black hat ASIA 2020
OCTOBER 1-2, 2020
BRIEFINGS

UCSB UNIVERSITY OF TWENTE.

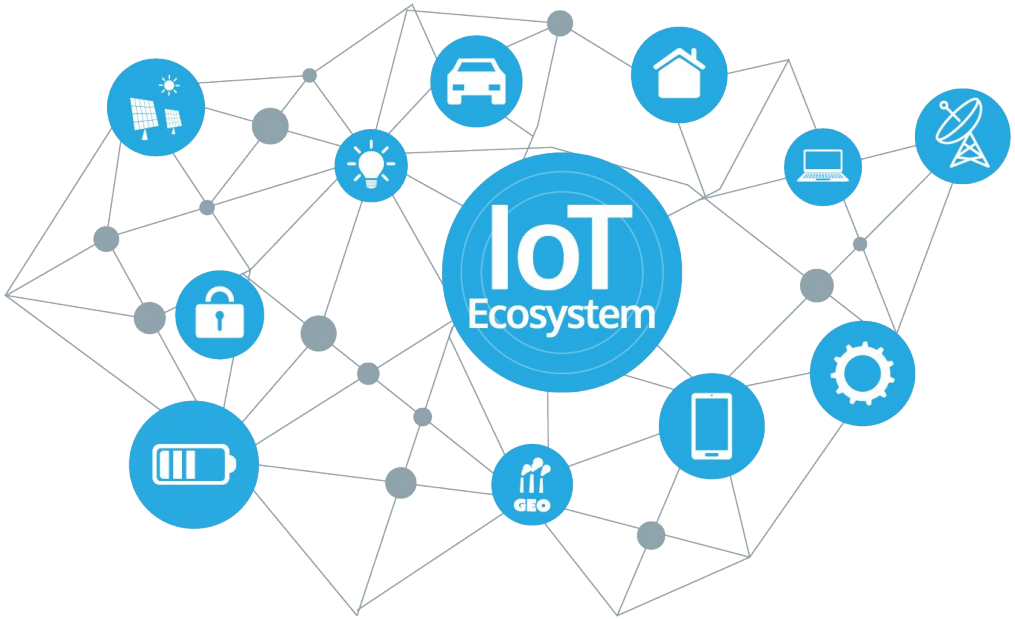
Identifying Multi-Binary Vulnerabilities in Embedded Firmware at Scale

Nilo Redini
UC Santa Barbara

Andrea Continella
University of Twente

Research co-authors: Aravind Machiry, Ruoyu (Fish) Wang, Chad Spensky, Yan Shoshitaishvili, Christopher Kruegel, Giovanni Vigna

Today's IoT Landscape



Today's IoT Landscape

Smart lock has a security vulnerability that leaves homes open for attacks

The lock isn't able to receive updates, which means the flaw allowing hackers to break in will always be present.

Alfred Ng December 11, 2019 4:45 AM PST

LISTEN - 02:50

TechNewsWorld > Security > Privacy | **Next Article in Privacy**

Webcam Maker Takes FTC's Heat for Internet-of-Things Security Failure

By Richard Adhikari
Sep 5, 2013 3:56 PM PT

Print
Email

Home > Hacking

FEATURE

What is a botnet? When armies of infected IoT devices attack

Controlling thousands or even millions of devices gives cyber attackers the upper hand to deliver malware or conduct a DDoS attack.

By Maria Korolov

Contributing Writer, CSO | JUN 27, 2019 3:00 AM PDT

Inside the infamous Mirai IoT Botnet: A Retrospective Analysis

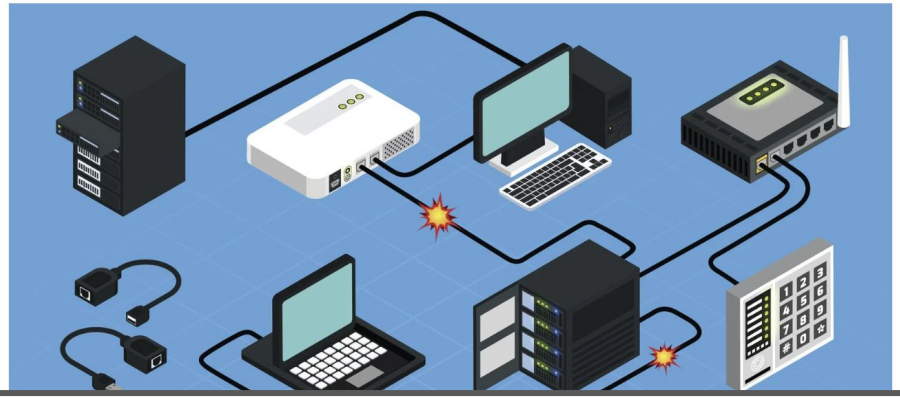
Tweet

Guest Post Guest Author

December 14, 2017 11:41 AM

LILY HAY NEWMAN SECURITY 12.09.16 07:00 AM

The Botnet That Broke the Internet Isn't Going Away



20y old vulnerabilities are back!

FEATURING
STACK OVERFLOWS
GETS()
SCANF()
ASLR WHO?

What makes firmware different?

Firmware Analysis 101: Challenges

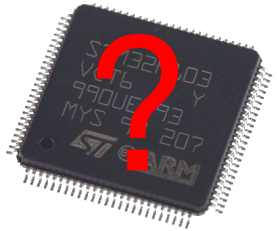
Hardware-dependent

Unique, minimal **environments** with **non-standard** configurations

Several different architectures

- ARM, MIPS, x86, PowerPC, etc.
- Sometimes proprietary

Manage **external peripherals**, often using custom code



Firmware Analysis

- Dynamic Analysis
  - Emulation, coverage-guided fuzzing, etc…
  - Currently **not generic**, too **unreliable**

Limitations of Dynamic Analysis

Firmware is heavily **hardware-dependent**

- Peripherals
- Interrupts
- DMA

Minimal, **non-standard** environments

- Shared memory
- Hardcoded addresses (MMIO)
- Unsupported/unmodeled architectures

Firmware Analysis

- Dynamic Analysis
  - Emulation, Coverage-guided Fuzzing, etc…
  - Currently **not generic**, too **unreliable**

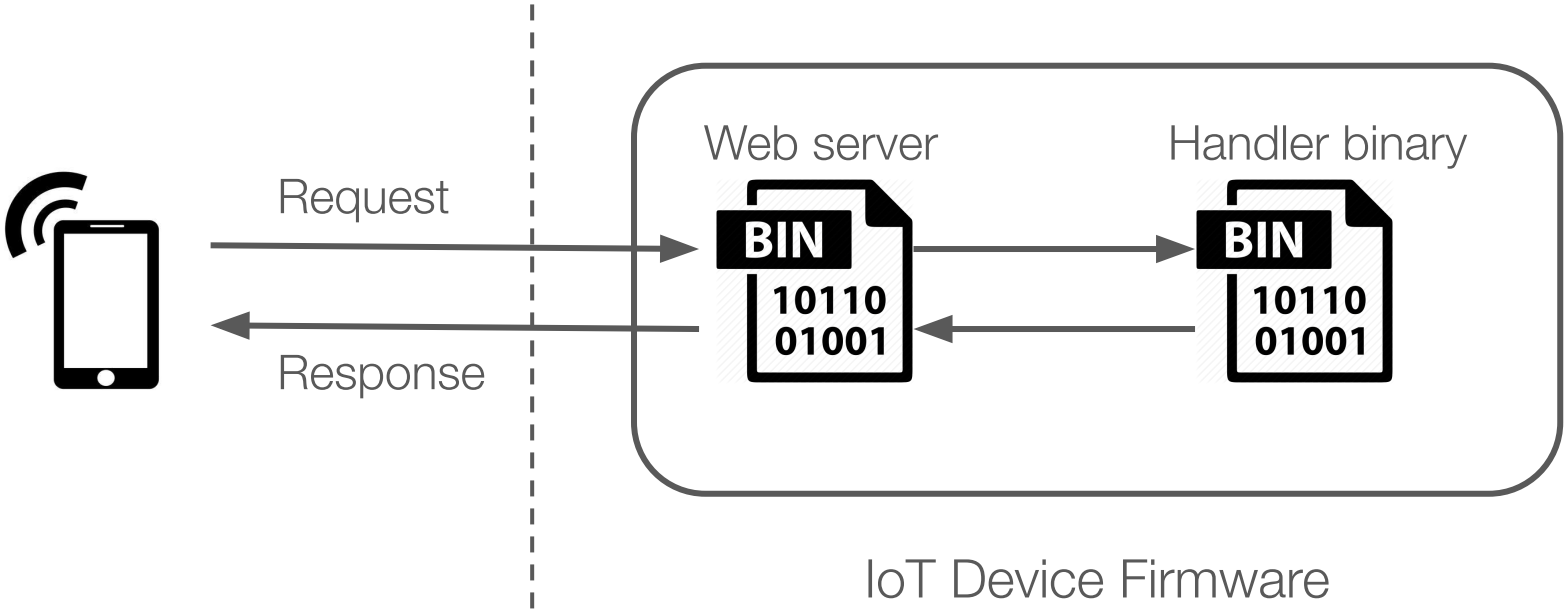
- Static Analysis
  - Current approaches are **insufficient**
  - Too many **false positives**

Firmware is Multi-binary!

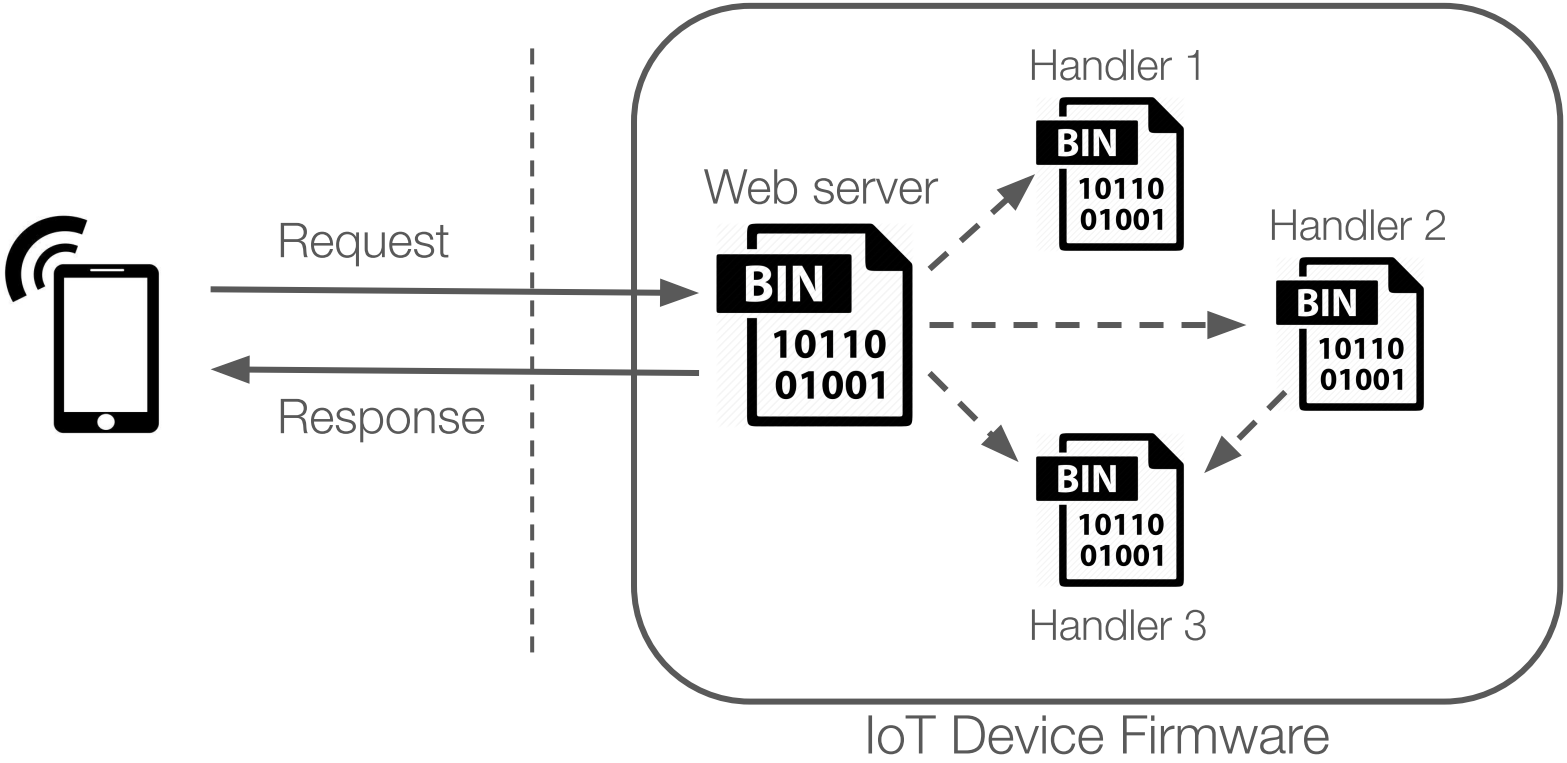
86% of firmware is Linux-based

On average (900+ samples), a firmware sample contains **157** binaries!

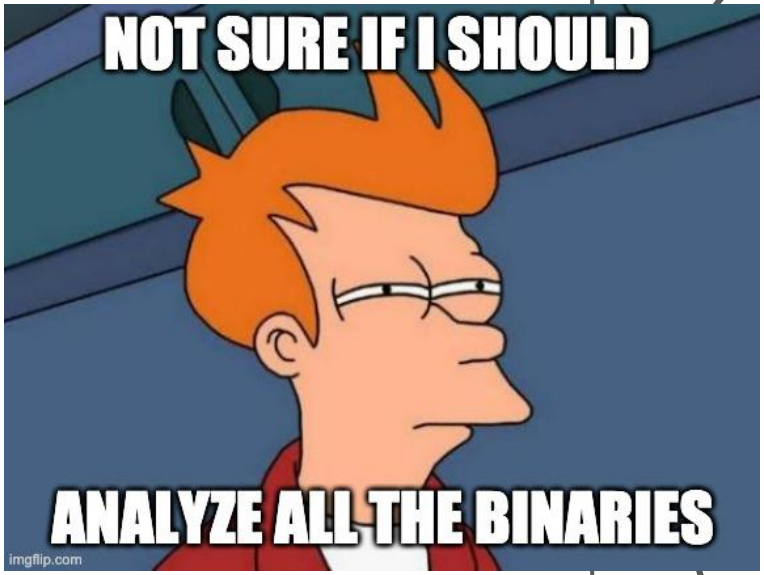
Typical Firmware Architecture

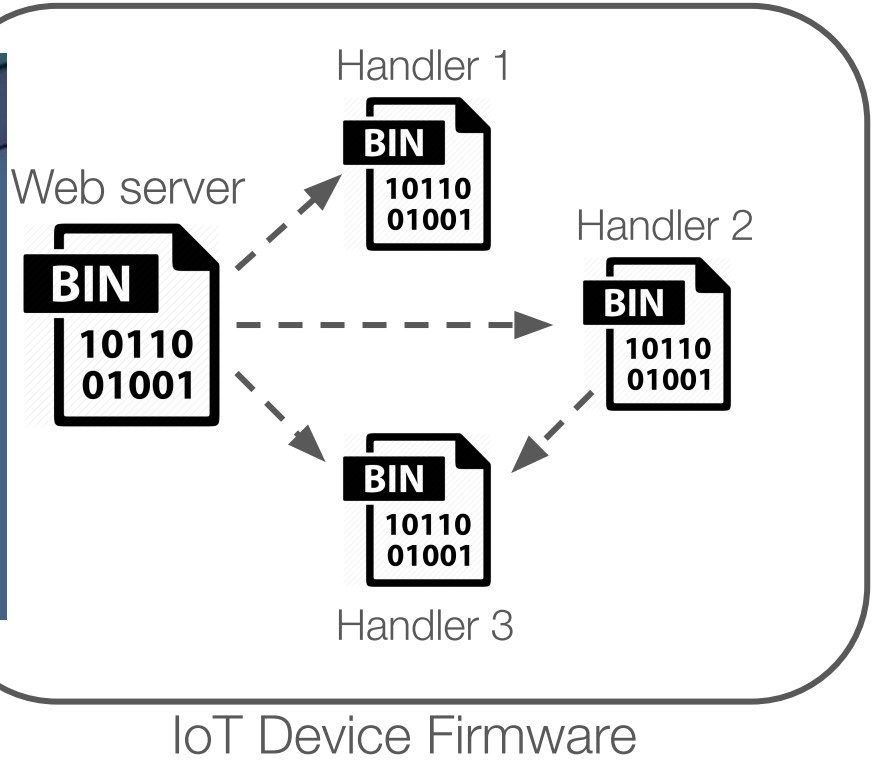


Typical Firmware Architecture



Typical Firmware Architecture





Real-World Example

```
int process_req(char *query,char *log_path){
  char *q, arg[128];
  char log_dir[128];
  if (!(q=strchr(query, "op=")))
    return -1;
  strcpy(arg, q); // query string argument
  strcpy(log_dir, dirname(log_path));
  // ...
  return 0;
}

int main(int argc,char *argv[],char *envp[])
{
  char *query = getenv("QUERY_STRING");
  char *log_path = getenv("LOG_PATH");
  process_req(query, log_path);
}
```

Real-World Example

```
int process_req(char *query,char *log_path){
  char *q, arg[128];
  char log_dir[128];
  if (!(q=strchr(query, "op=")))
    return -1;
  strcpy(arg, q); // query string argument
  strcpy(log_dir, dirname(log_path));
  // ...
  return 0;
}

int main(int argc,char *argv[],char *envp[])
{
  char *query = getenv("QUERY_STRING");
  char *log_path = getenv("LOG_PATH");
  process_req(query, log_path);
}
```

Real-World Example

```
char* parse_URI(Req* req) {
  char* p = req[1];
  if (!strncmp(p, "<soap:AddRule", 13))
    return p; // unconstrained
  // ...
  if (strlen(p) > 127)
    p[127] = 0;
  return p; // constrained data
}

int serve_request(Req *req) {
  char *data = parse_URI(req);
  setenv("QUERY_STRING", data, 1);
  setenv("LOG_PATH", "/var/log/1.log");
  execve(get_handler(req));
}
```

```
int process_req(char *query,char *log_path){
  char *q, arg[128];
  char log_dir[128];
  if (!(q=strchr(query, "op=")))
    return -1;
  strcpy(arg, q); // query string argument
  strcpy(log_dir, dirname(log_path));
  // ...
  return 0;
}

int main(int argc,char *argv[],char *envp[])
{
  char *query = getenv("QUERY_STRING");
  char *log_path = getenv("LOG_PATH");
  process_req(query, log_path);
}
```

Our work (& Takeaways)

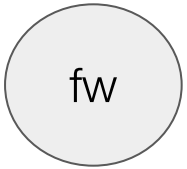
Firmware is mostly composed by **multiple interacting binaries**

Modeling **multi-binary interactions** is fundamental for effective analysis

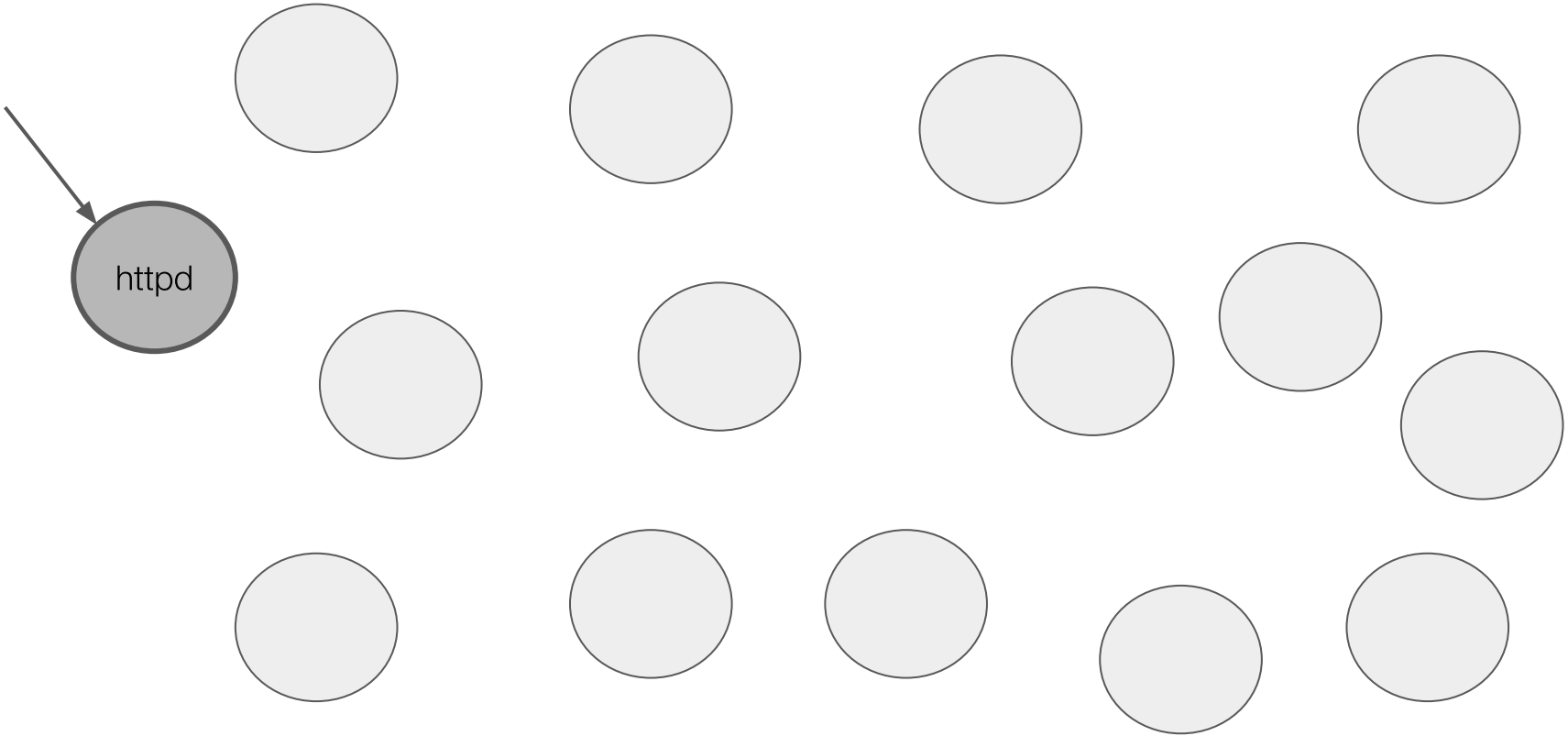
We introduced static analysis techniques to perform **multi-binary taint analysis**

Karonte can effectively **discover unknown bugs** radically **reducing** the number of **false positives**

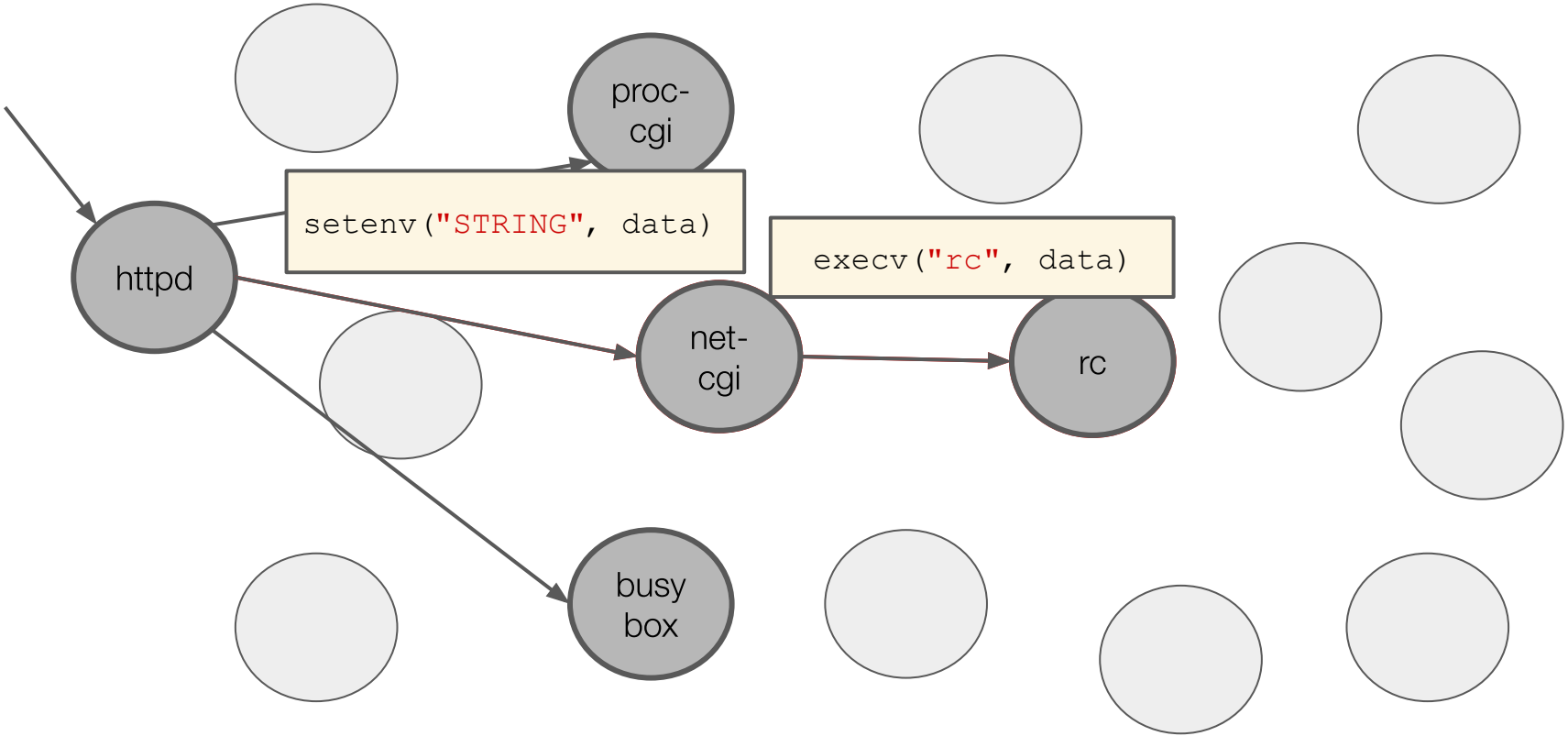
Karonte in a Nutshell



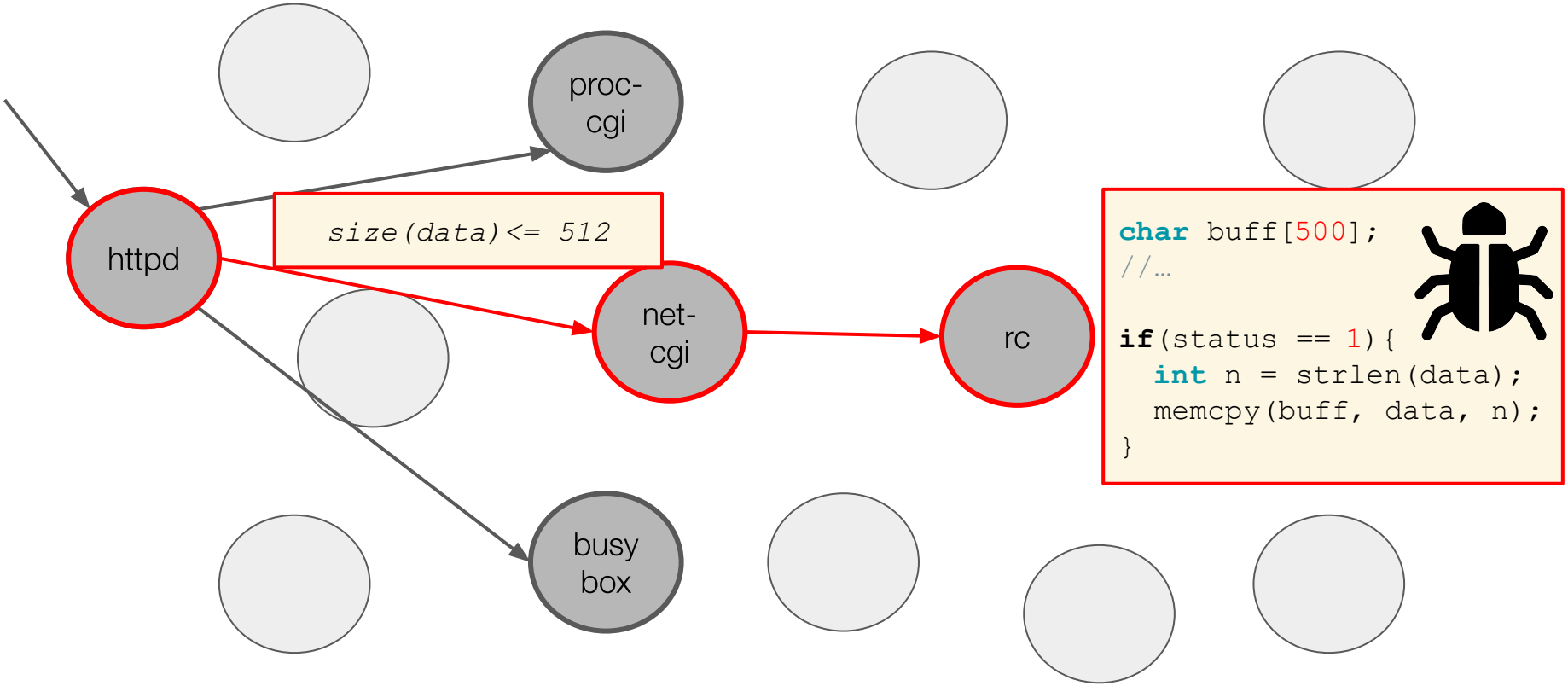
Karonte in a Nutshell



Karonte in a Nutshell



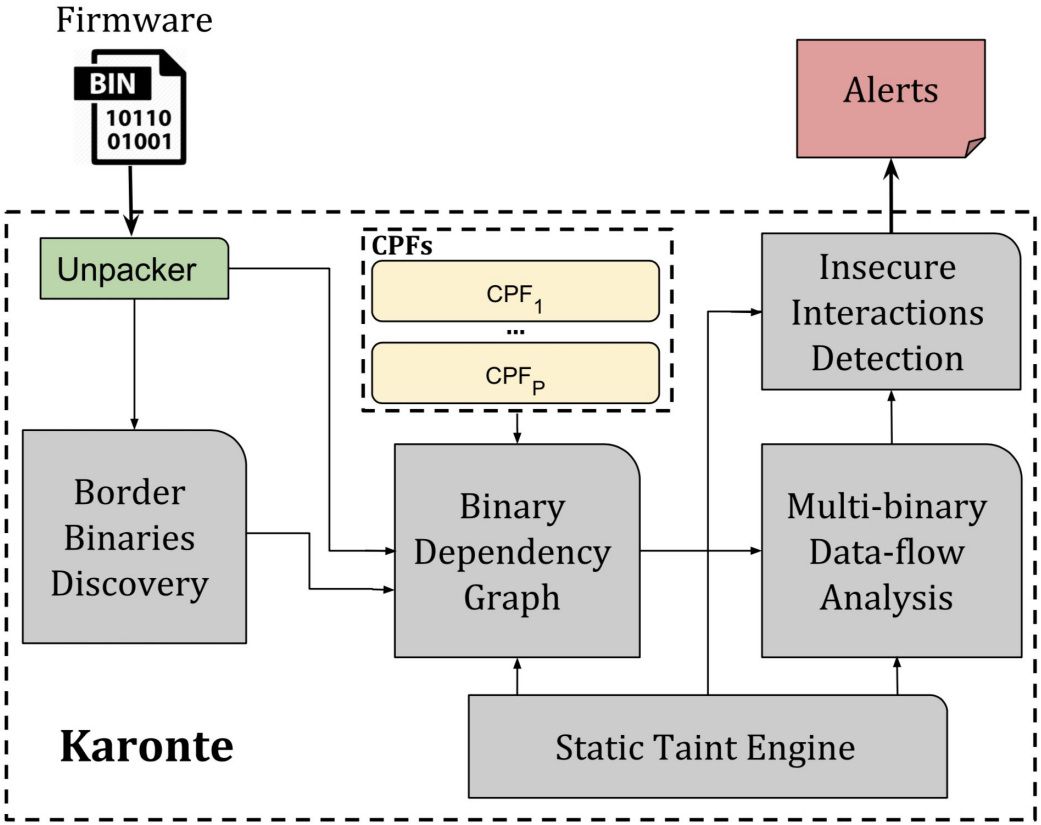
Karonte in a Nutshell



Karonte: System Architecture

Nilo Redini
UC Santa Barbara

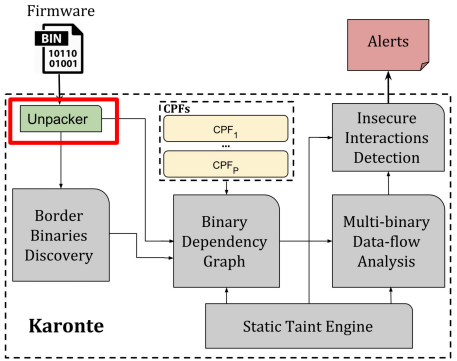
Karonte


Firmware
BIN
10110
01001
Alerts
Unpacker
CPFs
CPF1
...
CPFP
Insecure Interactions Detection
Border Binaries Discovery
Binary Dependency Graph
Multi-binary Data-flow Analysis
Karonte
Static Taint Engine

Firmware Pre-processing

Firmware unpacking: `binwalk`

```
➜  karonte binwalk wr1043v2.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             TP-Link firmware header, firmware version: 1.-22540.3, image vers
, product version: 272826370, kernel load address: 0x0, kernel entry point: 0x80002000, kernel
length: 512, rootfs offset: 868363, rootfs length: 1048576, bootloader offset: 7077888, bootloa
69424         0x10F30         Certificate in DER format (x509 v3), header length: 4, sequence l
92272         0x16870         U-Boot version string, "U-Boot 1.1.4 (Sep 25 2013 - 08:43:53)"
92448         0x16920         CRC32 polynomial table, big endian
131584        0x20200         TP-Link firmware header, firmware version: 0.0.3, image version:
duct version: 272826370, kernel load address: 0x0, kernel entry point: 0x80002000, kernel offse
h: 512, rootfs offset: 868363, rootfs length: 1048576, bootloader offset: 7077888, bootloader l
132096        0x20400         LZMA compressed data, properties: 0x5D, dictionary size: 33554432
e: 2488384 bytes
1180160       0x120200        Squashfs filesystem, little endian, version 4.0, compression:lzma
00 inodes, blocksize: 131072 bytes, created: 2013-09-25 01:01:42
```



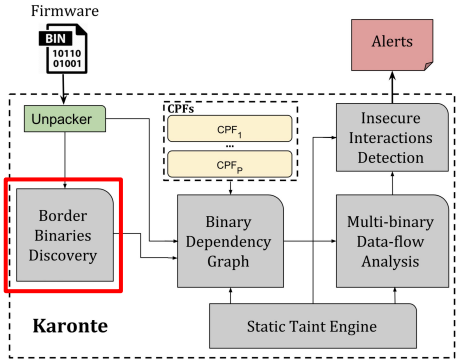
Border Binaries Discovery

Discover binaries exporting the IoT device functionality

Intuition: they need *parsing*!

Identify **network parsing** functions:

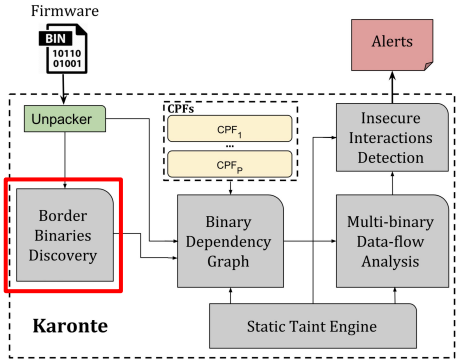
- *# basic blocks* (**bb**)
- *# memory comparisons* (**cmp**)
- *# branches* (**br**)
- *# network-related keywords (e.g., "<soap")* (**net**)
- *Data flow between a recv and a mem comparison* (**conn**)



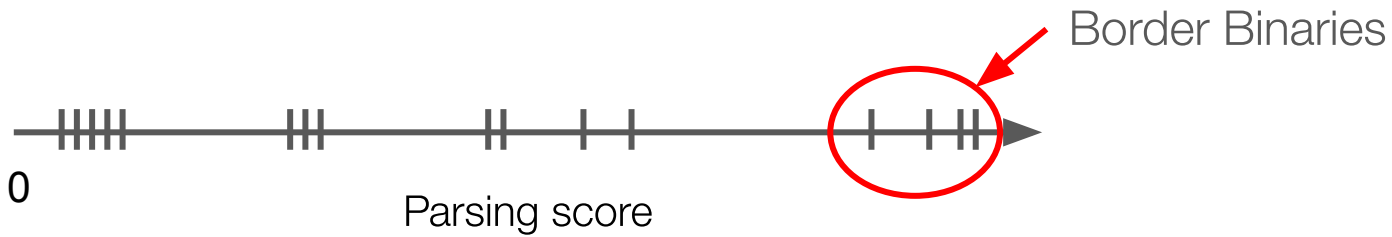
Border Binaries Discovery

$$ps_j = \left(\sum_{i \in \{bb, br, cmp\}} k_i * \#i_j \right) * (1 + k_n * \#net_j) * (1 + k_c * \#conn_j)$$

$$ps_b = max(\{ps_j \mid \forall j \in get\_functions(b)\})$$



Cluster binaries using their parsing scores (DBSCAN)

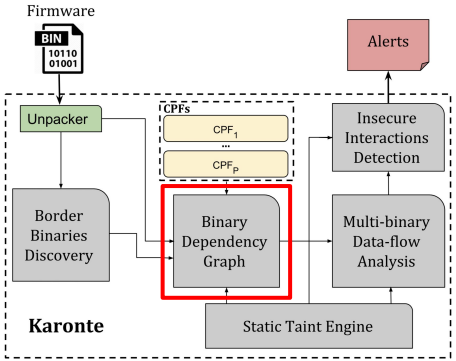


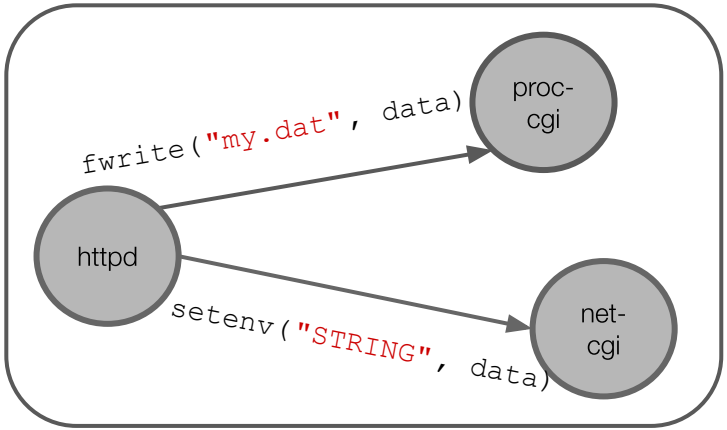
Binary Dependency Graph (BDG)

Directed graph that models multi-binary communications

We use our static taint engine to

1. Taint data compared against network-related keywords
2. Run analysis to detect data sharing (**CPFs**)





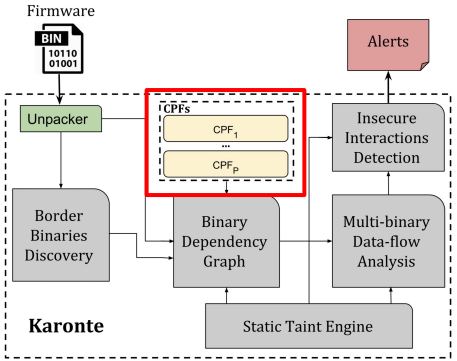
Communication Paradigm Finder

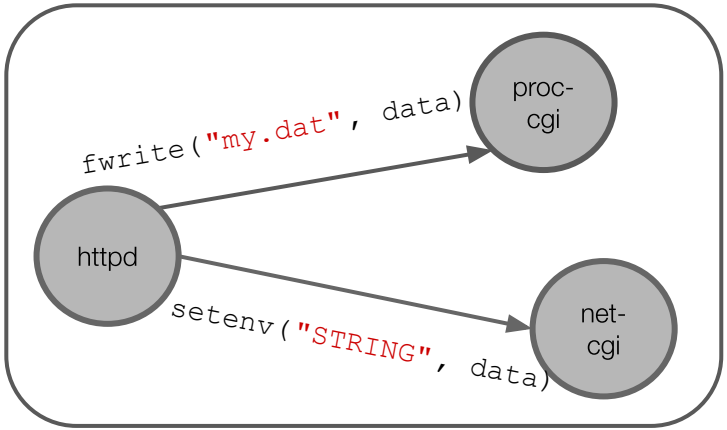
CPF modules reason about the different Inter-Process Communication paradigms (e.g., socket-based communication)

Provide a CPF for each IPC paradigm

CPF duties:

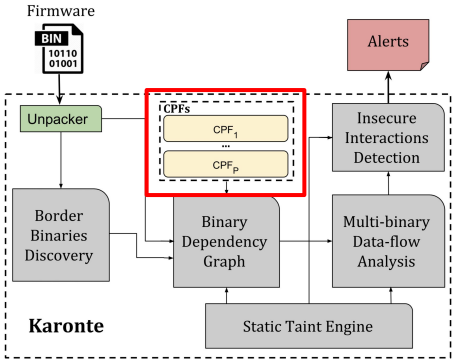
- Data Key Recovery
- Flow Direction Determination (**Setter** vs **Getter**)
- Binary Set Magnification





Communication Paradigm Finder

We provide Karonte with a generic CPF to cover cases where IPC is unknown



Intuition: data-key used as “index” to set or get data

```
user_struct['HTTP_REQUEST'] = req_data;
```

Communication Paradigm Finder

```
char* parse_URI(Req* req) {
  char* p = req[1];
  if (!strncmp(p, "<soap:AddRule", 13))
    return p;
  // ...
  if (strlen(p) > 127)
    p[127] = 0;
  return p;
}

int serve_request(Req *req) {
  char *data = parse_URI(req);
  setenv("QUERY_STRING", data, 1);
  setenv("LOG_PATH", "/var/log/1.log");
  execve(get_handler(req));
}
```

```
int process_req(char *query,char *log_path){
  char *q, arg[128];
  char log_dir[128];
  if (!(q=strchr(query, "op=")))
    return -1;
  strcpy(arg, q); // query string argument
  strcpy(log_dir, dirname(log_path));
  // ...
  return 0;
}

int main(int argc,char *argv[],char *envp[])
{
  char *query = getenv("QUERY_STRING");
  char *log_path = getenv("LOG_PATH");
  process_req(query, log_path);
}
```

Communication Paradigm Finder

```
char* parse_URI(Req* req) {
  char* p = req[1];
  if (!strncmp(p, "<soap:AddRule", 13))
    return p;
  // ...
  if (strlen(p) > 127)
    p[127] = 0;
  return p;
}

int serve_request(Req *req) {
  char *data = parse_URI(req);
  setenv("QUERY_STRING", data, 1);
  setenv("LOG_PATH", "/var/log/1.log");
  execve(get_handler(req));
}
```

```
int process_req(char *query,char *log_path){
  char *q, arg[128];
  char log_dir[128];
  if (!(q=strchr(query, "op=")))
    return -1;
  strcpy(arg, q); // query string argument
  strcpy(log_dir, dirname(log_path));
  // ...
  return 0;
}

int main(int argc,char *argv[],char *envp[])
{
  char *query = getenv("QUERY_STRING");
  char *log_path = getenv("LOG_PATH");
  process_req(query, log_path);
}
```

Communication Paradigm Finder

```
char* parse_URI(Req* req) {
  char* p = req[1];
  if (!strncmp(p, "<soap:AddRule", 13))
    return p;
  // ...
  if (strlen(p) > 127)
    p[127] = 0;
  return p;
}

int serve_request(Req *req) {
  char *data = parse_URI(req);
  setenv("QUERY_STRING", data, 1);
  setenv("LOG_PATH", "/var/log/1.log");
  execve(get_handler(req));
}
```

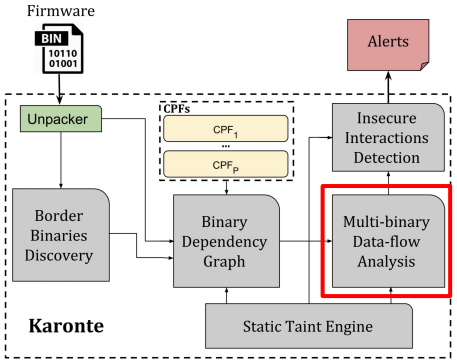
```
int process_req(char *query,char *log_path){
  char *q, arg[128];
  char log_dir[128];
  if (!(q=strchr(query, "op=")))
    return -1;
  strcpy(arg, q); // query string argument
  strcpy(log_dir, dirname(log_path));
  // ...
  return 0;
}

int main(int argc,char *argv[],char *envp[])
{
  char *query = getenv("QUERY_STRING");
  char *log_path = getenv("LOG_PATH");
  process_req(query, log_path);
}
```

Multi-binary Data-flow Analysis

Track how the data is propagated through the binary and collect the constraints applied to such data.

We propagate the data with its constraints to successor binaries in the BDG

Propagate the **least strict** set of constraints (tractable analysis)



Multi-binary Data-flow Analysis

```
char* parse_URI(Req* req) {
  char* p = req[1];
  if (!strncmp(p, "<soap:AddRule", 13))
    return p; // unconstrained
  // ...
  if (strlen(p) > 127)
    p[127] = 0;
  return p; // constrained data
}

int serve_request(Req *req) {
  char *data = parse_URI(req);
  setenv("QUERY_STRING", data, 1);
  setenv("LOG_PATH", "/var/log/1.log");
  execve(get_handler(req));
}
```

```
int process_req(char *query,char *log_path){
  char *q, arg[128];
  char log_dir[128];
  if (!(q=strchr(query, "op=")))
    return -1;
  strcpy(arg, q); // query string argument
  strcpy(log_dir, dirname(log_path));
  // ...
  return 0;
}

int main(int argc,char *argv[],char *envp[])
{
  char *query = getenv("QUERY_STRING");
  char *log_path = getenv("LOG_PATH");
  process_req(query, log_path);
}
```

Multi-binary Data-flow Analysis

```
char* parse_URI(Req* req) {
  char* p = req[1];
  if (!strncmp(p, "<soap:AddRule", 13))
    return p; // unconstrained
  // ...
  if (strlen(p) > 127)
    p[127] = 0;
  return p; // constrained data
}

int serve_request(Req *req) {
  char *data = parse_URI(req);
  setenv("QUERY_STRING", data, 1);
  setenv("LOG_PATH", "/var/log/1.log");
  execve(get_handler(req));
}
```

```
int process_req(char *query,char *log_path){
  char *q, arg[128];
  char log_dir[128];
  if (!(q=strchr(query, "op=")))
    return -1;
  strcpy(arg, q); // query string argument
  strcpy(log_dir, dirname(log_path));
  // ...
  return 0;
}

int main(int argc,char *argv[],char *envp[])
{
  char *query = getenv("QUERY_STRING");
  char *log_path = getenv("LOG_PATH");
  process_req(query, log_path);
}
```

Multi-binary Data-flow Analysis

```
char* parse_URI(Req* req) {
  char* p = req[1];
  if (!strncmp(p, "<soap:AddRule", 13))
    return p; // unconstrained
  // ...
  if (strlen(p) > 127)
    p[127] = 0;
  return p; // constrained data
}

int serve_request(Req *req) {
  char *data = parse_URI(req);
  setenv("QUERY_STRING", data, 1);
  setenv("LOG_PATH", "/var/log/1.log");
  execve(get_handler(req));
}
```

```
int process_req(char *query,char *log_path){
  char *q, arg[128];
  char log_dir[128];
  if (!(q=strchr(query, "op=")))
    return -1;
  strcpy(arg, q); // query string argument
  strcpy(log_dir, dirname(log_path));
  // ...
  return 0;
}

int main(int argc,char *argv[],char *envp[])
{
  char *query = getenv("QUERY_STRING");
  char *log_path = getenv("LOG_PATH");
  process_req(query, log_path);
}
```

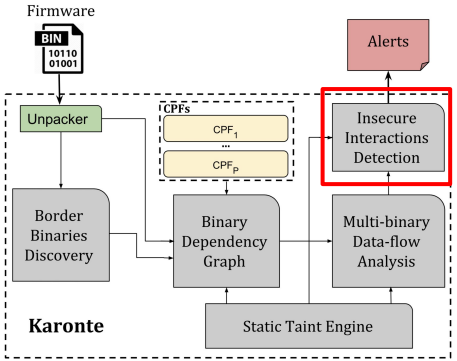
Unconstrained data

Insecure Interaction Detection

Taint engine to uncover insecure attacker-controlled data flows

Type of vulnerabilities

- Memory-corruption
  - Buffer overflows
- Denial of service (DoS) vulnerabilities
  - Attacker-controlled loops



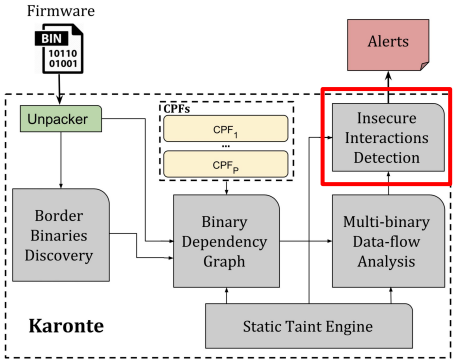
Insecure Interaction Detection

Taint data set or received by another binary

Raise an alert if **tainted && under constrained** data reaches a sink

Sinks

- Memcpy-like functions (semantic analysis)
- Dereference of a tainted variable
- Comparisons of tainted variables in loops' conditions



Insecure Interaction Detection

```
char* parse_URI(Req* req) {
  char* p = req[1];
  if (!strncmp(p, "<soap:AddRule", 13))
    return p; // unconstrained
  // ...
  if (strlen(p) > 127)
    p[127] = 0;
  return p; // constrained data
}

int serve_request(Req *req) {
  char *data = parse_URI(req);
  setenv("QUERY_STRING", data, 1);
  setenv("LOG_PATH", "/var/log/1.log");
  execve(get_handler(req));
}
```

```
int process_req(char *query,char *log_path){
  char *q, arg[128];
  char log_dir[128];
  if (!(q=strchr(query, "op=")))
    return -1;
  strcpy(arg, q); // query string argument
  strcpy(log_dir, dirname(log_path));
  // ...
  return 0;
}

int main(int argc,char *argv[],char *envp[])
{
  char *query = getenv("QUERY_STRING");
  char *log_path = getenv("LOG_PATH");
  process_req(query, log_path);
}
```

Insecure Interaction Detection

```
char* parse_URI(Req* req) {
  char* p = req[1];
  if (!strncmp(p, "<soap:AddRule", 13))
    return p; // unconstrained
  // ...
  if (strlen(p) > 127)
    p[127] = 0;
  return p; // constrained data
}

int serve_request(Req *req) {
  char *data = parse_URI(req);
  setenv("QUERY_STRING", data, 1);
  setenv("LOG_PATH", "/var/log/1.log");
  execve(get_handler(req));
}
```

```
int process_req(char *query,char *log_path){
  char *q, arg[128];
  char log_dir[128];
  if (!(q=strchr(query, "op=")))
    return -1;
  strcpy(arg, q); // query string argument
  strcpy(log_dir, dirname(log_path));
  // ...
  return 0;
}

int main(int argc,char *argv[],char *envp[])
{
  char *query = getenv("QUERY_STRING");
  char *log_path = getenv("LOG_PATH");
  process_req(query, log_path);
}
```

Insecure Interaction Detection

```
char* parse_URI(Req* req) {
  char* p = req[1];
  if (!strncmp(p, "<soap:AddRule", 13))
    return p; // unconstrained
  // ...
  if (strlen(p) > 127)
    p[127] = 0;
  return p; // constrained data
}

int serve_request(Req *req) {
  char *data = parse_URI(req);
  setenv("QUERY_STRING", data, 1);
  setenv("LOG_PATH", "/var/log/1.log");
  execve(get_handler(req));
}
```

```
int process_req(char *query,char *log_path){
  char *q, arg[128];
  char log_dir[128];
  if (!(q=strchr(query, "op=")))
    return -1;
  strcpy(arg, q); // query string argument
  strcpy(log_dir, dirname(log_path));
  // ...
  return 0;
}

int main(int argc,char *argv[],char *envp[])
{
  char *query = getenv("QUERY_STRING");
  char *log_path = getenv("LOG_PATH");
  process_req(query, log_path);
}
```

Insecure Interaction Detection

```
char* parse_URI(Req* req) {
  char* p = req[1];
  if (!strncmp(p, "<soap:AddRule", 13))
    return p; // unconstrained
  // ...
  if (strlen(p) > 127)
    p[127] = 0;
  return p; // constrained data
}

int serve_request(Req *req) {
  char *data = parse_URI(req);
  setenv("QUERY_STRING", data, 1);
  setenv("LOG_PATH", "/var/log/1.log");
  execve(get_handler(req));
}
```

```
int process_req(char *query,char *log_path){
  char *q, arg[128];
  char log_dir[128];
  if (!(q=strchr(query, "op=")))
    return -1;
  strcpy(arg, q); // query string argument
  strcpy(log_dir, dirname(log_path));
  // ...
  return 0;
}

int main(int argc,char *argv[],char *envp[])
{
  char *query = getenv("QUERY_STRING");
  char *log_path = getenv("LOG_PATH");
  process_req(query, log_path);
}
```

Insecure Interaction Detection

```
char* parse_URI(Req* req) {
  char* p = req[1];
  if (!strncmp(p, "<soap:AddRule", 13))
    return p; // unconstrained
  // ...
  if (strlen(p) > 127)
    p[127] = 0;
  return p; // constrained data
}

int serve_request(Req *req) {
  char *data = parse_URI(req);
  setenv("QUERY_STRING", data, 1);
  setenv("LOG_PATH", "/var/log/1.log");
  execve(get_handler(req));
}
```

```
int process_req(char *query,char *log_path){
  char *q, arg[128];
  char log_dir[128];
  if (!(q=strchr(query, "op=")))
    return -1;
  strcpy(arg, q); // query string argument
  strcpy(log_dir, dirname(log_path));
  // ...
  return 0;
}

int main(int argc,char *argv[],char *envp[])
{
  char *query = getenv("QUERY_STRING");
  char *log_path = getenv("LOG_PATH");
  process_req(query, log_path);
}
```

Taint Engine

Improved version of angr's taint engine

- Path prioritization strategy
- Taint dependencies

Path Prioritization

Prioritize *more interesting* paths

```
char* parse(char* start) {
  char* end = start + strlen(start) - 1;
  while (start < end)
    switch(*start[0]) {
      case '=': return start + 1;
      case ';': return 0;
      default: start++
    }
}

void serve(char* input) {
  char dst[512], cmd = parse(input);
  unsigned int n = strlen(cmd);

  if (n >= 512) return -1;
  strcpy(dst, cmd);
}
```

Path Prioritization

Prioritize *more interesting* paths

```
char* parse(char* start) {
  char* end = start + strlen(start) - 1;
  while (start < end)
    switch(*start[0]) {
      case '=': return start + 1;
      case ';': return 0;
      default: start++
    }
}

void serve(char* input) {
  char dst[512], cmd = parse(input);
  unsigned int n = strlen(cmd);

  if (n >= 512) return -1;
  strcpy(dst, cmd);
}
```

Path Prioritization

Prioritize *more interesting* paths

```
char* parse(char* start) {
  char* end = start + strlen(start) - 1;
  while (start < end)
    switch(*start[0]) {
      case '=': return start + 1;
      case ';': return 0;
      default: start++
    }
}

void serve(char* input) {
  char dst[512], cmd = parse(input);
  unsigned int n = strlen(cmd);

  if (n >= 512) return -1;
  strcpy(dst, cmd);
}
```

Prioritize paths that propagate the taint, and de-prioritize those that remove it

- Find basic blocks that return non-constant data
- Follow its return before considering others

Taint Dependencies

Alleviate overtainting issue

```
char* parse(char* start) {
  char* end = start + strlen(start) - 1;
  while (start < end)
    switch(*start[0]) {
      case '=': return start + 1;
      case ';': return 0;
      default: start++
    }
}

void serve(char* input) {
  char dst[512], cmd = parse(input);
  unsigned int n = strlen(cmd);

  if (n >= 512) return -1;
  strcpy(dst, cmd);
}
```

Taint Dependencies

Alleviate overtainting issue

```
char* parse(char* start) {
  char* end = start + strlen(start) - 1;
  while (start < end)
    switch(*start[0]) {
      case '=': return start + 1;
      case ';': return 0;
      default: start++
    }
}

void serve(char* input) {
  char dst[512], cmd = parse(input);
  unsigned int n = strlen(cmd);

  if (n >= 512) return -1;
  strcpy(dst, cmd);
}
```

Smart untaint strategies

- Create dependency between taint tag of `n` and the taint tag of `cmd`
- If `n` is untainted, `cmd` gets untainted as well
- `strcpy` does not generate the false positive

Taint Dependencies

Alleviate overtainting issue

```
char* parse(char* start) {
  char* end = start + strlen(start) - 1;
  while (star
    switch(*s
      case '=
      case ';
      default
    }
}

void serve(char* input) {
  char dst[512], cmd = parse(input);
  unsigned int n = strlen(cmd);

  if (n >= 512) return -1;
  strcpy(dst, cmd);
}
```

We automatically find functions that implement `strlen` semantically equivalent code, and create taint tag dependencies

Smart untaint strategies

etween taint
tag of `cmd`
gets untainted

enerate the false
positive

All of this is nice.. but does it work?

In-depth Evaluation

Firmware from 53 devices from 7 different vendors

46 new zero-day software bugs (CVE-2017-14948) and rediscover another 5

Number alerts decreased from an average of **945** to an average of **5** per firmware

Alert reduction of **two orders of magnitude** and a **low false-positive rate**

In-depth Evaluation

Firmware from 53 devices from 7 different vendors

46 new zero-day software bugs (CVE-2017-14948) and rediscover another 5

Number alerts decreased from an average of **945** to an average of **5** per firmware
Alert reduction of **two orders of magnitude** and a **low false-positive rate**

	ALL		
Vendor	**No. Bins**	**No. Alerts**	**Avg Time**
NETGEAR	280	12,393	7 days
D-Link	143	7,299	3 days
TP-Link	110	13,104	3 days
Tenda	105	3,318	5 days
Total	**638**	**36,114**	**18 days**

In-depth Evaluation

Firmware from 53 devices from 7 different vendors

46 new zero-day software bugs (CVE-2017-14948) and rediscover another 5

Number alerts decreased from an average of **945** to an average of **5** per firmware
Alert reduction of **two orders of magnitude** and a **low false-positive rate**

	ALL			Karonte		
Vendor	**No. Bins**	**No. Alerts**	**Avg Time**	**No. Bins**	**No. Alerts**	**Avg Time**
NETGEAR	280	12,393	7 days	8	36	17 hours
D-Link	143	7,299	3 days	6	24	14 hours
TP-Link	110	13,104	3 days	5	2	1.5 hours
Tenda	105	3,318	5 days	6	12	1 hour
Total	**638**	**36,114**	**18 days**	**25**	**74**	**34 hours**

Large-scale Evaluation

899 firmware samples from **21** different vendors

348 (38.7%) samples contain **multi-binary** interactions

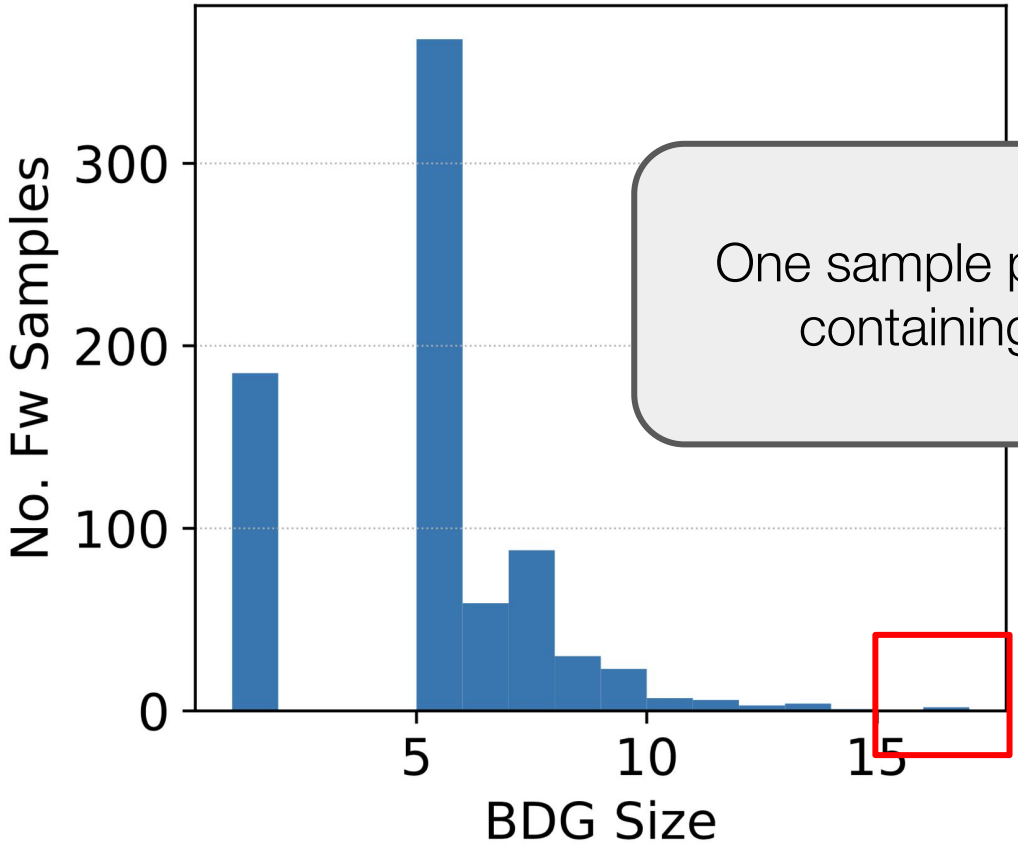
Karonte generated **1,003** alerts

- 2 alerts per sample on average
- Manually inspected 100 alerts
  - **44** to be true positive
  - **30** of them are **multi-binary** vulnerabilities

Large-scale Dataset

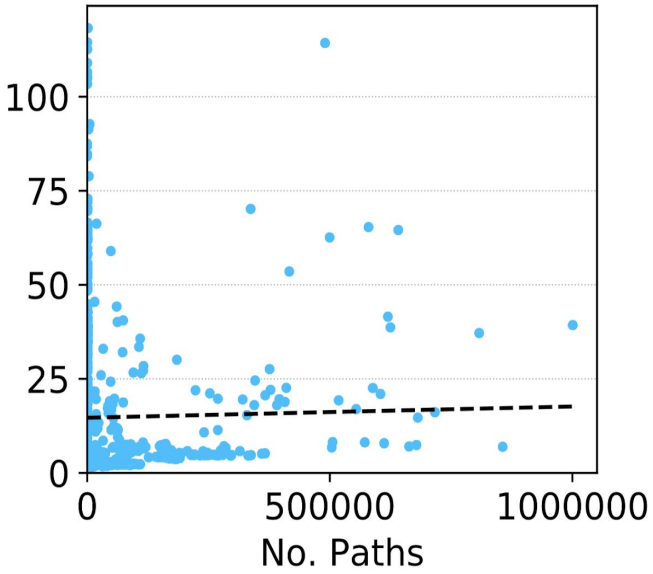
Vendor	# Firmware Samples	# Multi Binary (%)	# Binaries†	# Border Binaries†	BDG Size†	Subgraph Cadinality‡	Subgraph Depth‡	# Basic Blocks†	# Paths†	Explored Paths	Time† [hh:mm:ss]†
Airlink101	1	1 (100.0%)	94	5	8	4	1	9×10^{04}	1×10^{05}	68.58K	3:55:44
Belkin	6	1 (16.7%)	184	5	2	1	1	2×10^{05}	3×10^{81}	4.12K	0:49:46
Buffalo	3	0 (0.0%)	301	5	2	0	0	2×10^{06}	3×10^{14}	43.00	0:17:01
Cisco	21	6 (28.6%)	142	5	5	3	1	4×10^{05}	2×10^{22}	173.27K	5:36:15
D-Link	306	196 (64.1%)	103	3	3	1	1	7×10^{05}	3×10^{30}	41.64K	21:51:27
Foscam	5	5 (100.0%)	115	5	6	4	2	4×10^{05}	5×10^{15}	52.20K	18:01:00
Inmarsat	2	0 (0.0%)	640	5	5	0	0	2×10^{06}	9×10^{03}	3.10K	11:05:06
Linksys	12	1 (8.3%)	404	5	6	11	1	8×10^{05}	2×10^{305}	23.20K	3:32:36
NETGEAR	304	52 (17.1%)	115	5	5	3	1	5×10^{05}	4×10^{107}	82.83K	3:54:00
OpenWrt	12	1 (8.3%)	14	1	1	4	2	3×10^{04}	4×10^{15}	24.41K	1:06:16
Polycom	7	0 (0.0%)	130	4	3	0	0	1×10^{06}	2×10^{12}	1.01M	31:49:22
Supermicro	26	3 (11.5%)	209	5	5	2	1	4×10^{05}	2×10^{148}	12.16K	1:54:03
Synology	44	28 (63.6%)	679	3	3	1	1	5×10^{06}	1×10^{14}	4.55K	33:12:01
TP-Link	3	0 (0.0%)	200	5	5	0	0	7×10^{05}	1×10^{12}	2.00K	2:53:15
TRENDnet	55	26 (47.3%)	156	3	4	2	1	6×10^{05}	2×10^{118}	14.52K	22:59:12
Tenda	4	1 (25.0%)	332	5	5	1	1	6×10^{05}	2×10^{13}	13.04K	5:39:25
Tomato	51	11 (21.6%)	223	5	5	4	1	7×10^{05}	1×10^{26}	90.36K	9:40:55
Ubiquiti	15	7 (46.7%)	68	3	4	1	1	1×10^{05}	3×10^{08}	11.61K	3:06:21
Verizon	1	0 (0.0%)	10	5	5	0	0	1×10^{05}	5×10^{20}	2.49K	0:19:02
Zyxel	19	9 (47.4%)	153	5	6	3	1	3×10^{05}	4×10^{16}	260.87K	4:46:38
forceWare	2	0 (0.0%)	173	5	5	0	0	2×10^{05}	2×10^{03}	3.00	0:30:18

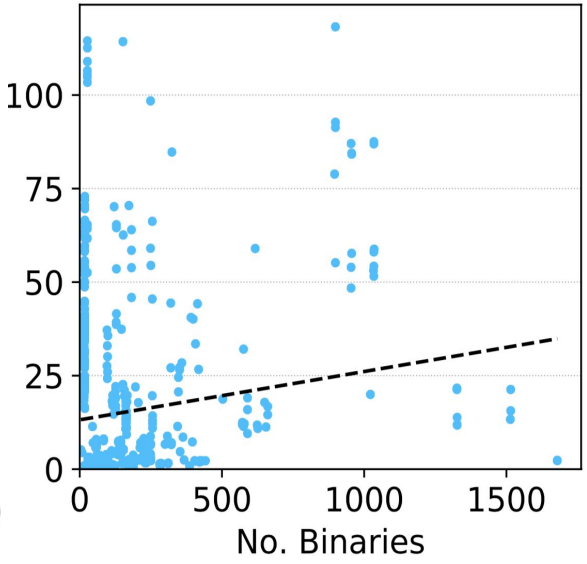
BDG Size

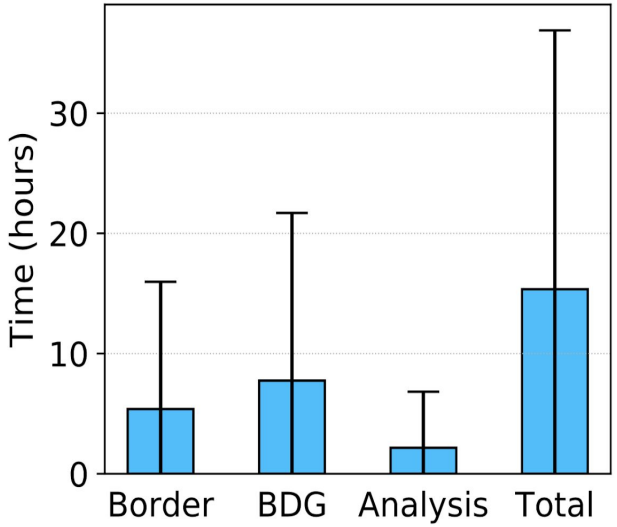


One sample produced a BDG containing **16** binaries!

Execution Time







AAAAND DEMO TIME!

(Do ~~not~~ try this at home)

Andrea Continella
University of Twente

https://github.com/ucsb-seclab/karonte/

ucsb-seclab / karonte

Watch 16 | Star 105 | Fork 20

Code | Issues 0 | Pull requests 0 | Actions | Projects 0 | Security | Insights

Karonte is a static analysis tool to detect multi-binary vulnerabilities in embedded firmware

17 commits | 1 branch | 0 packages | 0 releases | 2 contributors | BSD-2-Clause

Branch: master | New pull request | Find file | Clone or download

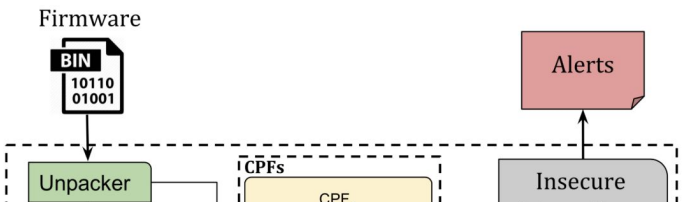
README.md

Karonte

license BSD-2-Clause

Karonte is a static analysis tool to detect multi-binary vulnerabilities in embedded firmware.

Overview



Summary & Takeaways

Firmware is often composed by **multiple interacting binaries**

We introduced static analysis techniques to perform **multi-binary taint analysis**

Karonte can effectively **discover unknown bugs** drastically **reducing** the number of **false positives**

Modelling **multi-binary interactions** can make program analysis **easier**!

https://github.com/ucsb-seclab/karonte

Thanks!

Nilo Redini

nredini@cs.ucsb.edu

https://badnack.it

@badnack

Andrea Continella

acontinella@iseclab.org

https://conand.me

@_conand

KARONTE: Detecting Insecure Multi-binary Interactions in Embedded Firmware

Nilo Redini*, Aravind Machiry*, Ruoyu Wang†, Chad Spensky*, Andrea Continella*, Yan Shoshitaishvili†, Giovanni Vigna*, and Christopher Kruegel*

*UC Santa Barbara

{nredini, machiry, cspensky, conand, vigna, chris}@cs.ucsb.edu

†Arizona State University

{fishw, yans}@asu.edu