○	○	○	·	·	·	·	·	·	·	·	·	·	1	·	·	1	·	·	·	·	○	·	·
	BAR19ii [29]	(11244)	·	○	○	○	·	·	·	·	·	○	○	○	○	·	1	3	1	·	·	·	2	1	1	·	·	2	11	·	·	·	·	·	·	·
	FuncNet [68]	(180)	·	○	·	○	·	○	·	·	·	○	○	○	○	○	·	·	·	1	·	·	·	·	·	·	·	·	1	·	·	·	·	·	·	○
2020	DeepBinDiff [27]	2114	·	·	·	·	·																											○	○	○
	ImOpt [69]	18	·	·	○	·	·																											·	·	·
	ACCESS20 [52]	12,000	·	○	○	·	·																											·	·	·
	Patchecko [28]	2,108	2	○	○	○	○	·	·	·	·	○	○	○	○	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○
	BinKit ★	243,128	·	○	○	○	○	○	○	○	○	○	○	○	○	○	·	1	1	1	1	1	·	1	1	1	1	·	9	○	○	○	○	○	○	○

No same benchmark

*Revisiting Binary Coc

BCSA Dataset in Previous Literature

		Source*		Architecture*								Optimization*					Compiler†*													Extra				Info.		
Year	Tool [Paper]	Binaries	Firmware	x86	x64	arm	aarch64	mips	mips64	mipseb	mips64eb	O0	O1	O2	O3	Os	GCC 3	GCC 4	GCC 5	GCC 6	GCC 7	GCC 8	Clang 3	Clang 4	Clang 5	Clang 6	Clang 7	etc.	Total #	Noinline	PIE	LTO	Obfus.	Code	Dataset	IDA
2014	TEDEM [10]	14	·	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○
	Tracy [56]	(115)	·	·	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○	·	○
	CoP [7]	(214)	·	·	·	·	·	·	·	·	·	○	○	○	○	○	·	1	·	·	·	·	·	·	·	·	·	1	2	·	·	·	○	·	·	○
	LoPD [8]	48	·	○	·	·	·	·	·	·	·	○	○	○	○	○	·	1	·	·	·	·	·	·	·	·	·	1	2	·	·	○	○	·	·	·
	BLEX [30]	1,140	·	·	○	·	·	·	·	·	·	○	○	○	○	·	·	1	·	·	·	·	1	·	·	·	·	1	3	·	·	·	·	·	·	○
	BinClone [57]	90	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○	·	○
2015	Multi-k-MH [22]	60	6	○	·	○	·	○	·	·	·	○	○	○	○	·	·	2	·	·	·	·	1	·	·	·	·	·	3	·	·	·	·	·	·	○
2016	discovRE [11]	593	3	○	·	○	·	○	·	·	·	○	○	○	○	○	·	1	·	·	·	·	1	·	·	·	·	2	4	○	·	·	·	○	◐	○
	Genius [23]	(7,848)	8,128	○	·	○	·	○	·	·	·	○	○	○	○	·	·	2	·	·	·	·	1	·	·	·	·	·	3	·	·	·	·	·	·	○
	Esh [58]	(833)	·	·	○	·	·	·	·	·	·	·	·	·	·	·	·	3	·	·	·	·	2	·	·	·	·	2	7	·	·	·	·	○	○	○
	BinGo [15]	(5,143)	·	○	○	○	·	·	·	·	·	○	○	○	○	·	·	3	·	·	·	·	1	·	·	·	·	1	5	·	·	·	·	·	·	○
	MockingBird [33]	80	·	○	·	○	·	○	·	·	·	○	·	○	○	·	·	1	·	·	·	·	1	·	·	·	·	·	2	·	·	·	·	·	·	○
	Kam1n0 [32]	96	·	○	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○	·	○
	BinDNN [59]	2,064	·	○	○	○	·	·	·	·	·	○	○	○	○	·	·	1	·	·	·	·	·	·	·	·	·	1	2	·	·	·	·	·	·	○
2017	BinSign [60]	(31)	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	2	2	·	·	·	○	·	·	○
	Xmatch [16]	72	1	○	·	·	·	○	·	·	·	·	·	·	·	·	·	2	·	·	·	·	1	·	·	·	·	·	3	○	·	·	·	·	·	○
	Gemini [12]	18,269	8,128	○	·	○	·	○	·	·	·	○	○	○	○	·	·	·	1	·	·	·	·	·	·	·	·	·	1	·	·	·	·	·	·	○
	GitZ [61]	44	·	·	○	·	○	·	·	·	·	○	○	○	○	○	·	3	·	·	·	·	2	1	·	·	·	·	6	·	·	·	·	·	·	·
	BinSim [62]	1062	·	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○	·	·	○
	BinSequence [34]	(1,718)	·	·	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○
	IMF-sim [31]	1,140	·	·	○	·	·	·	·	·	·	○	·	○	○	·	·	1	·	·	·	·	1	·	·	·	·	1	3	·	·	·	○	·	·	·
	CACompare [35]	72	·	○	·	○	·	○	·	·	·	○	·	○	○	·	·	1	·	·	·	·	1	·	·	·	·	·	2	·	·	·	·	·	·	○
	ASE17 [63]	55	·	○	○	·	·	·	·	·	·	○	·	○	○	·	·	1	·	·	·	·	1	·	·	·	·	·	2	·	·	·	○	·	·	·
2018	BinArm [17]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	·	·	·	○
	SANER18 [64]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	·	·	·	○
	BinGo-E [18]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	·	·	·	○
	WSB [65]	(173)	·	·	·	·	·	·	·	·	·	·	○	○	○	·	·	1	·	·	·	·	1	·	·	·	·	·	2	·	·	·	○	·	·	·
	BinMatch [66]	80	·	○	·	·	·	·	·	·	·	○	·	○	○	·	·	1	·	·	·	·	1	·	·	·	·	·	2	·	·	·	○	·	·	○
	MASES18 [25]	47	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○	·	·	·
	Zeek [67]	(20,680)	·	○	·	○	○	·	·	·	·	○	○	○	○	○	·	3	·	·	·	·	4	1	·	·	·	2	10	·	·	·	·	·	·	·
	FirmUp [14]	·	2,000	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○
	αDiff [19]	(69,989)	2	○	○	○	·	·	·	·	·	○	○	○	○	·	·	2	1	·	·	·	2	·	·	·	·	·	5	·	·	·	·	◐	◐	○
	VulSeeker [13]	(10,512)	4,643	○	○	○	○	○	○	·	·	○	○	○	○	·	·	1	1	·	·	·	·	·	·	·	·	·	2	·	·	·	·	○	·	○
2019	InnerEye [24]	(844)	·	·	○	○	·	·	·	·	·	·	○	○	○	·	·	·	·	·	·	·	·	·	·	1	·	·	1	·	·	·	·	◐	◐	·
	Asm2Vec [20]	68	·	·	○	·	·	·	·	·	·	○	○	○	○	·	·	1	1	·	·	·	2	·	·	·	·	·	4	·	·	·	○	○	·	○
	SAFE [21]	(5001)	·	·	○	○	·	·	·	·	·	○	○	○	○	·	1	3	1	1	1	·	2	1	1	1	·	·	12	·	·	·	·	○	◐	○
	BAR19i [26]	(804)	·	·	○	○	·	·	·	·	·	·	○	○	○	·	·	·	·	·	·	·	·	·	·	1	·	·	1	·	·	·	·	○	·	·
	BAR19ii [29]	(11244)	·	○	○	○	·	·	·	·	·	○	○	○	○	·	1	3	1	·	·	·	2	1	1	·	·	2	11	·	·	·	·	·	·	·
	FuncNet [68]	(180)	·	○	·	○	·	○	·	·	·	○	○	○	○	○	·	·	·	1	·	·	·	·	·	·	·	·	1	·	·	·	·	·	·	○
2020	DeepBinDiff [27]	2114	·	·	·	·	·	·	·	·	·	○	○	○	○	·	·	1	·	·	·	·	·	·	·	·	·	·	1	·	·	·	·	○	○	○
	ImOpt [69]	18	·	·	○	·	·	·	·	·	·	○	·	○	○	·	·	·	1	·	·	·	·	·	·	·	·	·	1	·	·	·	○	·	·	·
	ACCESS20 [52]	12,000	·	○	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
	Patchecko [28]	2,108	2	○	○	○	○	·	·	·	·	○	○	○	○	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○
	BinKit ★	243,128	·	○	○	○	○	○	○	○	○	○	○	○	○	○	·	1	1	1	1	1	·	1	1	1	1	·	9	○	○	○	○	○	○	○

Only 2 released full dataset

*Revisiting Binary Co

BCSA Dataset in Previous Literature

		Source*		Architecture*								Optimization*					Compiler†*													Extra				Info.		
Year	Tool [Paper]	Binaries	Firmware	x86	x64	arm	aarch64	mips	mips64	mipseb	mips64eb	O0	O1	O2	O3	Os	GCC 3	GCC 4	GCC 5	GCC 6	GCC 7	GCC 8	Clang 3	Clang 4	Clang 5	Clang 6	Clang 7	etc.	Total #	Noinline	PIE	LTO	Obfus.	Code	Dataset	IDA
2014	TEDEM [10]	14	·	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○
	Tracy [56]	(115)	·	·	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○	·	○
	CoP [7]	(214)	·	·	·	·	·	·	·	·	·	○	○	○	○	○	·	1	·	·	·	·	·	·	·	·	·	1	2	·	·	·	○	·	·	○
	LoPD [8]	48	·	○	·	·	·	·	·	·	·	○	○	○	○	○	·	1	·	·	·	·	·	·	·	·	·	1	2	·	·	○	○	·	·	·
	BLEX [30]	1,140	·	·	○	·	·	·	·	·	·	○	○	○	○	·	·	1	·	·	·	·	1	·	·	·	·	1	3	·	·	·	·	·	·	○
	BinClone [57]	90	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○	·	○
2015	Multi-k-MH [22]	60	6	○	·	○	·	○	·	·	·	○	○	○	○	·	·	2	·	·	·	·	1	·	·	·	·	·	3	·	·	·	·	·	·	○
2016	discovRE [11]	593	3	○	·	○	·	○	·	·	·	○	○	○	○	○	·	1	·	·	·	·	1	·	·	·	·	2	4	○	·	·	·	○	◐	○
	Genius [23]	(7,848)	8,128	○	·	○	·	○	·	·	·	○	○	○	○	·	·	2	·	·	·	·	1	·	·	·	·	·	3	·	·	·	·	·	·	○
	Esh [58]	(833)	·	·	○	·	·	·	·	·	·	·	·	·	·	·	·	3	·	·	·	·	2	·	·	·	·	2	7	·	·	·	·	○	○	○
	BinGo [15]	(5,143)	·	○	○	○	·	·	·	·	·	○	○	○	○	·	·	3	·	·	·	·	1	·	·	·	·	1	5	·	·	·	·	·	·	○
	MockingBird [33]	80	·	○	·	○	·	○	·	·	·	○	·	○	○	·	·	1	·	·	·	·	1	·	·	·	·	·	2	·	·	·	·	·	·	○
	Kam1n0 [32]	96	·	○	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○	·	○
	BinDNN [59]	2,064	·	○	○	○	·	·	·	·	·	○	○	○	○	·	·	1	·	·	·	·	·	·	·	·	·	1	2	·	·	·	·	·	·	○
2017	BinSign [60]	(31)	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	2	2	·	·	·	○	·	·	○
	Xmatch [16]	72	1	○	·	·	·	○	·	·	·	·	·	·	·	·	·	2	·	·	·	·	1	·	·	·	·	·	3	○	·	·	·	·	·	○
	Gemini [12]	18,269	8,128	○	·	○	·	○	·	·	·	○	○	○	○	·	·	·	1	·	·	·	·	·	·	·	·	·	1	·	·	·	·	·	·	○
	GitZ [61]	44	·	·	○	·	○	·	·	·	·	○	○	○	○	○	·	3	·	·	·	·	2	1	·	·	·	·	6	·	·	·	·	·	·	·
	BinSim [62]	1062	·	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○	·	·	○
	BinSequence [34]	(1,718)	·	·	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○
	IMF-sim [31]	1,140	·	·	○	·	·	·	·	·	·	○	·	○	○	·	·	1	·	·	·	·	1	·	·	·	·	1	3	·	·	·	○	·	·	·
	CACompare [35]	72	·	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	○
	ASE17 [63]	55	·	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	·
2018	BinArm [17]	·	2,6[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	○
	SANER18 [64]	7	·	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	○
	BinGo-E [18]	(5,145)	·	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	○
	WSB [65]	(173)	·	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	·
	BinMatch [66]	80	·	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	○
	MASES18 [25]	47	·	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	·
	Zeek [67]	(20,680)	·	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	·
	FirmUp [14]	·	2,0[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	○
	αDiff [19]	(69,989)	2[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	○
	VulSeeker [13]	(10,512)	4,6[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	○
2019	InnerEye [24]	(844)	·	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	·
	Asm2Vec [20]	68	·	·	○	·	·	·	·	·	·	○	○	○	○	·	·	1	1	·	·	·	2	·	·	·	·	·	4	·	·	·	○	○	·	○
	SAFE [21]	(5001)	·	·	○	○	·	·	·	·	·	○	○	○	○	·	1	3	1	1	1	·	2	1	1	1	·	·	12	·	·	·	·	○	◐	○
	BAR19i [26]	(804)	·	·	○	○	·	·	·	·	·	·	○	○	○	·	·	·	·	·	·	·	·	·	·	1	·	·	1	·	·	·	·	○	·	·
	BAR19ii [29]	(11244)	·	○	○	○	·	·	·	·	·	○	○	○	○	·	1	3	1	·	·	·	2	1	1	·	·	2	11	·	·	·	·	·	·	·
	FuncNet [68]	(180)	·	○	·	○	·	○	·	·	·	○	○	○	○	○	·	·	·	1	·	·	·	·	·	·	·	·	1	·	·	·	·	·	·	○
2020	DeepBinDiff [27]	2114	·	·	·	·	·	·	·	·	·	○	○	○	○	·	·	1	·	·	·	·	·	·	·	·	·	·	1	·	·	·	·	○	○	○
	ImOpt [69]	18	·	·	○	·	·	·	·	·	·	○	·	○	○	·	·	·	1	·	·	·	·	·	·	·	·	·	1	·	·	·	○	·	·	·
	ACCESS20 [52]	12,000	·	○	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
	Patchecko [28]	2,108	2[illegible]	○	○	○	○	·	·	·	·	○	○	○	○	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○
	BinKit ★	243,128	·	○	○	○	○	○	○	○	○	○	○	○	○	○	·	1	1	1	1	1	·	1	1	1	1	·	9	○	○	○	○	○	○	○

Insufficient benchmarks
(86% < 10,000 binaries)
(98% < 4 architectures)
➔ Hard to evaluate useful features

*Revisiting Binary Cod

BCSA Dataset in Previous Literature

Year	Tool [Paper]	Source* Binaries	Firmware	Architecture* x86	x64	arm	aarch64	mips	mips64	mipseb	mips64eb	Optimization* O0	O1	O2	O3	Os	Compiler†* GCC 3	GCC 4	GCC 5	GCC 6	GCC 7	GCC 8	Clang 3	Clang 4	Clang 5	Clang 6	Clang 7	etc.	Total #	Extra Noinline	PIE	LTO	Obfus.	Info. Code	Dataset	IDA
2014	TEDEM [10]	14	·	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○
	Tracy [56]	(115)	·	·	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○	·	○
	CoP [7]	(214)	·	·	·	·	·	·	·	·	·	○	○	○	○	○	·	1	·	·	·	·	·	·	·	·	·	1	2	·	·	·	○	·	·	○
	LoPD [8]	48	·	○	·	·	·	·	·	·	·	○	○	○	○	○	·	1	·	·	·	·	·	·	·	·	·	1	2	·	·	○	○	·	·	·
	BLEX [30]	1,140	·	·	○	·	·	·	·	·	·	○	○	○	○	·	·	1	·	·	·	·	1	·	·	·	·	1	3	·	·	·	·	·	·	○
	BinClone [57]	90	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○	·	○
2015	Multi-k-MH [22]	60	6	○	·	○	·	○	·	·	·	○	○	○	○	·	·	2	·	·	·	·	1	·	·	·	·	·	3	·	·	·	·	·	·	○
2016	discovRE [11]	593	3	○	·	○	·	○	·	·	·	○	○	○	○	○	·	1	·	·	·	·	1	·	·	·	·	2	4	○	·	·	·	○	◑	○
	Genius [23]	(7,848)	8,128	○	·	○	·	○	·	·	·	○	○	○	○	·	·	2	·	·	·	·	1	·	·	·	·	·	3	·	·	·	·	·	·	○
	Esh [58]	(833)	·	·	○	·	·	·	·	·	·	·	·	·	·	·	·	3	·	·	·	·	2	·	·	·	·	2	7	·	·	·	·	○	○	○
	BinGo [15]	(5,143)	·	○	○	○	·	·	·	·	·	○	○	○	○	·	·	3	·	·	·	·	1	·	·	·	·	1	5	·	·	·	·	·	·	○
	MockingBird [33]	80	·	○	·	○	·	○	·	·	·	○	·	○	○	·	·	1	·	·	·	·	1	·	·	·	·	·	2	·	·	·	·	·	·	○
	Kam1n0 [32]	96	·	○	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○	·	○
	BinDNN [59]	2,064	·	○	○	○	·	·	·	·	·	○	○	○	○	·	·	1	·	·	·	·	·	·	·	·	·	1	2	·	·	·	·	·	·	○
2017	BinSign [60]	(31)	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	2	2	·	·	·	○	·	·	○
	Xmatch [16]	72	1	○	·	·	·	○	·	·	·	·	·	·	·	·	·	2	·	·	·	·	1	·	·	·	·	·	3	○	·	·	·	·	·	○
	Gemini [12]	18,269	8,128	○	·	○	·	○	·	·	·	○	○	○	○	·	·	·	1	·	·	·	·	·	·	·	·	·	1	·	·	·	·	·	·	○
	GitZ [61]	44	·	·	○	·	○	·	·	·	·	○	○	○	○	○	·	3	·	·	·	·	2	1	·	·	·	·	6	·	·	·	·	·	·	·
	BinSim [62]	1062	·	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○	·	·	○
	BinSequence [34]	(1,718)	·	·	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○
	IMF-sim [31]	1,140	·	·	○	·	·	·	·	·	·	○	·	○	○	·	·	1	·	·	·	·	1	·	·	·	·	1	3	·	·	·	○	·	·	·
	CACompare [35]	72	·	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	·	·	○
	ASE17 [63]	55	·	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	·	·	·
2018	BinArm [17]	·	2,628	·	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	·	·	○
	SANER18 [64]	7	·	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	·	·	○
	BinGo-E [18]	(5,145)	·	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	·	·	○
	WSB [65]	(173)	·	·	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	·	·	·
	BinMatch [66]	80	·	○	·	·	·	·	·	·	·	○	·	○	○	·	·	1	·	·	·	·	1	·	·	·	·	·	2	·	·	·	○	·	·	○
	MASES18 [25]	47	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○	·	·	·
	Zeek [67]	(20,680)	·	○	·	○	○	·	·	·	·	○	○	○	○	○	·	3	·	·	·	·	4	1	·	·	·	2	10	·	·	·	·	·	·	·
	FirmUp [14]	·	2,000	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○
	αDiff [19]	(69,989)	2	○	○	○	·	·	·	·	·	○	○	○	○	·	·	2	1	·	·	·	2	·	·	·	·	·	5	·	·	·	·	◑	◑	○
	VulSeeker [13]	(10,512)	4,643	○	○	○	○	○	○	·	·	○	○	○	○	·	·	1	1	·	·	·	·	·	·	·	·	·	2	·	·	·	·	○	·	○
2019	InnerEye [24]	(844)	·	·	○	○	·	·	·	·	·	·	○	○	○	·	·	·	·	·	·	·	·	·	·	1	·	·	1	·	·	·	·	◑	◑	·
	Asm2Vec [20]	68	·	·	○	·	·	·	·	·	·	○	○	○	○	·	·	1	1	·	·	·	2	·	·	·	·	·	4	·	·	·	○	○	·	○
	SAFE [21]	(5001)	·	·	○	○	·	·	·	·	·	○	○	○	○	·	1	3	1	1	1	·	2	1	1	1	·	·	12	·	·	·	·	○	◑	○
	BAR19i [26]	(804)	·	·	○	○	·	·	·	·	·	·	○	○	○	·	·	·	·	·	·	·	·	·	·	1	·	·	1	·	·	·	·	○	·	·
	BAR19ii [29]	(11244)	·	○	○	○	·	·	·	·	·	○	○	○	○	·	1	3	1	·	·	·	2	1	1	·	·	2	11	·	·	·	·	·	·	·
	FuncNet [68]	(180)	·	○	·	○	·	○	·	·	·	○	○	○	○	○	·	·	·	1	·	·	·	·	·	·	·	·	1	·	·	·	·	·	·	○
2020	DeepBinDiff [27]	2114	·	·	·	·	·	·	·	·	·	○	○	○	○	·	·	1	·	·	·	·	·	·	·	·	·	·	1	·	·	·	·	○	○	○
	ImOpt [69]	18	·	·	○	·	·	·	·	·	·	○	·	○	○	·	·	·	1	·	·	·	·	·	·	·	·	·	1	·	·	·	○	·	·	·
	ACCESS20 [52]	12,000	·	○	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
	Patchecko [28]	2,108	2	○	○	○	○	·	·	·	·	○	○	○	○	○	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	○
	BinKit ★	243,128	·	○	○	○	○	○	○	○	○	○	○	○	○	○	·	1	1	1	1	1	·	1	1	1	1	·	9	○	○	○	○	○	○	○

A few focused on IoT vulnerability Analysis

*Revisiting Binary Coc

Problems of Existing Studies

- In IoT devices, vulnerabilities can exist in
  - Libraries or utility binaries — **What BCSA studies have focused on**
  - Custom binaries (mostly, CGI binaries) — **None of BCSA studies targeted**

- Existing studies focus on **only libraries or utility binaries**
  - Open-source packages (e.g., OpenSSL, bash, vsftpd, …)
  - Easy to generate training dataset

- **None** has analyzed **custom binaries** (e.g., CGI binaries)
  - No available dataset (or vulnerability details)
  - Not enough samples

Problems of Existing Studies

- **No** available open-source tools
  - Among 43 BCSA studies, 10 released their source code
  - Among these 10 tools,
    - **Only 2 supports x86, ARM, MIPS** (i.e., Gemini, **VulSeeker**)
  - Most IoT devices are based on **ARM/MIPS**

- Limitations of Gemini and VulSeeker
  - Do not have full source code
  - Based on complex machine learning ➔ Hard to interpret/understand the results
  - **How about performance?**

Motivating Example: CVE-2015-1791

- VulSeeker released partial results without full source code
  - Target firmware: Tomato Cisco M10v2 (router)
  - Target vulnerability: *ssl3_get_new_session_ticket* in *libssl.so*
  - Race condition causes double free (DoS)

- Approach
  - Compile vulnerable OpenSSL package (v1.0.1f) with 48 compiler options
  - Query each of the 48 functions in the target firmware
  - Average the similarity scores for all functions

- Result
  - VulSeeker found the vulnerability at **Rank 21** → **Enough?**

Our Approach

- Fundamental problems of existing BCSA studies
  - No available dataset ➔ Establish a baseline benchmark (BinKit)
  - Heavy use of machine learning ➔ Develop a simple & interpretable model (TikNib)
  - Heavy use of semantic features ➔ Investigate pre-semantic features

- Problems of BCSA-based IoT vulnerability analysis
  - No analysis on custom binaries ➔ Establish ground truth dataset (FirmKit)
  - No available tool & Not enough studies ➔ Empirically analyze firmware images

Building a Comprehensive Benchmark (BinKit)

- Compile GNU software packages
- Build ground truth by leveraging source file names and line numbers

Category	Previous Options	Our Options (Count)
Architecture	98% tested ≤ 4	x86, arm ,mips, mipseb for 32, 64 bits (4x2=8)
Compiler	95% tested ≤ 5	GCC: v4~v8 (5) Clang: v4~v7 (4)
Optimization	16% tested all opti-levels	O0, O1, O2, O3, Os (5)
Noinline	5% tested	Include (1)
PIE	0% tested	Include (1)
Link Time Optimization	2% tested	Include (1)
Obfuscation	26% tested	Obfuscator-LLVM (4)

Building a Comprehensive Benchmark (BinKit)

- ❖ Compile GNU software packages
- ❖ Build ground truth by leveraging source file names and line numbers

Category	Previous Options	Our Options (Count)
Architecture	98% tested ≤ 4	x86, arm ,mips, mipseb for 32, 64 bits (4x2=8)
Compiler	95% tested ≤ 5	GCC: v4~v8 (5) Clang: v4~v7 (4)
Optimization	16% tested all opti-levels	O0, O1, O2, O3, Os (5)
Noinline	5% tested	Include (1)
PIE	0% tested	Include (1)
Link Time O		1)
Obfus		/M (4)

243,128 binaries for 36,256,322 functions

Analyze Pre-Semantic Features

- Justify semantic features (84%) and machine learning (90% after 2019)
  - ➔ **Cannot understand** the results
- Simple pre-semantic features
  - ➔ **Can understand** the results

Numeric Level	Feature Category	Example
CFG-Level (41 Features)	Graphic	Basic Blocks, Edges, …
	Computing	Arithmetic, Logic, …
	Data Manipulating	Copy, Addressing, …
	Control Transferring	Jmp, Conditional Jmp, …
	Category Mixing	Arithmetic + Shifting, …
CG-Level (6 Features)	Counting Unique	Callers, Callees, Imported Callees
	Including Duplicates	Incoming Calls, Outgoing Calls, Imported Calls

Design an Interpretable Model (TikNib)

- An intuitive model to easily understand the results

- Relative difference of feature *f* of function *A* and *B*

$$rdiff(A_f - Bf) = \frac{|A_f - B_f|}{|max(A_f, B_f)|}$$

- Similarity score of function *A* and *B*
  - Average of the relative differences of all features from *f1* to *fN*

$$score(A,B) = \frac{rdiff(A_{f1}, Bf_1) + \cdots + rdiff(A_{fN}, BfN)}{N}$$

  - Any other scoring metric can be integrated (e.g., Jaccard index)

Experiment Methodology

- There exist over 36M functions
  - ➔ We need a fast approach to obtain the tendency

- Utilize TP/TN pairs for each function **λ** (same as Gemini, VulSeeker)

$$\overbrace{(\lambda^{\text{Target}}, \lambda^{\text{True Positive}}}^{\text{Different Compiler Option}}) \qquad (\lambda^{\text{Target}}, \lambda^{\text{True Negative}})$$

Same Compiler Option

- Greedily select features with ROC AUC
- 10-fold cross validation for each test

: Exist in all 10 tests

Feature	Rand.	Opt Level O0 vs. O3	Opt Level O2 vs. O3	Opt Level Rand.	Compiler GCC v4 vs. GCC v8	Compiler Clang v4 vs. Clang v7	Compiler GCC vs. Clang	Compiler Rand.	Arch x86 vs. ARM	Arch x86 vs. MIPS	Arch ARM vs. MIPS	Arch 32 vs. 64	Arch LE vs. BE	Arch Rand.	vs. SizeOpt† O0 vs. Os	vs. SizeOpt† O1 vs. Os	vs. SizeOpt† O3 vs. Os	vs. Extra† PIE	vs. Extra† NoInline	vs. Extra† LTO	vs. Obfus.† BCF	vs. Obfus.† FLA	vs. Obfus.† SUB	vs. Obfus.† All	Bad‡ Norm.	Bad‡ Norm. vs. Obfus.†
CFG Avg. # of edges	0.33	0.26	0.42	0.34	0.44	0.46	0.37	0.41	0.43	0.37	0.37	0.43	0.47	0.41	0.34	0.42	0.36	0.45	0.44	0.40	0.26	0.34	0.47	0.22	0.32	**0.19**
CFG # of backedges	0.39	0.33	0.44	0.39	0.46	0.45	0.41	0.43	0.47	**0.45**	**0.45**	0.46	0.48	0.46	0.39	0.42	0.38	0.50	0.40	0.46	0.23	0.08	**0.47**	0.05	0.32	**0.03**
CFG # of edges	0.47	0.37	0.63	0.48	0.66	0.69	0.52	0.60	0.65	0.57	0.57	0.65	0.72	0.61	0.46	0.59	0.52	**0.71**	0.61	0.64	0.25	0.23	**0.72**	0.10	**0.42**	0.06
CFG # of loops	**0.40**	**0.34**	0.44	**0.40**	**0.46**	0.46	**0.41**	**0.44**	**0.47**	**0.45**	**0.45**	**0.46**	0.47	**0.46**	**0.40**	0.42	**0.39**	**0.50**	0.40	0.46	0.23	0.13	**0.47**	0.10	**0.33**	0.08
CFG # of basic blocks	0.41	0.36	0.59	0.46	0.62	0.65	0.48	0.56	0.44	0.55	**0.39**	0.62	0.69	0.52	0.43	0.56	0.50	0.67	0.57	0.60	0.26	0.23	0.67	0.10	**0.26**	0.00
CG # of callees	0.50	**0.43**	0.59	**0.52**	**0.62**	**0.63**	**0.52**	0.58	**0.63**	**0.54**	0.55	0.57	**0.64**	0.57	0.50	**0.59**	**0.53**	0.60	0.57	**0.57**	**0.60**	**0.59**	**0.64**	**0.56**	**0.46**	**0.47**
CG # of callers	0.45	0.40	0.54	**0.48**	0.59	**0.58**	0.49	0.54	0.53	0.41	0.45	0.54	**0.60**	0.50	**0.50**	0.57	0.50	0.56	0.54	0.52	**0.54**	**0.54**	**0.58**	**0.52**	**0.37**	**0.41**
CG # of imported callees	0.44	0.39	0.54	0.47	0.58	**0.56**	0.48	0.52	0.59	0.44	0.45	0.55	0.55	0.50	0.46	0.53	0.48	**0.55**	0.50	0.52	**0.53**	**0.53**	**0.56**	**0.50**	**0.36**	**0.43**
CG # of imported calls	0.45	0.38	0.56	0.48	0.60	0.58	0.48	0.54	0.61	0.45	**0.47**	0.57	0.57	0.52	0.46	0.54	0.48	0.57	0.52	0.54	0.49	0.54	**0.59**	0.46	0.37	0.40
CG # of incoming calls	**0.46**	**0.41**	**0.56**	**0.50**	**0.61**	**0.60**	**0.50**	**0.56**	**0.55**	**0.42**	**0.46**	**0.56**	**0.62**	**0.52**	**0.52**	**0.58**	**0.50**	0.58	**0.57**	**0.55**	0.50	**0.56**	**0.60**	0.47	**0.37**	**0.38**
CG # of outgoing calls	0.52	0.44	0.62	0.54	0.66	0.66	0.54	0.60	0.67	**0.57**	0.58	0.61	**0.68**	0.60	**0.52**	0.61	0.55	0.64	0.60	0.61	0.53	**0.62**	**0.67**	0.50	**0.48**	**0.44**
Inst Avg. # of arith+shift	0.17	0.16	0.51	0.30	0.50	0.50	0.27	0.39	0.21	**0.08**	0.10	0.29	0.52	0.21	0.19	0.43	**0.41**	0.49	0.50	0.43	0.28	0.22	0.46	0.17	0.07	0.12
Inst Avg. # of ctransfer	0.17	**0.15**	0.28	0.20	0.28	0.30	0.20	0.25	0.26	0.19	0.21	0.25	0.32	0.22	**0.19**	0.24	0.22	0.30	0.27	0.27	0.19	0.18	0.31	0.12	0.12	0.07
Inst Avg. # of dtransfer+misc	0.17	0.08	0.44	0.22	0.42	0.46	0.27	**0.36**	0.30	0.15	0.17	0.31	0.49	0.25	0.09	0.36	0.35	0.45	0.44	0.36	0.28	0.26	0.45	0.17	0.12	0.08
Inst Avg. # of dtransfer	0.19	0.10	0.45	0.23	0.43	**0.48**	0.28	0.37	0.30	**0.20**	0.22	0.32	**0.53**	0.28	0.11	0.38	**0.36**	0.46	0.46	0.38	0.28	0.27	**0.47**	0.18	0.10	0.08
Inst Avg. # of instrs.	0.17	0.11	0.38	0.21	0.37	0.41	0.25	0.33	0.30	0.15	0.15	0.28	0.45	0.24	0.12	0.32	0.31	0.40	0.38	**0.33**	**0.26**	**0.25**	0.37	**0.17**	**0.14**	**0.10**
Inst Avg. # of logic	0.24	0.23	0.51	0.34	0.45	0.56	0.27	0.39	0.22	0.21	0.25	0.40	0.54	0.31	0.25	0.40	0.42	0.56	0.48	0.54	0.23	0.22	0.40	0.12	**0.24**	0.05
Inst # of arith+shift	0.24	0.26	0.59	0.40	0.59	0.60	0.38	0.49	0.28	0.11	0.12	0.37	0.61	0.27	0.28	0.52	0.48	0.58	0.56	0.53	0.26	**0.41**	0.55	0.14	0.13	0.02
Inst # of ctransfer	**0.42**	**0.35**	0.56	**0.43**	0.57	0.62	0.43	0.51	**0.57**	**0.51**	**0.54**	**0.56**	0.64	**0.54**	0.38	0.51	0.46	0.62	0.54	**0.57**	0.25	0.23	**0.64**	0.10	**0.32**	0.03
Inst # of dtransfer+misc	0.27	0.15	0.58	0.30	0.58	0.60	0.43	0.51	0.46	0.26	0.29	0.46	0.63	0.38	0.11	0.52	0.47	**0.60**	0.57	0.51	0.28	0.22	0.59	0.09	0.25	0.01
Inst # of arith	0.24	0.26	0.59	0.39	0.59	**0.60**	0.38	0.49	0.27	0.12	0.13	0.38	0.61	0.27	0.28	0.52	0.48	0.57	0.57	0.53	0.25	0.41	0.55	0.14	0.12	0.01
Inst # of bit-manipulating	0.09	0.12	0.34	0.21	0.31	0.23	0.18	0.24	0.07	0.04	0.06	0.22	0.20	0.10	0.14	0.31	0.28	**0.36**	0.32	0.34	0.17	0.16	0.20	0.05	0.05	0.01
Inst # of compare	0.47	0.42	0.62	**0.53**	0.67	0.68	0.55	0.62	0.39	**0.58**	0.32	0.61	0.72	0.55	**0.54**	0.61	0.52	**0.71**	0.60	0.64	0.23	0.25	0.68	0.09	0.40	0.06
Inst # of cond ctransfer	0.53	0.42	0.62	0.54	0.67	**0.70**	**0.58**	**0.63**	0.67	**0.65**	**0.65**	**0.68**	0.72	**0.66**	**0.55**	0.62	0.51	**0.72**	0.60	0.65	0.23	0.23	0.71	0.10	0.42	0.06
Inst # of dtransfer	0.28	0.16	0.58	0.31	0.58	0.60	0.43	0.51	0.45	0.29	0.35	0.46	0.63	**0.41**	0.13	0.53	**0.48**	0.60	0.57	0.52	**0.31**	0.22	0.59	0.09	**0.23**	0.03
Inst # of float instrs.	0.09	0.09	**0.27**	0.16	0.16	0.28	0.12	0.17	0.09	0.10	0.08	0.17	0.32	0.11	0.09	0.22	0.23	**0.27**	0.23	0.17	0.24	0.12	**0.30**	0.08	0.00	0.00
Inst # instrs.	0.31	0.21	0.57	0.34	**0.58**	**0.60**	**0.45**	**0.52**	**0.52**	0.29	0.30	0.48	**0.62**	0.42	0.18	**0.52**	0.47	**0.60**	0.55	0.52	0.26	0.22	0.56	0.08	0.30	0.02
Inst # of misc	0.10	0.04	0.47	0.17	0.40	0.46	0.20	0.32	0.06	0.11	0.02	0.36	0.67	0.19	0.04	0.29	0.27	0.47	0.44	0.37	0.16	0.19	**0.49**	0.09	0.00	0.00
Inst # of shift	0.22	0.23	0.42	0.30	0.42	0.39	0.27	0.34	0.22	0.20	0.19	0.31	0.54	0.26	0.24	0.35	0.34	**0.48**	0.37	0.41	**0.40**	0.34	0.42	**0.31**	0.19	**0.23**
Avg. TP-TN Gap	0.31	0.26	0.49	0.35	0.49	0.51	0.36	0.43	0.39	0.33	0.32	0.43	0.54	0.38	0.30	0.44	0.40	0.52	0.47	0.47	0.28	0.26	0.50	0.17	0.24	0.11
Avg. of Grey	0.43	0.34	0.42	0.48	0.57	0.59	0.49	0.50	0.55	0.44	0.49	0.57	0.61	0.52	0.46	0.56	0.45	0.55	0.57	0.51	0.44	0.47	0.55	0.41	0.33	0.33
ROC AUC	0.94	0.90	0.97	0.95	0.99	1.00	0.96	0.98	0.99	0.98	0.98	0.99	1.00	0.98	0.96	0.98	0.95	1.00	0.97	0.98	0.98	0.98	1.00	0.95	0.91	0.91
Std. of ROC AUC	0.01	0.01	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.01	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.02	0.01

Examples of Findings

Architecture has a small impact	ROC AUC
x86 vs ARM	0.99
x86 vs MIPS	0.98
ARM vs MIPS	0.98
32-bit vs 64-bit (Bits)	0.99
Little vs Big (Endian)	1.00

Optimization is largely influential	
O0 vs O3	0.90
O2 vs O3	0.97

Compiler version has almost no effect	
GCCv4 vs GCCv8	0.99
Clangv4 vs Clangv7	1.00

GCC and Clang have diverse characteristics	ROC AUC
GCC vs Clang	0.96

Extra Options are less effective	
vs PIE	1.00
vs Noinline	0.97
vs LTO	0.98

O-LLVM is insufficient for evaluation	
vs Bogus Control Flow	0.98
vs Control Flow Flattening	0.98
vs Instruction Substitution	1.00
vs All Three Options	0.95

Pre-semantic Features Are Effective!

- VulSeeker (ASE'18)
  - State of the art using numeric features
  - Use both pre-semantic and semantic features with deep neural network

- vs VulSeeker

ROC AUC

Dataset	Packages	Arch	Compilers	VulSeeker	Ours
ASE1	2	3	1	0.99	**0.9661**
ASE2	**5**	3	1	-	**0.9610**
ASE3	5	**6**	2	0.8849	**0.9616**
ASE4	5	**8**	**9**	-	**0.9450**

Larger Dataset

Case Study: Heartbleed

- Utilize TikNib to analyze Heartbleed (CVE-2014-0160)
  - Genius, Gemini, Multi-kMH, DiscovRE, SAFE, …
- Target: *tls1_process_heartbeat, dtls1_process_heartbeat*
  - OpenSSL v1.0.1f (vulnerable), v1.0.1u (patched)
  - Query *tls1_process_heartbeat*
- Average the similarity score rank in each option

Source option to Target option	All to All	ARM to ARM	ARM to MIPS	ARM to x86	MIPS to MIPS	MIPS to ARM	MIPS to x86	x86 to x86	x86 to ARM	x86 to MIPS	O2 to O3	O3 to O2	GCC to Clang	GCC v4 to GCC v8	GCC v8 to GCC v4	Clang v4 to Clang v7	Clang v7 to Clang v4
# of Option Pairs	552	56	64	64	56	64	64	56	64	64	144	144	144	36	36	36	36
Rank (tls, vuln)*	1.19	1.14	1.66	1	1	1.62	1	1	1.25	1	1.18	1.19	1	1.44	1.06	1	1
Precision@1 (tls, vuln)*	0.89	0.86	0.66	1	1	0.75	1	1	0.75	1	0.9	0.89	1	0.78	0.94	1	1
Rank (dtls, vuln)†	4.54	9.82	11.81	3.06	2	4.72	2	2.07	1.75	3.62	4.5	4.38	2.72	3.11	5.06	3.61	3.33
Rank (tls, patched)‡	29.16	12.12	57.69	3.56	3.82	51.62	43.94	4.29	6.38	70.59	27.5	28.96	27.68	32.89	40.89	20.22	22.67
Rank (dtls, patched)‡	76.47	46.95	145.75	7.25	8.21	128	128.94	9.57	11.94	181.03	73.04	75.41	87.31	66.28	87.33	68.44	78

*tls: tls1_process_heartbeat *dtls: dtls1_process_heartbeat

Case Study: Heartbleed

- Utilize TikNib to analyze Heartbleed (CVE-2014-0160)
  - Genius, Gemini, Multi-kMH, DiscovRE, SAFE, …
- Target: *tls1_process_heartbeat, dtls1_process_heartbeat*
  - OpenSSL v1.0.1f (vulnerable), v1.0.1u (patched)
  - Query *tls1_process_heartbeat*
- Average the similarity score rank in each option

Source option to Target option	All to All	ARM to ARM	ARM to MIPS	ARM to x86	MIPS to MIPS	MIPS to ARM	MIPS to x86	x86 to x86	x86 to ARM	x86 to MIPS	O2 to O3	O3 to O2	GCC to Clang	GCC v4 to GCC v8	GCC v8 to GCC v4	Clang v4 to Clang v7	Clang v7 to Clang v4
# of Option Pairs	552	56	64	64	56	64	64	56	64	64	144	144	144	36	36	36	36
Rank (tls, vuln)*	1.19	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1.06	1	1
Precision@1 (tls, vuln)*	0.89	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.94	1	1
Rank (dtls, vuln)†	4.54	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	5.06	3.61	3.33
Rank (tls, patched)‡	29.16	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	40.89	20.22	22.67
Rank (dtls, patched)‡	76.47	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	87.33	68.44	78

Pre-semantic features with a simple/interpretable model is effective!

*tls: tls1_process_heartbeat *dtls: dtls1_process_heartbeat

Our Approach

- Fundamental problems of existing BCSA studies
  - No available dataset ➔ Establish a baseline benchmark (BinKit)
  - Heavy use of machine learning ➔ Develop a simple & interpretable model (TikNib)
  - Heavy use of semantic features ➔ Investigate pre-semantic features

  ➔ **Proper feature engineering is important**

  ➔ **Simple model with presemantic features can show promising performance**

- Problems of BCSA-based IoT vulnerability analysis
  - No analysis on custom binaries ➔ Establish ground truth dataset (FirmKit)
  - No available tool & Not enough studies ➔ Empirically analyze firmware images

Establishing Ground Truth Dataset

Wireless Routers, IP Cameras

- Simple custom binaries
- Target dataset
  - 1,124 firmware images
  - 52,086,995 functions
  - **267 vulnerable functions**
    - 98 command injection
    - 162 information leak
    - 7 buffer overflow

    ➔ 19 unique vulnerabilities

Smartphone Cellular Baseband

- Complex custom binaries
- Target dataset
  - 18 firmware images
  - 1,405,959 functions
  - **56 vulnerable functions**
    - 18 uninitialized pointer dereference
    - 38 buffer overflow

    ➔ 4 unique vulnerabilities

➔ **Manually marked vulnerable function addresses**

Analyzing Linux-based IoT Devices

- ❖ Randomly select one sample for each unique vulnerability
- ❖ Query it for each firmware image (1,124 images, 52M funcs)

	Original TikNib	
Top-k	# of Total Vulns	Percent
1	141 / 267	52.81%
5	167 / 267	62.55%
10	182 / 267	68.16%
50	196 / 267	73.41%
100	**196 / 267**	**73.41%**

Analyzing Linux-based IoT Devices

- Randomly select one sample for each unique vulnerability
- Query it for each firmware image (1,124 images, 52M funcs)

	Original TikNib	
Top-k	# of Total Vulns	Percent
1	141 / 267	52.81%
5	167 / 267	62.55%
10	182 / 267	68.16%
50	196 / 267	73.41%

How to increase the performance?

Failure Case Study - CVE-2015-2051

- Architecture specific issues
  - ARM -> ARM: detected at Rank 1.75 on average
  - ARM -> MIPS: detected at Rank over 1000

- Arm produces a wrapper function for a library function call (.PLT)
  - ➔ # of callees, # of imported callees, cfg_size, …

```
v9 = 0;
memset(s, 0, sizeof(s));
v6 = getenv("HTTP_AUTHORIZATION");
haystack = getenv("HTTP_SOAPACTION");
s1 = getenv("REQUEST_METHOD");
```

ARM (Wrapper Function Call)

```
memset(v26, 0, sizeof(v26));
v4 = getenv("HTTP_AUTHORIZATION");
v5 = getenv("HTTP_SOAPACTION");
v6 = getenv("REQUEST_METHOD");
```

MIPS (External Function Call)

Failure Case Study - CVE-2017-5521

```c
sub_155D4(a1, "answer1");
sub_155D4(a1, "answer2");
v4 = (const char *)acosNvramConfig_get(&unk_87F8E);
v5 = (const char *)acosNvramConfig_get(&unk_87F9F);
if ( !strcasecmp(v26, v4) && !strcasecmp(v25, v5) )
{
  if ( (int)time(0) > 0x47302D4D )
  {
    time(&timer);
    localtime_r(&timer, &tp);
    v7 = (const char *)sub_6A460("language");
    if ( !strcmp("Japanese", v7) )
    {
      tm_year = tp.tm_year;
      v13 = sub_15FE8("year");
      v14 = tm_year + 1900;
      if ( tp.tm_mon )
      {
        switch ( tp.tm_mon )
        {
          case 1:
            v15 = "month_feb";
            break;
          case 2:
            v15 = "month_mar";
            break;
          case 3:
            v15 = "month_apr";
            break;
          case 4:
            v15 = "month_may";
            break;
          case 5:
            v15 = "month_jun";
```

Different version has an additional check routine

```c
websGetVar(a1, "answer1", v10);
websGetVar(a1, "answer2", v11);
v4 = (const char *)acosNvramConfig_get("password_answer1");
v5 = (const char *)acosNvramConfig_get("password_answer2");
if ( !strcasecmp(v10, v4) && !strcasecmp(v11, v5) )
{
  if ( time(0) > 0x47302D4D )
  {
    time(&v9);
    v7 = localtime(&v9);
    v8 = asctime(v7);
    strcpy(v12, v8);
    acosNvramConfig_set("timestamp_of_last_recovery", v12);
  }
  else
  {
    acosNvramConfig_set("timestamp_of_last_recovery", "");
  }
  acosNvramConfig_save();
  sendPage2Client("MNU_accessPassword_recovered.htm", a2);
}
else
{
  sendPage2Client("MNU_accessUnauthorized_checkAnswerAgain.htm", a2);
}
return 0;
}
```

No such routine exists

Failure Case Study - CVE-2017-5521

```
sub_155D4(a1, "answer1");
sub_155D4(a1, "answer2");
v4 = (const char *)acosNvramConfig_get(&unk_87F8E);
v5 = (const char *)acosNvramConfig_get(&unk_87F9F);
if ( !strcasecmp(v26, v4) && !strcasecmp(v25, v5) )
{
  if ( (int)time(0) > 0x47302D4D )
  {
    time(&timer);
    localtime_r(&timer, &tp);
    v7 = (const char *)sub_6A460("language");
    if ( !strcmp("Japanese", v7) )
    {
      tm_year = tp.tm_year;
      v13 = sub_15FE8("year");
      v14 = tm_year + 1900;
      if ( tp.tm_mon )
      {
        switch ( tp.tm_mon )
        {
          case 1:
            v15 = "month_feb";
            break;
          case 2:
            v15 = "month_mar";
            break;
          case
            v15
            bre
          case
            v15
            bre
          case
            v15
```

```
websGetVar(a1, "answer1", v10);
websGetVar(a1, "answer2", v11);
v4 = (const char *)acosNvramConfig_get("password_answer1");
v5 = (const char *)acosNvramConfig_get("password_answer2");
if ( !strcasecmp(v10, v4) && !strcasecmp(v11, v5) )
{
  if ( time(0) > 0x47302D4D )
  {
    time(&v9);
    v7 = localtime(&v9);
    v8 = asctime(v7);
    strcpy(v12, v8);
    acosNvramConfig_set("timestamp_of_last_recovery", v12);
  }
  else
  {
    acosNvramConfig_set("timestamp_of_last_recovery", "");
  }
  acosNvramConfig_save();
  sendPage2Client("MNU_accessPassword_recovered.htm", a2);
}
else
{
  sendPage2Client("MNU_accessUnauthorized_checkAnswerAgain.htm", a2);
```

No such routine exists

Need features robust against different architectures and versions

Developing Heuristic Features

- Leverage heuristic knowledge of binary analysts

- IoT binaries often contain function names
  - Use caller and callee names (i.e., internal and library function names)

- Data strings often contain useful information
  - CGI binaries parse URLs with hard-coded strings
    - "HTTP", "POST", "answer1", "password", …
  - Use words in a string

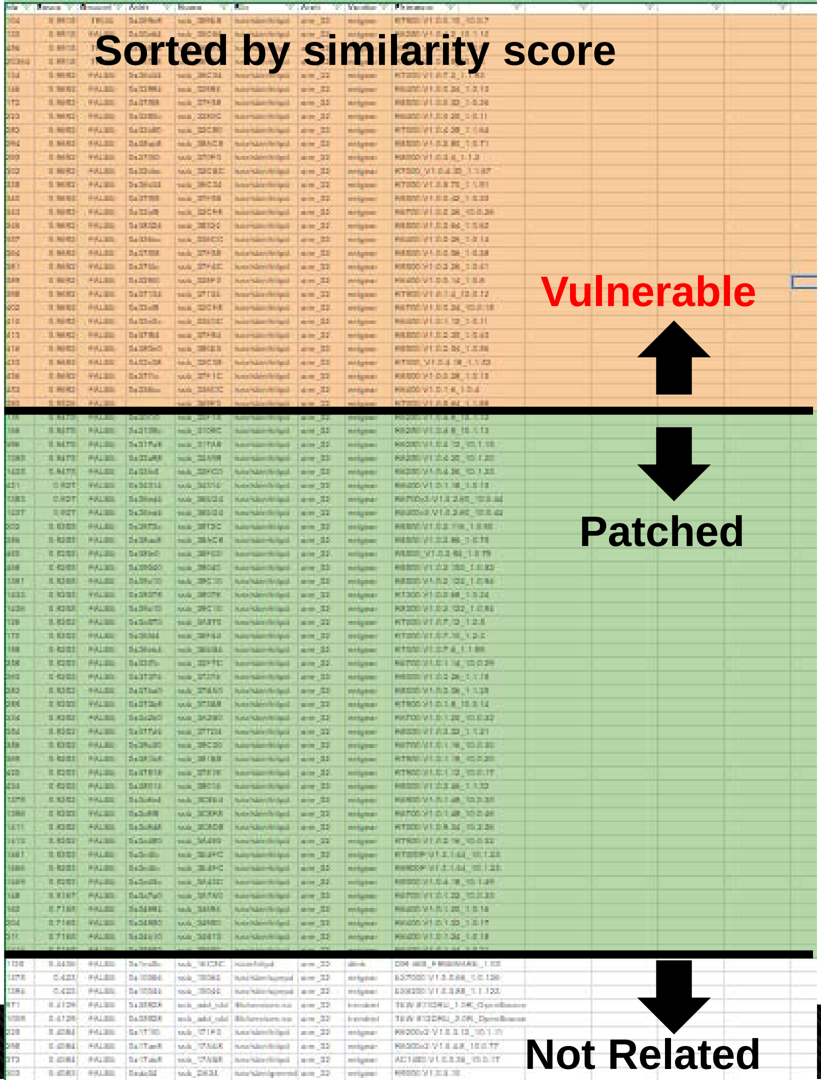
- Compare each word with Jaccard index
  - The score is merged with TikNib

$$Jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Final Results of Linux-based IoT Devices

- Randomly select one sample for each unique vulnerability
- Query it for each firmware image (1,124 images, 52M funcs)

	Original TikNib		TikNib (+Heuristic Features)	
Top-k	# of Total Vulns	Percent	# of Total Vulns	Percent
1	141 / 267	52.81%	**263 / 267**	**98.50%**
5	167 / 267	62.55%	263 / 267	98.50%
10	182 / 267	68.16%	266 / 267	99.63%
50	196 / 267	73.41%	266 / 267	99.63%
100	196 / 267	73.41%	267 / 267	100%



				Vendor							Arch‡			
Vulnerability†	Range	# of Funcs	Vuln†	Netgear	D-Link	TRENDnet	Belkin	Asus	ZyXEL	Linksys	arm	mips	mipseb	Binary
CVE-2016-6277 (104)	0.95–1.00	29 (3)	V	✓	·	·	·	·	·	·	✓	·	·	/usr/sbin/httpd
	0.5–0.95	40 (-)	P	✓	·	·	·	·	·	·	✓	·	·	/usr/sbin/httpd
CVE-2015-2051 (619)	0.81–1.00	5 (4)	V	·	✓	·	·	·	·	·	✓	·	·	/htdocs/cgibin
	0.68–0.73	25 (-)	P	·	✓	·	·	·	·	·	✓	·	·	/htdocs/cgibin
	0.58–0.75	6 (5)	V	·	✓	✓	·	·	·	·	·	✓	·	/htdocs/cgibin
	0.53–0.59	3 (-)	P	·	✓	·	·	·	·	·	·	✓	·	/htdocs/cgibin
	0.68	1 (-)	P	·	✓	·	·	·	·	·	·	·	✓	/htdocs/cgibin
	0.58–0.69	15 (14)	V	·	✓	·	·	·	·	·	·	·	✓	/htdocs/cgibin
	0.53	9 (-)	P	·	✓	·	·	·	·	·	·	·	✓	/htdocs/cgibin
	0.49–0.53	17 (-)	P	·	✓	·	·	·	·	·	·	✓	✓	/usr/sbin/upnpkits
CVE-2017-7240 (118)	0.95–1.00	3 (3)	V	·	·	·	✓	·	·	·	·	✓	·	/usr/sbin/httpd
	0.54–0.83	6 (-)	N	·	·	·	✓	·	·	·	·	✓	·	/usr/sbin/httpd
	0.50–0.53	23 (-)	N	·	·	·	·	✓	✓	✓	·	✓	✓	/usr/sbin/httpd
CVE-2018-10106 (2)	0.99–1.00	45 (42)	V	·	✓	✓	·	·	·	·	·	✓	✓	/htdcos/cgibin
	0.48–0.86	42 (41)	V	·	✓	·	·	·	·	·	✓	✓	✓	/htdocs/cgibin
	0.55–0.84	5 (-)	P	·	✓	·	·	·	·	·	✓	✓	✓	/htdocs/cgibin
CVE-2014-2962 (510)	0.96–1.00	2 (2)	V	·	·	·	✓	·	·	·	·	·	✓	/usr/www/cgi-bin/webproc
	0.66–0.86	13 (0)	V*	✓	·	✓	✓	·	·	·	·	·	✓	/usr/www/cgi-bin/webproc
	0.53	1 (-)	P	✓	·	·	·	·	·	·	·	✓	·	/usr/www/cgi-bin/webproc
CVE-2020-15893 (2)	0.86–1.00	43 (40)	V	·	✓	✓	·	·	·	·	·	✓	✓	/htdocs/cgibin
	0.96	1 (-)	P	·	·	✓	·	·	·	·	·	✓	✓	/htdocs/cgibin
	0.85	17 (12)	V	·	✓	·	·	·	·	·	·	✓	✓	/usr/sbin/upnpkits
	0.82	7 (7)	V	·	✓	·	·	·	·	·	✓	·	·	/htdocs/cgibin
	0.74–0.81	42 (-)	P	·	✓	·	·	·	·	·	✓	✓	✓	/htdocs/cgibin
	0.52	1 (1)	V	·	✓	·	·	·	·	·	·	·	✓	/htdocs/cgibin
CVE-2016-11021 (804)	0.97–1.00	11 (1)	V	·	✓	·	·	·	·	·	·	✓	·	/bin/alphapd
	0.97	2 (2)	V	·	✓	·	·	·	·	·	·	✓	·	/bin/goahead
	0.67–0.75	21 (-)	P	·	✓	✓	·	·	·	·	·	✓	·	/bin/alphapd
	0.60–0.67	9 (0)	V	·	✓	·	·	·	·	·	·	✓	·	/bin/alphapd
	0.59	1 (-)	P	·	·	✓	·	·	·	·	·	✓	·	/bin/alphapd
	0.50–0.59	18 (-)	N	·	✓	·	·	·	·	·	·	✓	·	/bin/alphapd
CVE-2017-6077 (186)	0.85–1.00	2 (2)	V	✓	·	·	·	·	·	·	·	·	✓	/usr/sbin/httpd
	0.5–0.85	1 (0)	V	✓	·	·	·	·	·	·	·	·	✓	/usr/sbin/httpd
CVE-2012-2765 (37)	0.72–1.00	7 (3)	V	·	·	·	✓	·	·	·	·	✓	·	/usr/sbin/httpd
	0.66	1 (-)	P	·	·	·	✓	·	·	·	·	✓	·	/usr/sbin/httpd
	0.58	2 (0)	V	·	·	·	✓	·	·	·	·	✓	·	/usr/sbin/httpd
	0.53	1 (-)	N	·	·	·	·	·	·	✓	·	✓	·	/usr/sbin/httpd
Linksys (53)	0.72–1.00	10 (1)	V	·	·	·	·	·	·	✓	·	✓	·	/usr/sbin/httpd
	0.53–0.64	7 (-)	P	·	·	·	·	·	·	✓	·	✓	·	/usr/sbin/httpd
CVE-2017-5521 (99, Stage 1)	0.98–1.00	40 (26)	V	✓	·	·	·	·	·	·	✓	·	·	/usr/sbin/httpd
	0.74–0.83	73 (-)	P	✓	·	·	·	·	·	·	✓	·	·	/usr/sbin/httpd
	0.79	2 (0)	V*	✓	·	·	·	·	·	·	✓	·	·	/usr/sbin/httpd
	0.51–0.52	11 (9)	V	✓	·	·	·	·	·	·	·	✓	·	/usr/sbin/httpd
	0.51–0.59	171 (-)	U	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	22 different binaries
CVE-2017-5521 (99, Stage2)	0.98–1.00	79 (26)	V	✓	·	·	·	·	·	·	✓	·	·	/usr/sbin/httpd
	0.76–0.92	36 (-)	P	✓	·	·	·	·	·	·	✓	·	·	/usr/sbin/httpd
	0.74–0.78	24 (6)	V	✓	·	·	·	·	·	·	·	✓	·	/usr/sbin/httpd
	0.68–0.73	9 (-)	P	✓	·	·	·	·	·	·	·	✓	·	/usr/sbin/httpd
	0.51–0.53	3 (-)	N	✓	·	·	·	·	·	·	·	·	✓	/usr/sbin/httpd
	0.51–0.51	1 (-)	P	✓	·	·	·	·	·	·	·	✓	·	/usr/sbin/upnpd
	0.51–0.51	14 (3)	V	✓	·	·	·	·	·	·	·	✓	·	/usr/sbin/upnpd

Case Study of CVE-2016-6277

- ❖ Command injection in CGI parsing (NETGEAR)

```
cmd = cmd.replace(" ", "$IFS")
path = "/cgi-bin/;{}".format(cmd)

self.http_request(
    method="GET",
    path=path
)
```

- ❖ Simple patch based on a block list

```
if ( !strchr(uri, ';') && !strchr(uri, '`') && !strchr(uri, '$') && !strstr(uri, "..") )
{
```

Range	# of Samples	Is Vulnerable?	Vendor	Arch
0.95 ~ 1.00	29 (3 Ground Truths)	**Vulnerable**	Netgear	ARM
0.5 ~ 0.95	40	Patched	Netgear	ARM

- ❖ **BCSA can distinguish vulnerabilities from the patched ones**

Case Study of CVE-2015-2051

- ❖ Command injection in HNAP cgibin, D-Link
  - No parameter check, no authentication

/htdocs/cgibin

Range	# of Samples	Is Vulnerable?	Vender	Arch
0.81 ~ 1.00	5 (4 Ground Truths)	**Vulnerable**	D-Link	ARM
0.68 ~ 0.73	25	Patched	D-Link	ARM
0.58 ~ 0.75	6 (5 Ground Truths)	**Vulnerable**	D-Link, TRENDnet	MIPS
0.53 ~ 0.59	3	Patched	D-Link	MIPS
0.68	1	Patched	D-Link	MIPSEB
0.58 ~ 0.69	15 (14 Ground Truths)	**Vulnerable**	D-Link	MIPSEB
0.53	9	Patched	D-Link	MIPSEB
0.49 ~ 0.53	17	Patched	D-Link	MIPS, MIPSEB

/usr/sbin/upnpkits

Case Study of CVE-2017-7240

- ❖ Directory traversal in CGI parsing
- ❖ DD-WRT's httpd
  - – Designed to accept only allowed file types
  - – Customized images allow all file types

```
response = self.http_request(
    method="GET",
    path="/etc/passwd"
)
```

Range	# of Samples	Is Vulnerable?	Vendors
0.95 ~ 1.00	3 (3 Ground Truths)	**Vulnerable**	Belkin
0.54 ~ 0.83	6	Not Vulnerable	Belkin
0.50 ~ 0.53	23	Not Vulnerable	Asus, ZyXEL, linksys

- ❖ The vulnerability resides in the data section, but BCSA found it

➔ **BCSA can detect diversities in compile environments**

*DD-WRT: Open source IoT firmware

Case Study of CVE-2018-10106

- Permission bypass with a newline (AUTHORIZED_GROUP)

Range	# of Samples	Is Vulnerable?	Vendor
0.99 ~ 1.00	45 (42 Ground Truths)	**Vulnerable**	**D-Link, TRENDnet**
0.48 ~ 0.86	42 (41 Ground Truths)	**Vulnerable**	D-Link
	5	Patched	D-Link

- Same vulnerability appears in new **versions** (D-Link)
  - CVE-2018-10106, CVE-2019-17506, CVE-2019-20213, CVE-2020-9376

- Same vulnerability appears in different **vendors** (TRENDnet, with score: 1.0)
  - CVE-2018-7034

- Same vulnerability appears in different **architectures** (MIPS, MIPSEB, ARM)
  - MIPS: 0.65~1, ARM: 0.5~0.6

Case Study of CVE-2014-2962

❖ Directory traversal in parsing a "getpage" parameter in CGI

Range	# of Samples	Is Vulnerable?	Vender
0.96 ~ 1.00	2 (2 Ground Truths)	**Vulnerable**	Belkin
0.66 ~ 0.86	13	**Potentially Vulnerable**	**Belkin, TRENDnet, Netgear**
0.53	1	Patched	Netgear

❖ Similar/same vulnerability has existed from 2006 in multiple vendors

- CVE-2006-2337 D-Link
- CVE-2006-5607 Inca
- CVE-2006-5536 D-Link
- CVE-2014-2962 Belkin
- CVE-2015-7250 Zte
- CVE-2017-15647 Fiberhome
- CVE-2017-8770 Twsz

Case Study of CVE-2020-15893

- Command injection in parsing SSDP parameters in "/htdocs/cgibin"

Range	# of Samples	Is Vulnerable?	Vender	Arch
0.86 ~ 1.00	43 (40 Ground Truths)	**Vulnerable**	**D-Link, TRENDnet**	**MIPS, MIPSEB**
0.96	1	Patched	TRENDnet	**MIPS, MIPSEB**
0.85	17 (12 Ground Truths)	**Vulnerable**	**D-Link**	**/usr/sbin/upnpkits**
0.82	7 (7 Ground Truths)	**Vulnerable**	**D-Link**	ARM
0.74 ~ 0.81	42	Patched	D-Link	MIPS, MIPSEB, ARM
0.52	1 (1 Ground Truth)	**Vulnerable**	**D-Link**	MIPSEB

A debugging routine exists
_dtrace()

- Same vulnerability has multiple CVE (D-Link)
  - CVE-2019-20015, CVE-2019-20016, CVE-2019-20017
- Same vulnerability appears in newer versions (D-Link)
  - CVE-2020-15893, CVE-2021-29379

Case Study of CVE-2016-11021

❖ Command injection in a debugging feature

Range	# of Samples	Is Vulnerable?	Vender
0.97 ~ 1.00	13 (3 Ground Truths)	**Vulnerable**	**2 goahead servers**
0.67 ~ 0.75	21	Patched	D-Link, TRENDnet
0.60 ~ 0.67	9 (0 Ground Truths)	**Vulnerable**	**D-Link**
0.59	1	Patched	TRENDnet
0.50 ~ 0.59	18	Not Vulnerable	D-Link

Static binaries

❖ D-Link images mostly use "/bin/alphapd"

- Some use "/bin/goahead" with the same vulnerability
  - GoAhead, an open-source embedded webserver
  - D-Link customized GoAhead

Case Study of Linksys Vuln.

- Command injection vulnerability
  - Testing function calls the vulnerable function
  - After authentication, the vulnerable function can be called

Range	# of Samples	Is Vulnerable?	Vendor	Arch
0.72 ~ 1.00	10 (1 Ground Truths)	**Vulnerable**	Linksys	MIPS

  - 2 images are **vulnerable**
  - 3 images removed the test function, but still **vulnerable** after auth.
  - 5 images removed the vulnerable function call (actually **not vulnerable**)

Range	# of Samples	Is Vulnerable?	Vendor	Arch
0.53 ~ 0.64	7	Patched	Linksys	MIPS

  - Added sanitizer to validate input strings

Final Results of Baseband Software

- Preprocess firmware images with the firmware analysis heuristics
- Randomly select one sample for each unique vulnerability
- Query it for each firmware image (18 images, 1.4M funcs)

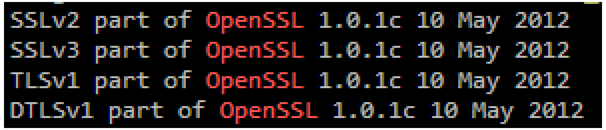
	Original TikNib		→	TikNib (+Heuristic Features)	
Top-k	# of Total Vulns	Percent		# of Total Vulns	Percent
1	36 / 56	64.29%		**48 / 56**	**85.71%**
5	41 / 56	73.21%		49 / 56	87.50%
10	41 / 56	73.21%		49 / 56	87.50%
50	42 / 56	75.00%		50 / 56	89.29%
100	44 / 56	78.57%		**52 / 56**	**92.86%**

Analyzing Open-Source Vulnerabilities

- Two well-known OpenSSL vulnerabilities
  - CVE-2015-1791: *ssl3_get_new_session_ticket*
    - Genius, Gemini, VulSeeker
  - CVE-2014-0160: *tls1_process_heartbeat*
    - Genius, Gemini, Multi-kMH, DiscovRE, SAFE

- Approach
  - Compile OpenSSL v1.0.1f with combinations of compiler options
  - Search all compiled functions in each firmware image
  - Average the similarity score for each function in each firmware image

- Ground truth
  - Match a function name and version string
  - CVE-2015-1791: 309 of 455 are vulnerable
  - CVE 2014-0160: 34 of 222 are vulnerable



Version strings in *libssl.so*

Final Results of Two CVEs (Only Vulns)

CVE-2015-1791

	Original TikNib		TikNib + Heuristic Features	
Top-k	# of Vulns	Percent	# of Vulns	Percent
1	252 / 309	81.55%	**309 / 309**	**100%**
5	284 / 309	91.91%	309 / 309	100%
10	293 / 309	94.82%	309 / 309	100%
50	294 / 309	95.15%	309 / 309	100%
100	294 / 309	95.15%	309 / 309	100%

CVE-2014-0160

	Original TikNib		TikNib + Heuristic Features	
Top-k	# of Vulns	Percent	# of Vulns	Percent
1	4 / 34	11.76%	**34 / 34**	**100%**
5	17 / 34	50.00%	34 / 34	100%
10	17 / 34	50.00%	34 / 34	100%
50	32 / 34	94.12%	34 / 34	100%
100	34 / 34	100%	34 / 34	100%

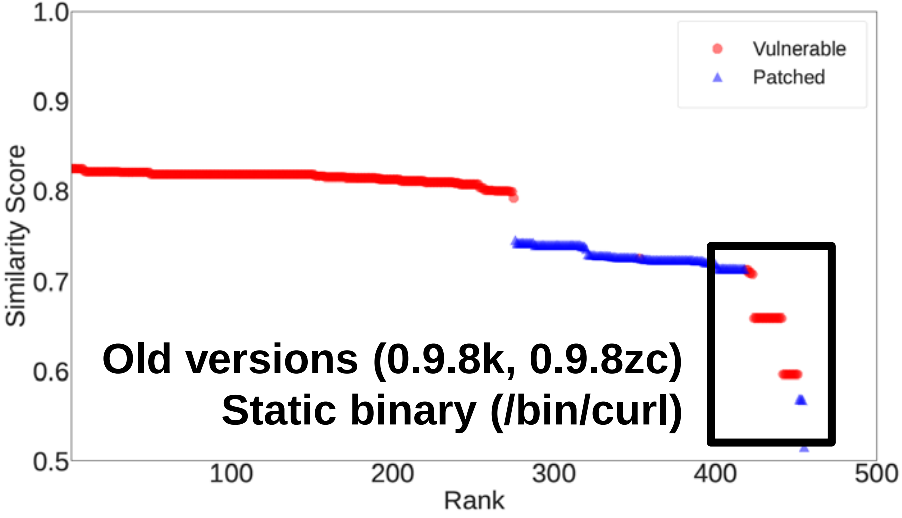
Final Results of Two CVEs (Inc. Patched)

		Original TikNib		TikNib + Heuristic Features	
	Top-k	# of Funcs	Percent	# of Funcs	Percent
CVE-2015-1791	1	252 / 455	55.38%	**455 / 455**	**100%**
	5	284 / 455	62.42%	455 / 455	100%
	10	293 / 455	64.40%	455 / 455	100%
	50	337 / 455	74.07%	455 / 455	100%
	100	382 / 455	83.96%	455 / 455	100%
	Top-k	# of Funcs	Percent	# of Funcs	Percent
CVE-2014-0160	1	7 / 222	3.15%	**215 / 222**	**96.85%**
	5	29 / 222	13.06%	215 / 222	96.85%
	10	44 / 222	19.82%	222 / 222	100%
	50	110 / 222	49.55%	222 / 222	100%
	100	158 / 222	71.17%	222 / 222	100%

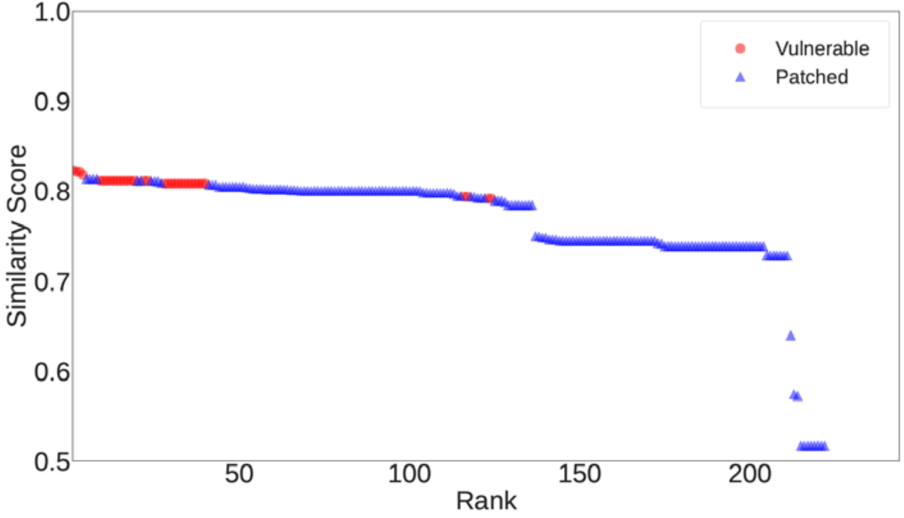
Similarity Score (Vulnerable vs Patched)

❖ Vulnerable functions are ranked higher than patched functions
  – Queried *OpenSSL v1.0.1f*

CVE-2015-1791 (309 of 455 are vulns)



CVE-2014-0160 (34 of 222 are vulns)



Comparison Results of CVE-2015-1791

- Top-k results of all functions in all firmware images (***NOT*** each image)
- Gemini and VulSeeker utilized 4643 firmware images (**unavailable**)
- TikNib utilized 1,124 firmware images (FirmAE)

	Gemini		VulSeeker		TikNib (O0-O3)		TikNib (O2-O3)		TikNib (+Heuristics)	
Top-k	# of Funcs	%	# of Funcs	%	# of Funcs	%	# of Funcs	%	# of Funcs	%
1	1	100%	1	100%	**1**	**100%**	**1**	**100%**	**1**	**100%**
5	2	40%	3	60%	**5**	**100%**	**5**	**100%**	**5**	**100%**
10	4	40%	6	60%	**9**	**90%**	**10**	**100%**	**10**	**100%**
50	36	72%	41	82%	19	38%	**46**	**92%**	**50**	**100%**
100	75	75%	83	83%	50	50%	**82**	**82%**	**100**	**100%**

Firmware images are highly likely compiled with O2-O3

Limitation and Future Works

- Developing other effective features
  - Type recovery (NDSS'11, SIGPLAN'13, SEC'17, CCS'18, …)
    - Type-related features are effective
    - # of arguments, each argument type, function return type
    - All benchmark tests achieved ROC AUC close to **1.0**
  - Inter-procedural analysis
    - Optimization affects function in-lining
  - Inter-binary analysis
    - Handle static binaries

- Determining whether a detected function is indeed vulnerable
  - Function-level: e.g., leverage symbolic execution
  - Binary-level: e.g., emulate a target binary and check dynamically
  - Firmware-level: e.g., analyze vulnerabilities spread over multiple binaries

  ➔ Leave as future work

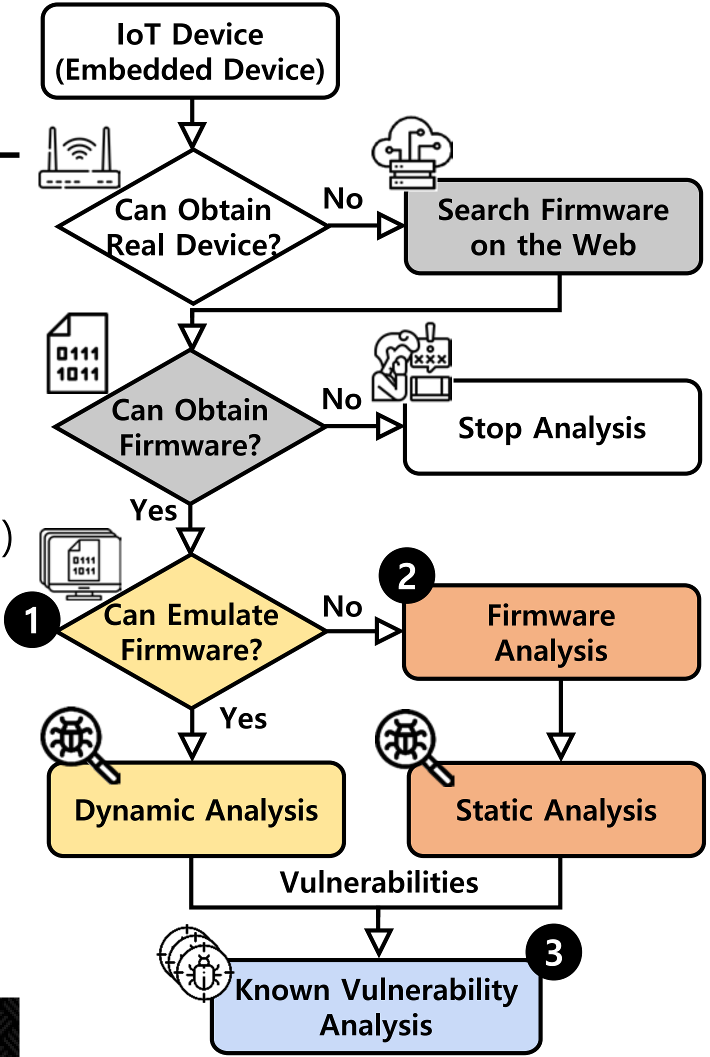
Conclusion and Lessons Learned

- Existing studies focused on **complex and novel** approaches without releasing **neither** dataset **nor** full source code
  - Systematized terms, features, benchmarks in existing literature
  - Built a comprehensive benchmark (BinKit)
  - ➔ **Demonstrated a simple model with pre-semantic features (TikNib) is effective!**

- **A few** analyzed IoT devices, **none** targeted custom binaries
  - Established IoT vulnerability ground truth dataset (FirmKit)
  - Systematized heuristic features into TikNib
  - ➔ **Successfully identified vulnerabilities (442 in custom binaries, 343 in OpenSSL libraries)**
  - ➔ **BCSA is effective in IoT vulnerability analysis as many devices share similar codebases**

Analysis Roadmap

1. Firmware Emulation Problem
   - Successful emulation (16.28% ➔ 79.36%)
   - ➔ Wireless routers, IP cameras
2. Firmware Analysis Problem
   - Successful analysis (595 funcs ➔ 73,874 funcs)
   - ➔ Smartphone baseband
3. Known Vulnerability Analysis Problem
   - Effective bug discovery (253 ➔ 442)
   - ➔ Both device categories



Thesis Conclusion

❖ Existing studies focused on developing novel approaches, disregarding heuristics

➔ To remove security threats in convoluted IoT ecosystem, heuristics are inevitable!

	Summary of Results	
1	Emulate firmware	183 images ➔ 892 images (16.28% ➔ 79.36%)
	Discover vulnerability	known vulns.: 320, new vulns: 95
2	Identify function boundary	595 funcs ➔ 73,874 funcs (18 images, on avg.)
	Detect target function	18 decoders, 0 false positives
	Discover software bug	functional bugs: 78, known vulns: 6, new vulns: 50
3	Systematic study of BCSA	Systematized features and benchmarks of 43 studies. Built a benchmark dataset of 243K bins for 36M funcs.
	Discover known vulnerability	Built a vulnerability dataset of 323 vulns in 1,142 images. 442 vulns in custom bins, 343 vulns in OpenSSL libs.

➔ Developing/Systematizing heuristics helped test 1,256 vulnerabilities

Thesis Conclusion

❖ Existing studies focused on developing novel approaches, disregarding heuristics

➔ To remove security threats in convoluted IoT ecosystem, heuristics are inevitable!

	Summary of Results	
1	Emulate firmware	183 images ➔ 892 images (16.28% ➔ 79.36%)
	Discover vulnerability	known vulns.: 320, new vulns: 95
2	Identify function boundary	595 funcs ➔ 73,874 funcs (18 images, on avg.)
	Det	
	Dis	
3	Sys	
	Dis	s.

Although heuristics seem to be trivial and not novel, developing/systematizing "dirty" heuristics is necessary to enable large-scale vulnerability analysis of IoT devices

➔ ities

Publications (Related to This Thesis)

- IoT dynamic analysis
  - **Dongkwan Kim**, Eunsoo Kim, Mingeun Kim, Yeongjin Jang, and Yongdae Kim, "Enabling Large-scale Emulation of IoT Firmware with Heuristic Workarounds", *IEEE S&P*
  - Mingeun Kim, **Dongkwan Kim**, Eunsoo Kim, Suryeon Kim, Yeongjin Jang, and Yongdae Kim, "FirmAE: Towards Large-Scale Emulation of IoT Firmware for Dynamic Analysis", *ACSAC 2020*
- IoT static analysis
  - Eunsoo Kim∗, **Dongkwan Kim∗**, CheolJun Park, Insu Yun, and Yongdae Kim, "BaseSpec: Comparative Analysis of Baseband Software and Cellular Specifications for L3 Protocols", *NDSS 2021*
- Binary code similarity analysis
  - **Dongkwan Kim**, Eunsoo Kim, Sang Kil Cha, Sooel Son, and Yongdae Kim, "Revisiting Binary Code Similarity Analysis using Interpretable Feature Engineering and Lessons Learned", *IEEE Transactions on Software Engineering (major revision, under review)*

***: co-first author**

Publications (Other IoT devices)

- Wearable IoT devices
  - **<u>Dongkwan Kim</u>**, Suwan Park, Kibum Choi, Yongdae Kim, "BurnFit: Analyzing and Exploiting Wearable Devices", *WISA 2015*
- Sensors in IoT devices
  - Yunmok Son, Hocheol Shin, **<u>Dongkwan Kim</u>**, Youngseok Park, Juhwan Noh, Kibum Choi, Jungwoo Choi, and Yongdae Kim, "Rocking Drones with Intentional Sound Noise on Gyroscopic Sensors", *USENIX Security 2015*
- USIM in smartphone
  - Shinjo Park, Suwan Park, Insu Yun, **<u>Dongkwan Kim</u>**, Yongdae Kim, "Analyzing Security of Korean USIM-based PKI Certificate Service", *WISA 2014*
- Blockchain
  - Sangsup Lee, Daejun Kim, **<u>Dongkwan Kim</u>**, Sooel Son, and Yongdae Kim, "Who Spent My EOS? On the (In)Security of Resource Management of EOS.IO", *WOOT 2019*

Publications (Cellular Infrastructure)

- Control plane
  - Byeongdo Hong, Shinjo Park, Hongil Kim, **Dongkwan Kim**, Hyunwook Hong, Hyunwoo Choi, Jean-Pierre Seifert, Sung-Ju Lee, and Yongdae Kim, "Peeking over the Cellular Walled Gardens - A Method for Closed Network Diagnosis", *IEEE Transactions on Mobile Computing (TMC 2018)*
- Data plane
  - Sangwook Bae, Mincheol Son, **Dongkwan Kim**, CheolJun Park, Jiho Lee, Sooel Son, Yongdae Kim, "Watching the Watchers: Practical Video Identification Attack in LTE Networks", *USENIX Security 2021*
  - Hyunwook Hong, Hyunwoo Choi, **Dongkwan Kim**, Hongil Kim, Byeongdo Hong, Jiseong Noh, and Yongdae Kim, "When Cellular Networks Met IPv6: Security Problems of Middleboxes in IPv6 Cellular Networks", *EuroS&P 2017*
  - Hyunwook Hong, Hongil Kim, Byeongdo Hong, **Dongkwan Kim**, Hyunwoo Choi, Eunkyu Lee and Yongdae Kim, "7. Pay As You Want: Bypassing Charging System in Operational Cellular Networks", *WISA 2016*
- Hybrid plane
  - Hongil Kim*, **Dongkwan Kim***, Minhee Kwon, Hyungseok Han, Yeongjin Jang, Dongsu Han, Taesoo Kim, and Yongdae Kim, "Breaking and Fixing VoLTE: Exploiting Hidden Data Channels and Mis-implementations", *CCS 2015*

***: co-first author**

Participated Projects (Selected)

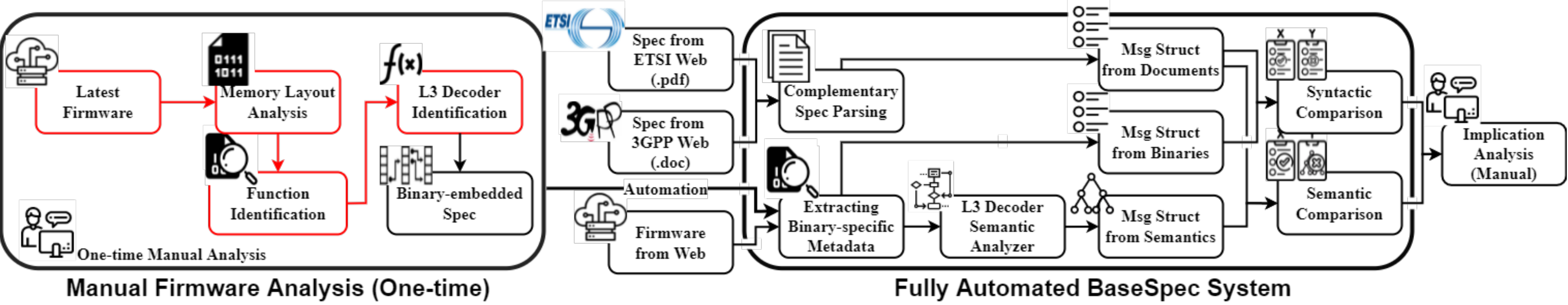
- Industrial Projects
  - Researcher, *Samsung*, “An Industry-academia Task with Samsung Electronics Device Solutions Business”, 2020
  - Leader, *Samsung*, “Organizing Samsung Capture-the-flag (SCTF)”, 2017-2018
  - Researcher, *SK Telecom*, “A Study on the Security Vulnerability Analysis and Response Method of LTE Networks”, 2016-2017
  - Researcher, *Hyundai NGV*, “A Security Vulnerability Analysis of Smartcar Core Modules”, 2016-2017
  - Researcher, *SK Telecom*, “A Study on the Security Analysis and Response Method of LTE Networks”, 2015-2016
  - Researcher, *Samsung*, “A Security Analysis of Samsung SmartTV”, 2014-2015
- Governmental Projects (including US)
  - Leader, *AOARD*, “Cyber Physical Analysis of System Software Survivability by Stimulating Sensors on Drones”, 2020-2021
  - Leader, *NSR*, “A study on the Android-based Security Analysis Technology”, 2020
  - Researcher, *IITP*, “A study on the Security of Random Number Generator and Embedded Devices”, 2017-2019
  - Leader, *NSR*, “A study on the Firmware Emulation Technology for Linux-based Routers”, 2017
  - Researcher, *IITP*, “A development of Automated Reverse Engineering and Vulnerability Detection Base Technology through Binary Code Analysis”, 2016-2018
  - Leader, *KAIST*, “A CAPTCHA Design based on Human Perception Characteristics”, 2016
  - Leader, *NSR*, “A Study on the Vulnerability Analysis Method of Domestic/International Smartcars”, 2015
  - Researcher, *KISA*, “A Study on the Analysis of Technology and Security Threats in LTE Femtocell”, 2013-2014

Thank You!

dkay@kaist.ac.kr

BACKUP SLIDES

BaseSpec Separation



- ❖ Dongkwan
  - – Developing heuristics for firmware structure analysis
- ❖ Eunsoo
  - – Extract embedded specification from the baseband binary
  - – Compare the message structures implemented in the baseband and specification

Type Features Should Be Studied

- Function type does not change unless source code varies
  - # of arguments
  - Leverage Jaccard index for checking argument type, return type

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

- All benchmark tests achieved ROC AUC over **0.99**

- vs VulSeeker

ROC AUC

Dataset	Packages	Arch	Compilers	VulSeeker	Ours	Ours (Type)
ASE1	2	3	1	0.99	0.9727	**0.9924**
ASE2	**5**	3	1	-	0.9764	**0.9931**
ASE3	5	**6**	2	0.8849	0.9782	**0.9939**
ASE4	5	**8**	**9**	-	0.9584	**0.9841**

Larger Dataset

➔ **Features from type information is effective**
(NDSS'11, SIGPLAN'13, SEC'17, CCS'18, …)

Failure Case Analysis

- **Errors in IDA Pro (72% use IDA Pro)**
  - Cannot handle some registers in GCC and Clang
    - GCC: ‘gp’, Clang: ‘s0’, ‘v0’
  - incomplete CFGs
    - switch table, data in code section

- **Diversity of compiler backends**
  - Conditional instructions for ARM
    - GCC: MOVLE, MOVGT, Clang: MOV + JLE, MOV + JGT
  - Instruction pointer loading
    - GCC: call __x86.get_pc_thunk.bx, Clang: call $+5

- **Architecture-specific macros**
  - mul_add in OpenSSL

➔ **Need to consider these cases carefully!**

Case Study of CVE-2017-6077

- Command injection in parsing “ping_IPAddr” (for debug)

- 3 firmware images have the functionality
  - If functionality exists, vulnerable

Range	# of Samples	Is Vulnerable?	Vendor
0.85 ~ 1.00	2 (2 Ground Truths)	**Vulnerable**	Netgear
0.5 ~ 0.85	1	**Vulnerable**	Netgear

➡ **Gets additional parameter for VPN**

Case Study of CVE-2012-2765

- Password disclosure in login page (client-side password checking)

Range	# of Samples	Is Vulnerable?	Vender
0.72 ~ 1.00	7 (3 Ground Truths)	**Vulnerable**	Belkin
0.66	1	Patched	Belkin
0.58	2 (0 Ground Truths)	**Vulnerable**	Belkin
0.53	1	Not Vulnerable	Linksys

(Latest)
No debug routine

- Vulnerability in a webpage, but detected at the binary level
- Three images are released at Feb. 2018
  - One patched, the other two were still vulnerable

➔ **BCSA can detect diversities in compile environments**

Case Study of CVE-2017-5521

- Two staged vulnerability
  - **Stage1: leak the device id**
  - Stage2: leak the user id/password using the device id

Range	# of Samples	Is Vulnerable?	Vender	Arch
0.98 ~ 1.00	40 (26 Ground Truths)	**Vulnerable**	Netgear	ARM
0.74 ~ 0.83	73	Patched	Netgear	ARM
0.79	2 (0 Ground Truths)	**Incorrectly Patched**	Netgear	ARM
0.51 ~ 0.52	11 (9 Ground Truths)	**Vulnerable**	Netgear	**Stripped**
0.52 ~ 0.59	151	Unknown	Netgear, TRENDnet, D-Link,…	ARM, MIPS, MIPSEB

Old firmware images, different binaries

Case Study of CVE-2017-5521

- Two staged vulnerability
  - Stage1: leak the device id
  - **Stage2: leak the user id/password using the device id**

Range	# of Samples	Is Vulnerable?	Vender	Arch
0.98 ~ 1.00	79 (26 Ground Truths)	**Vulnerable**	Netgear	ARM
0.76 ~ 0.92	36	Patched	Netgear	ARM
0.74 ~ 0.78	24 (6 Ground Truths)	**Vulnerable**	Netgear	MIPS
0.68 ~ 0.73	9	Patched	Netgear	MIPS
0.51 ~ 0.53	3	No Functionality	Netgear	MIPSEB
0.51 ~ 0.51	1	Patched	Netgear	MIPS
0.51 ~ 0.51	14 (3 Ground Truths)	**Vulnerable**	Netgear	MIPS

Old images,
Different implementation

Failure Case Study – Baseband B7

- Too small function
  - CFG size: 3

- Code for a new routine
  - To support Dual SIMs
  - ➔ Takes a large portion

Size of the target function is critical

Old versions

```
int __fastcall iemm_68_0x227_IEMM_NETWORK_DAYLIGHT_SAVING_TIMEdecoder(int a1, int a2)
{
  int result; // r0

  if ( a1 )
  {
    result = sub_40D54270(&unk_423CDFE0, a1, a2);
    dword_423CDFE8 = a2;
  }
  else
  {
    result = 0;
    dword_423CDFE8 = 0;
  }
  return result;
}
```

Recent versions

Function for Dual SIMs

```
void __fastcall iemm_68_0x22f_IEMM_NETWORK_DAYLIGHT_SAVING_TIMEdecoder(int a1, unsigned int a2,
{
  int v5; // r0
  const char *v6; // r1
  int v7; // r2

  if ( a1 )
  {
    v5 = sub_40E90476(-1, (const char *)a2, a3);
    sub_40E90452((int)&byte_422F2E80[9500 * v5 + 7316], a1, a2);
    *(_DWORD *)&byte_422F2E80[9500 * sub_40E90476(-1, v6, v7) + 7324] = a2;
  }
  else
  {
    *(_DWORD *)&byte_422F2E80[9500 * sub_40E90476(-1, (const char *)a2, a3) + 7324] = 0;
  }
  JUMPOUT(0x413D4AA4);
}
```