

A decorative graphic on the left side of the slide, consisting of a network of thin, light blue lines that resemble a circuit board or a neural network. These lines are connected to small, light blue circles, creating a complex, branching pattern that extends from the top to the bottom of the frame.

FIRMWARE SLAP: AUTOMATING DISCOVERY OF EXPLOITABLE VULNERABILITIES IN FIRMWARE

CHRISTOPHER ROBERTS



REDLattice

WHO AM I

- Researcher at REDLattice Inc.
- Interested in finding bugs in embedded systems
- Interested in program analysis
- CTF Player

The background is black with white line art in the corners representing circuit traces. In the top-left, several lines branch out from the left edge, ending in small circles. In the top-right, a few lines branch out from the right edge, ending in small circles. In the bottom-left, more lines branch out from the left edge, ending in small circles. In the bottom-right, a few lines branch out from the right edge, ending in small circles.

A QUICK BACKGROUND IN EXPLOITABLE BUGS



DARPA CYBER GRAND CHALLENGE

- **Automated** cyber reasoning systems:
  - Find vulnerabilities
  - Exploit vulnerabilities
  - Patch vulnerabilities
- Automatically generates full exploits and proof of concepts.



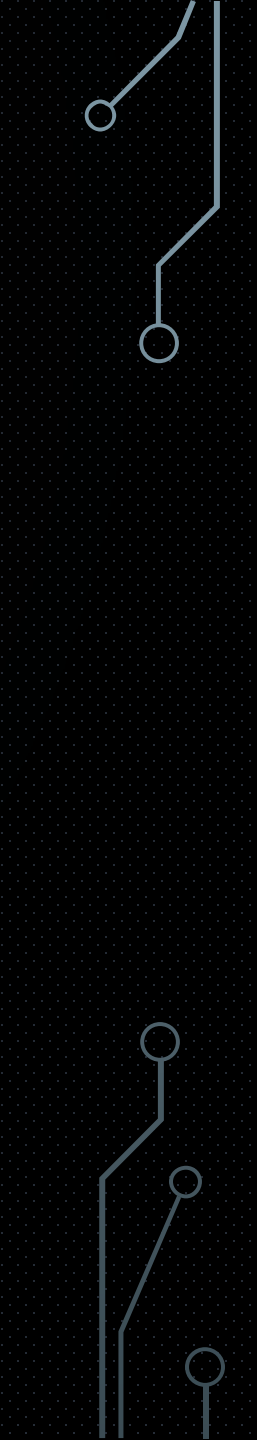
PREVENTING BUGS AUTOMATICALLY



- Source level protections
  - LLVM's Clang static analyzers
- Compile time protections
  - Non-executable stack
  - Stack canaries
  - RELRO
  - `_FORTIFY_SOURCE`
- Operating system protections
  - ASLR



PREVENTING BUGS AUTOMATICALLY In Embedded Devices

- Source level protections
    - LLVM's Clang static analyzers - Maybe
  - Compile time protections
    - Non-executable stack - Maybe
    - ~~Stack canaries~~
    - ~~RELRO~~
    - ~~__FORTIFY_SOURCE~~
  - Operating system protections
    - ~~ASLR~~
- 

1-16 of over 3,000 results for "router"

Amazon Prime



AmazonFresh



Department

Computers & Accessories
Computer Routers
Computer Networking Wireless Access Points
Computer Networking
Whole Home & Mesh Wi-Fi Systems
Tools & Home Improvement
Routers
Router Tables
Straight Router Bits
Industrial & Scientific
Industrial Drill Bits
See All 11 Departments

Avg. Customer Review

★★★★☆ & Up
★★★★☆ & Up
★★★★☆ & Up
★★★★☆ & Up

Brand

☐ NETGEAR
☐ TP-LINK
☐ Linksys
☐ D-Link
☐ ASUS
☐ Belkin
☐ DEWALT
☐ Bosch
☐ Tenda
☐ Routers
☐ Router
☐ Google
☐ Mediabridge

Subscribe & Save

☐ Subscribe & Save Eligible

Connectivity Type

☐ Wireless
☐ Ethernet
☐ USB

Computer Activity Type

☐ Personal
☐ Gaming
☐ Business

Wireless Access Point Transmission Speed

☐ 54 Mbps
☐ 150 Mbps
☐ 300 Mbps
☐ 450 Mbps
☐ 600 Mbps

Router Features

☐ Compact
☐ Plunge
☐ Fixed Base



NETGEAR Nighthawk Smart WiFi Router (R6700) - AC1750 Wireless Speed (up to 1750 Mbps) | Up to 1500 sq ft Coverage & 25 Devices | 4 x 1G Ethernet and 1 x 3.0 USB ports | Armor Security

★★★★☆ ~ 17,353

Prime Day deal

\$68⁴⁹ ~~\$129.99~~



More Buying Choices
\$49.99 (60 used & new offers)

Amazon Certified: Works with Alexa

Amazon's Choice



TP-Link AC1750 Smart WiFi Router - Dual Band Gigabit Wireless Internet Router for Home, Works with Alexa, VPN Server, Parental Control&QoS(Archer A7)

★★★★☆ ~ 5,416

\$64⁹⁹ ~~\$79.99~~

Save 20% with coupon



More Buying Choices
\$51.99 (11 used & new offers)

Amazon Certified: Works with Alexa



TP-Link N450 Wi-Fi Router - Wireless Internet Router for Home, Wireless Access Point Mode (TL-WR940N)

★★★★☆ ~ 19,659

\$24⁹⁸



More Buying Choices



NETGEAR Nighthawk Smart WiFi Router (R7000) - AC1900 Wireless Speed (up to 1900 Mbps) | Up to 1800 sq ft Coverage & 30 Devices | 4 x 1G Ethernet and 2 USB ports | Armor Security

★★★★☆ ~ 11,762

\$153⁰⁰ ~~\$189.99~~

prime FREE Delivery Fri, Jul 19

More Buying Choices
\$71.49 (45 used & new offers)

Amazon Certified: Works with Alexa



NETGEAR WiFi Router (R6230) - AC1200 Dual Band Wireless Speed (up to 1200 Mbps) | Up to 1200 sq ft Coverage & 20 Devices | 4 x 1G Ethernet and 1 x 2.0 USB Ports

★★★★☆ ~ 3,052

Prime Day deal

\$41⁹⁹ ~~\$59.99~~

prime FREE Delivery Fri, Jul 19

More Buying Choices
\$32.69 (14 used & new offers)

EXPLOIT MITIGATIONS

- There has to be an **exploit** to mitigate it, right?

amazon

prime

All

router

Browsing History

Prime Day Deals

NULL's Amazon.com

Buy Again

Gift Cards

1-16 of over 3,000 results for "router"

Amazon Prime

☐ prime

AmazonFresh

☐ fresh

Department

Computers & Accessories

Computer Routers

Computer Networking Wireless Access Points

Computer Networking

Whole Home & Mesh Wi-Fi Systems

Tools & Home Improvement

Routers

Router Tables

Straight Router Bits

Industrial & Scientific

Industrial Drill Bits

See All 11 Departments

Avg. Customer Review

★★★★☆ & Up

★★★★☆ & Up

★★★★☆ & Up

★★★★☆ & Up

Brand

☐ NETGEAR

☐ TP-LINK

☐ Linksys

☐ D-Link

☐ ASUS

☐ Belkin

☐ DEWALT

☐ Bosch

☐ Tenda

☐ Routers

☐ Router

☐ Google

☐ Mediabridge

Subscribe & Save

☐ Subscribe & Save Eligible

Connectivity Type

☐ Wireless

☐ Ethernet

☐ USB

Computer Activity Type

☐ Personal

☐ Gaming

☐ Business

Wireless Access Point Transmission Speed

☐ 54 Mbps

☐ 150 Mbps

☐ 300 Mbps

☐ 450 Mbps

☐ 600 Mbps

Router Features

☐ Compact

☐ Plunge

☐ Fixed Base



Amazon's Choice











ALMOND 3





DEMO

- CVE-2019-13087
- CVE-2019-13088
- CVE-2019-13089
- CVE-2019-13090
- CVE-2019-13091
- CVE-2019-13092

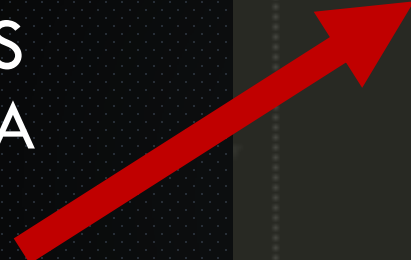


CONCOLIC ANALYSIS

- **Symbolic** Analysis + **Concrete** Analysis
  - Lots of talks already on this subject.
  - Really good at find specific inputs to trigger code paths
  - For my work in Firmware Slap I used **angr**!
    - Concolic analysis
    - CFG analysis
    - Used in Cyber Grand Challenge for 3rd place!

BUILDING REAL INPUTS FROM SYMBOLIC DATA

- **Symbolic Variable** Here
  - `get_user_input()`
  - To get our "You did it" output
    - angr will create several **program states**
    - One has the **constraints**:
      - $x \geq 200$
      - $x < 250$
    - angr sends these **constraints** to it's theorem prover to give:
      - $x=231$ or $x=217$ or $x=249...$



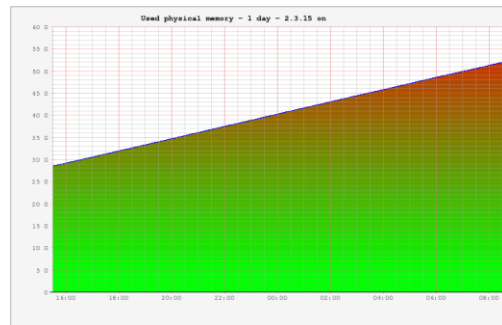
```
int x = get_user_input();
if (x >= 200)
{
    if (x < 250)
    {
        printf("You did it!\n");
    }
    else
    {
        printf("WRONG\n");
    }
}
else
{
    printf("WRONG\n");
}
```

- 
- The slide features a dark background with decorative white circuit-like lines. These lines, consisting of straight segments and small circles at junctions, are positioned along the left and right edges of the slide, framing the central text area.
- **Symbolically** represent more of the program state.
    - Registers, Call Stack, Files
  - Query the analysis for more interesting conditions
    - Does a network read **influence** or **corrupt** the **program counter**?
    - Does network data get fed into sensitive system calls?
  - Can we track all reads and writes required to trigger a **vulnerability**?

```
import angr

project = angr.Project('bin/lighttpd')

project.execute()
```

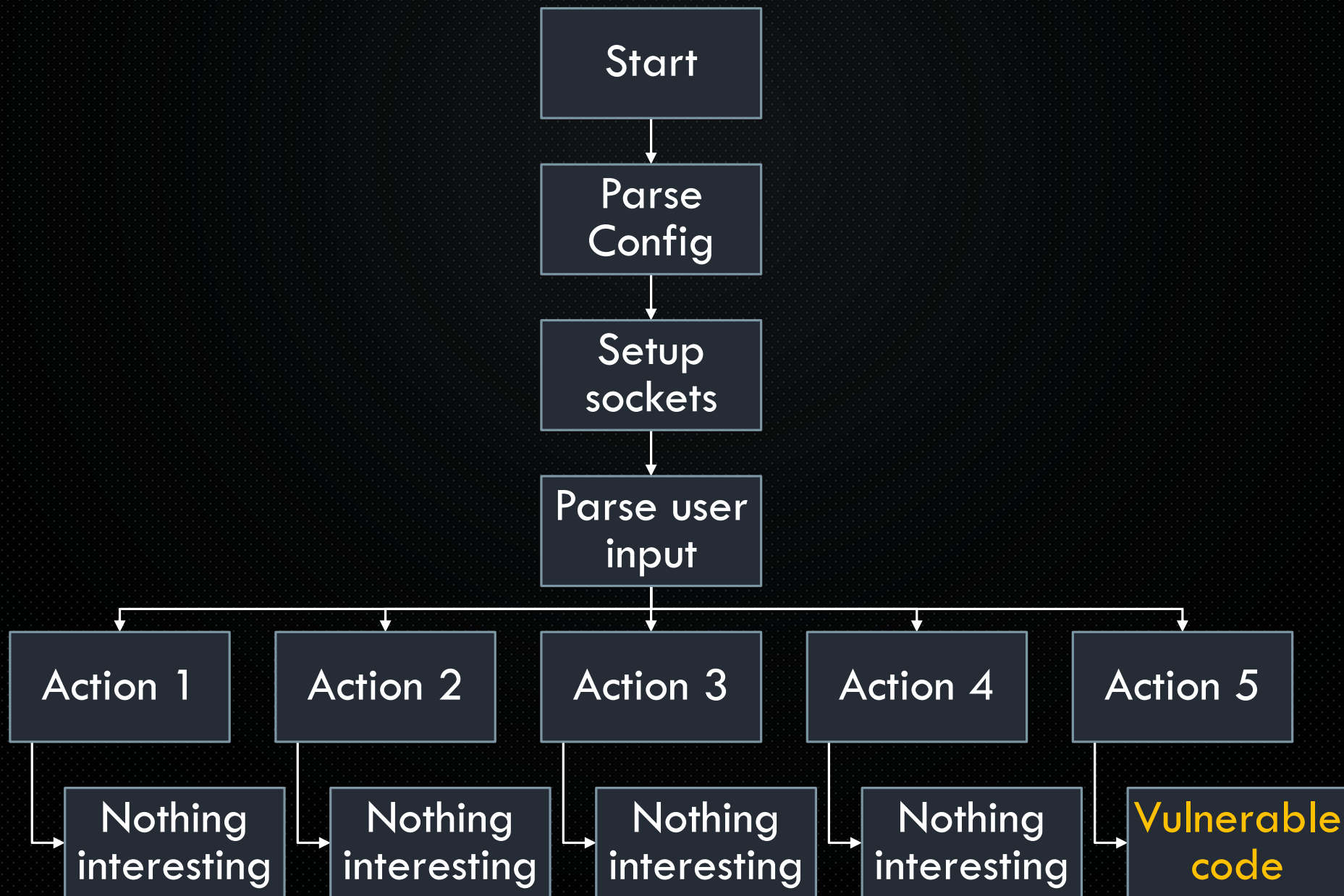


Memory Usage

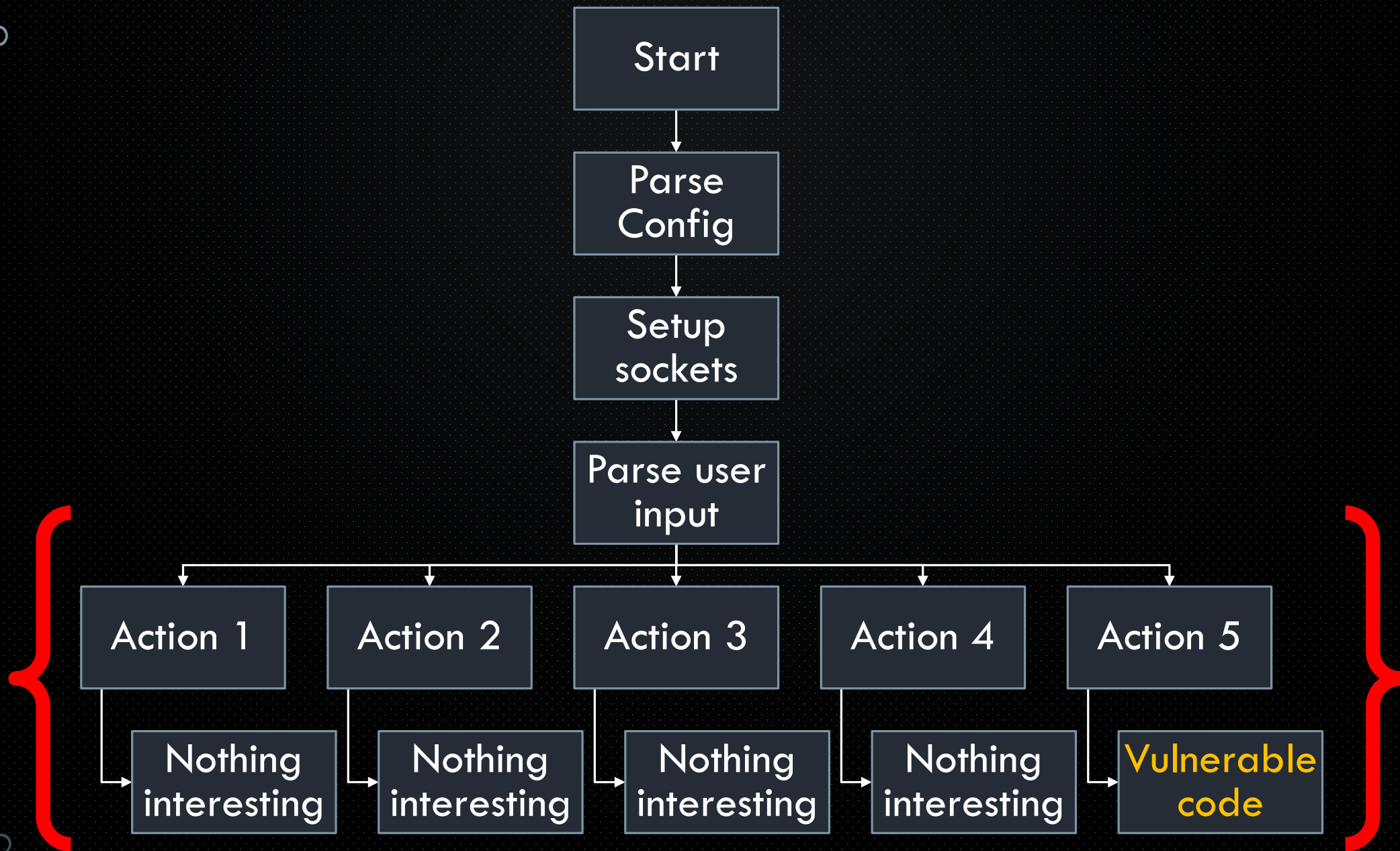
WHERE DOES CONCOLIC ANALYSIS FAIL?

- **Big code bases**
- **Angr** is trying to map out every single potential path through a program. Programs of non-trivial size will eat all your resources.
- A compiled lighttpd binary might be **~200KB**
- Angr will run your computer **out of memory** before it can example every potential program state in a webserver
- Embedded system's firmware can be a lot larger...

- 
- 
- 
- 
- Challenge:
    - Model **complicated binaries** with limited resources
    - Model **unknown** input
    - Identify **vulnerabilities** in binaries
    - Find binaries and functions that **similar** to one-another



- 
- 
- Underconstraining concolic analysis:
    - Values from hardware peripherals and NVRAM are **UNKNOWN**
    - Spin up and initialization consumes valuable **time** and **resources**
    - Configs can be setup any number of ways
  - Skip the hard stuff
    - Make hardware peripherals and NVRAM return **symbolic** variables
    - Start concolic analysis **after** the initialization steps
- 
- 



MODELING FUNCTIONS

- **angr** can analyze code at this level, but it needs to know where to start.
- **Ghidra** can produce a function prototype that **angr** can use to analyze a function...

The Ghidra logo is displayed in a stylized, blocky font. The letters are primarily yellow with a red outline. The background of the logo features a grid of small squares, some of which are filled with red, creating a pixelated or digital effect. The logo is set against a dark background with faint circuit-like patterns.

- Finding bugs in binaries

- **Recover** every function prototype using ghidra
  - Build an angr **program state** with information with symbolic arguments from the prototype
  - Run each analysis job in **parallel**
- 



FINDING BUGS IN FUNCTIONS

- Demo
- 
- 

- 
- 
- With less code to analyze we can introduce more **heavy-weight** analysis
    - **Tracking memory** instructions imposed by all instructions
    - **Memory regions tainted** by user supplied arguments
    - Mapping **memory loading actions** to values in memory.
  - Every step through a program
    - Store **any new constraints** to user input
    - Does user input influence a **system()** call or **corrupt the program counter**
    - Does user input **taint** a stack or heap variable
- 
- 

```
Memory load value b'\x00\x00\x00\x00reboot'
```

```
void remove_routing_rule(char *iptable_arg, char *netmask_arg, char *inet_device_arg)
{
    int iVar1;
    size_t buf_len;
    size_t sVar2;
    size_t buf_len2;
    char *user_buf;
    char acStack276 [256];

    strcpy(&user_buf, "route del ", 0x100);
    iVar1 = strcmp(netmask_arg, "255.255.255.255");
    if (iVar1 == 0) {
        buf_len = strlen((char *)&user_buf);
        /* -host */
        *(undefined4 *)((int)&user_buf + buf_len) = 0x736f682d;
        acStack276[buf_len] = 't';
        acStack276[buf_len + 1] = ' ';
        acStack276[buf_len + 2] = 0;
    }
    else {
        buf_len2 = strlen();
        /* -net */
        *(undefined4 *)((int)&user_buf + buf_len2) = 0x74656e2d;
        acStack276[buf_len2] = ' ';
        acStack276[buf_len2 + 1] = 0;
    }
    strcat((char *)&user_buf, iptable_arg);
    sVar2 = strlen((char *)&user_buf);
    *(undefined *)((int)&user_buf + sVar2) = 0x20;
    *(undefined *)((int)&user_buf + sVar2 + 1) = 0;
    iVar1 = strcmp(netmask_arg, "255.255.255.255");
    if (iVar1 != 0) {
        snprintf((char *)&user_buf, 0x100, "%s netmask %s", &user_buf, netmask_arg);
    }
    snprintf((char *)&user_buf, 0x100, "%s dev %s ", &user_buf, inet_device_arg);
    system((char *)&user_buf);
    return;
}
```

Command Injection found in internet.cgi at remove_routing_rule

```
void remove_routing_rule(undefined4 uParm1,undefined4 uParm2,undefined4 uParm3);
```

```
a0:4 : 0xffffe000 -> b'`reboot` @@@@ @
```

```
a1:4 : 0xffffffffc0 -> b'255.255.255.255'
```

```
a2:4 : 0xfffffffffc -> b'@ @ @ @ @ @ @ @
```

(Injected Memory Location

```
0x7ffefed8 -> b'route del -host `reboot` @@@@ @ dev @ @ @ @ @
```

Tainted memory values

remove_routing_rule+0x7c in internet.cgi (0x4019d8)

Memory load addr 0xffffe000

Memory load value b'`reboot` @@@@ @

remove_routing_rule+0xc4 in internet.cgi (0x401a20)

Memory load addr 0x7ffefed8

Memory load value b'route del -host `reboot` @@@@ @

remove_routing_rule+0x140 in internet.cgi (0x401a9c)

Memory load addr 0x7ffefed8

Memory load value b'route del -host `reboot` @@@@ @

remove_routing_rule+0x160 in internet.cgi (0x401abc)

Memory load addr 0x7ffefee8

Memory load value b'`reboot`

```
void remove_routing_rule(char *iptable_arg,char *netmask_arg,char *inet_device_arg)
{
    int iVar1;
    size_t buf_len;
    size_t sVar2;
    size_t buf_len2;
    char *user_buf;
    char acStack276 [256];

    strcpy(&user_buf,"route del ",0x100);
    iVar1 = strcmp(netmask_arg,"255.255.255.255");
    if (iVar1 == 0) {
        buf_len = strlen((char *)&user_buf);
        /* -host */
        *(undefined4 *)((int)&user_buf + buf_len) = 0x736f682d;
        acStack276[buf_len] = 't';
        acStack276[buf_len + 1] = ' ';
        acStack276[buf_len + 2] = 0;
    }
    else {
        buf_len2 = strlen();
        /* -net */
        *(undefined4 *)((int)&user_buf + buf_len2) = 0x74656e2d;
        acStack276[buf_len2] = ' ';
        acStack276[buf_len2 + 1] = 0;
    }
    strcat((char *)&user_buf,iptable_arg);
    sVar2 = strlen((char *)&user_buf);
    *(undefined *)((int)&user_buf + sVar2) = 0x20;
    *(undefined *)((int)&user_buf + sVar2 + 1) = 0;
    iVar1 = strcmp(netmask_arg,"255.255.255.255");
    if (iVar1 != 0) {
        sprintf((char *)&user_buf,0x100,"%s netmask %s",&user_buf,netmask_arg);
    }
    sprintf((char *)&user_buf,0x100,"%s dev %s ",&user_buf,inet_device_arg);
    system((char *)&user_buf);
    return;
}
```

Command Injection found in internet.cgi at remove_routing_rule

```
void remove_routing_rule(undefined4 uParm1,undefined4 uParm2,undefined4 uParm3);
```

```
  a0:4 : 0xffffe000 -> b'`reboot` @@@@ @'
```

```
  a1:4 : 0xffffffffc0 -> b'255.255.255.255'
```

```
  a2:4 : 0xfffffffffc -> b'@ @ @ @ @ @ @ @'
```

(Injected Memory Location

```
  0x7ffefed8 -> b'route del -host `reboot` @@@@ @ dev @ @ @ @ @
```

Tainted memory values

remove_routing_rule+0x7c in internet.cgi (0x4019d8)

Memory load addr 0xffffe000

Memory load value b'`reboot` @@@@ @'

remove_routing_rule+0xc4 in internet.cgi (0x401a20)

Memory load addr 0x7ffefed8

Memory load value b'route del -host `reboot` @@@@ @'

remove_routing_rule+0x140 in internet.cgi (0x401a9c)

Memory load addr 0x7ffefed8

Memory load value b'route del -host `reboot` @@@@ @'

remove_routing_rule+0x160 in internet.cgi (0x401abc)

Memory load addr 0x7ffefee8

Memory load value b'`reboot`'

```
void remove_routing_rule(char *iptable_arg,char *netmask_arg,char *inet_device_arg)
{
    int iVar1;
    size_t buf_len;
    size_t sVar2;
    size_t buf_len2;
    char *user_buf;
    char acStack276 [256];

    strcpy(&user_buf,"route del ",0x100);
    iVar1 = strcmp(netmask_arg,"255.255.255.255");
    if (iVar1 == 0) {
        buf_len = strlen((char *)&user_buf);
        /* -host */
        *(undefined4 *)((int)&user_buf + buf_len) = 0x736f682d;
        acStack276[buf_len] = 't';
        acStack276[buf_len + 1] = ' ';
        acStack276[buf_len + 2] = 0;
    }
    else {
        buf_len2 = strlen();
        /* -net */
        *(undefined4 *)((int)&user_buf + buf_len2) = 0x74656e2d;
        acStack276[buf_len2] = ' ';
        acStack276[buf_len2 + 1] = 0;
    }
    strcat((char *)&user_buf,iptable_arg);
    sVar2 = strlen((char *)&user_buf);
    *(undefined4 *)((int)&user_buf + sVar2) = 0x20;
    *(undefined4 *)((int)&user_buf + sVar2 + 1) = 0;
    iVar1 = strcmp(netmask_arg,"255.255.255.255");
    if (iVar1 != 0) {
        sprintf((char *)&user_buf,0x100,"%s netmask %s",&user_buf,netmask_arg);
    }
    sprintf((char *)&user_buf,0x100,"%s dev %s",&user_buf,inet_device_arg);
    system((char *)&user_buf);
    return;
}
```

```
void remove routing rule(undefined4 uParm1,undefined4 uParm2,undefined4 uParm3);
```

```
a1:4 : 0xffffffffc0 -> b'255.255.255.255'
```

Injected Memory Location

Tainted memory values

```
Memory load addr 0xffffe000
```

```
remove routing rule+0xc4 in internet.cgi (0x401a20)
```

```
Memory load addr 0x7ffefed8
```

```
Memory load value b'route del -host `reboot` @@@@ @@ '
```

```
remove routing rule+0x140 in internet.cgi (0x401a9c)
```

Memory load addr 0x7ffefed8

```
Memory load value b'route del -host `reboot` @@@@@ @'  '
```

```
remove routing rule+0x160 in internet.cgi (0x401abc)
```

```
Memory load addr 0x7ffefee8
```

```
Memory load value b'\x00reboot\x00'
```

```
void remove_routing_rule(char *iptable_arg, char *netmask_arg, char *inet_device_arg)
{
    int iVar1;
    size_t buf_len;
    size_t sVar2;
    size_t buf_len2;
    char *user_buf;
    char acStack276 [256];

    strcpy(&user_buf, "route del ", 0x100);
    iVar1 = strcmp(netmask_arg, "255.255.255.255");
    if (iVar1 == 0) {
        buf_len = strlen((char *)&user_buf);
        /* -host */
        *(undefined4 *)((int)&user_buf + buf_len) = 0x736f682d;
        acStack276[buf_len] = 't';
        acStack276[buf_len + 1] = ' ';
        acStack276[buf_len + 2] = 0;
    }
    else {
        buf_len2 = strlen();
        /* -net */
        *(undefined4 *)((int)&user_buf + buf_len2) = 0x74656e2d;
        acStack276[buf_len2] = ' ';
        acStack276[buf_len2 + 1] = 0;
    }
    strcat((char *)&user_buf, iptable_arg);
    sVar2 = strlen((char *)&user_buf);
    *(undefined *)((int)&user_buf + sVar2) = 0x20;
    *(undefined *)((int)&user_buf + sVar2 + 1) = 0;
    iVar1 = strcmp(netmask_arg, "255.255.255.255");
    if (iVar1 != 0) {
        snprintf((char *)&user_buf, 0x100, "%s netmask %s", &user_buf, netmask_arg);
    }
    snprintf((char *)&user_buf, 0x100, "%s dev %s ", &user_buf, inet_device_arg);
    system((char *)&user_buf);
    return;
}
```

```
void remove routing rule(undefined4 uParm1,undefined4 uParm2,undefined4 uParm3);
```

```
a1:4 : 0xffffffffc0 -> b'255.255.255.255'
```

Injected Memory Location

```
0x7ffefed8 -> b'route del -host `reboot` [REDACTED] day @ @ @ @@ @
```

```
void remove_routing_rule(char *iptable_arg, char *netmask_arg, char *inet_device_arg)
{
    int iVar1;
    size_t buf_len;
    size_t sVar2;
    size_t buf_len2;
    char *user_buf;
    char acStack276 [256];

```

```
strcpy(&user_buf,"route del ",0x100);  
iVar1 = strcmp(netmask_arg,"255.255.255.255");  
if (iVar1 == 0) {  
    buf_len = strlen((char *)&user_buf);  
    /* -host */  
    *(int *)(&user_buf + buf_len) = 0x736f682d;
```

Injected Memory Location

0x7ffefed8 -> b'route del -host `reboot`'

```
remove routing rule+0xc4 in Internet.cgi (0x401a20)
```

Memory load addr 0x7ffefed8

```
Memory load value b'route del -host `reboot` @@@@ @' '
```

```
remove routing rule+0x140 in internet.cgi (0x401a9c)
```

```
Memory load addr 0x7ffefed8
```

```
Memory load value b'route del -host `reboot` @@@@ @@ '
```

```
remove routing rule+0x160 in internet.cgi (0x401abc)
```

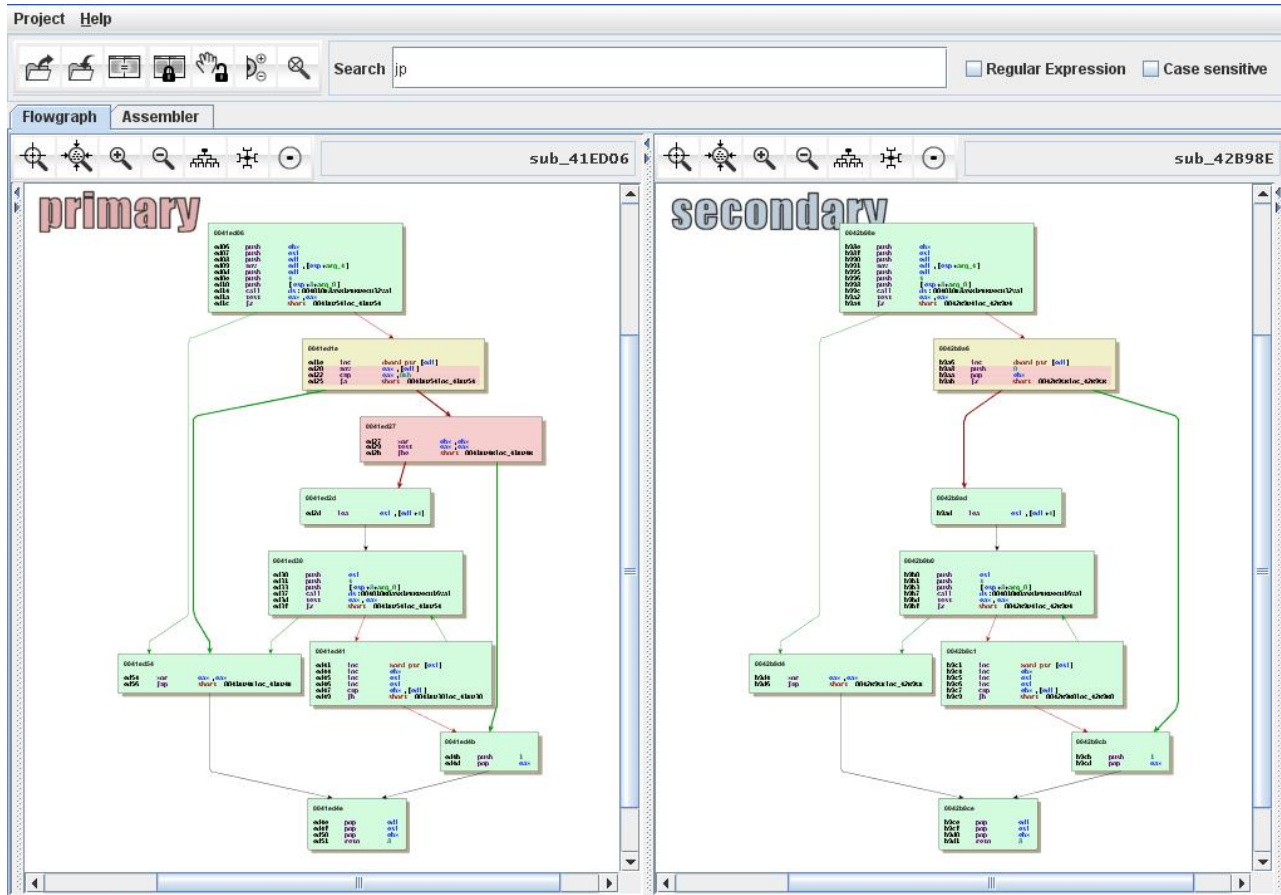
```
Memory load addr 0x7ffefee8
```

```
Memory load value b'`reboot`'
```

```

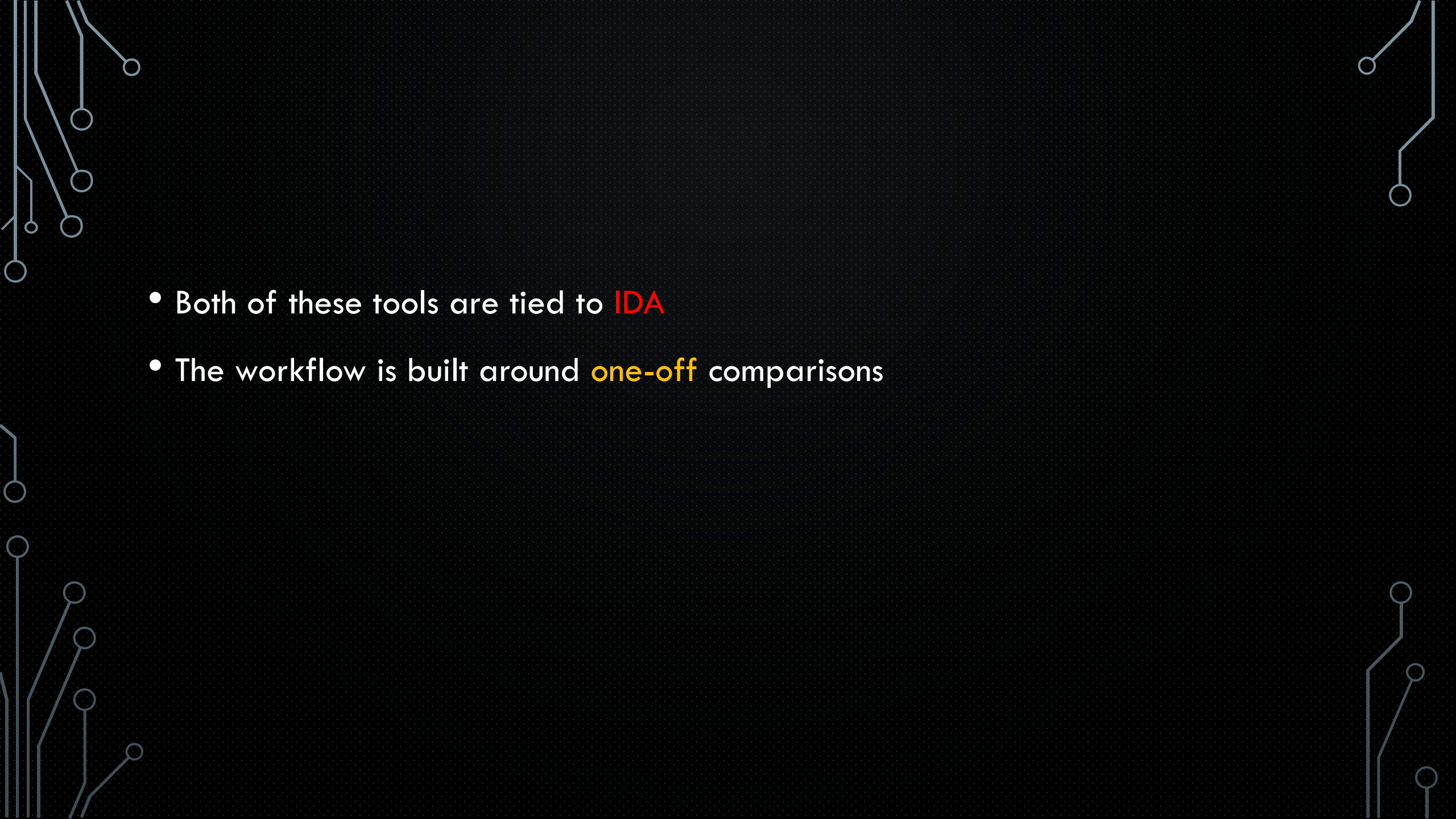
*(undefined4 *)((int)&user_buf + buf_len2) = 0x746556e2d;
acStack276[buf_len2] = ' ';
acStack276[buf_len2 + 1] = 0;
}
strcat((char *)&user_buf, iptable_arg);
sVar2 = strlen((char *)&user_buf);
*(undefined *)((int)&user_buf + sVar2) = 0x20;
*(undefined *)((int)&user_buf + sVar2 + 1) = 0;
iVar1 = strcmp(netmask_arg, "255.255.255.255");
if (iVar1 != 0) {
    snprintf((char *)&user_buf, 0x100, "%s netmask %s", &user_buf, netmask_arg);
}
snprintf((char *)&user_buf, 0x100, "%s dev %s ", &user_buf, inet_device_arg);
system((char *)&user_buf);
return;
}

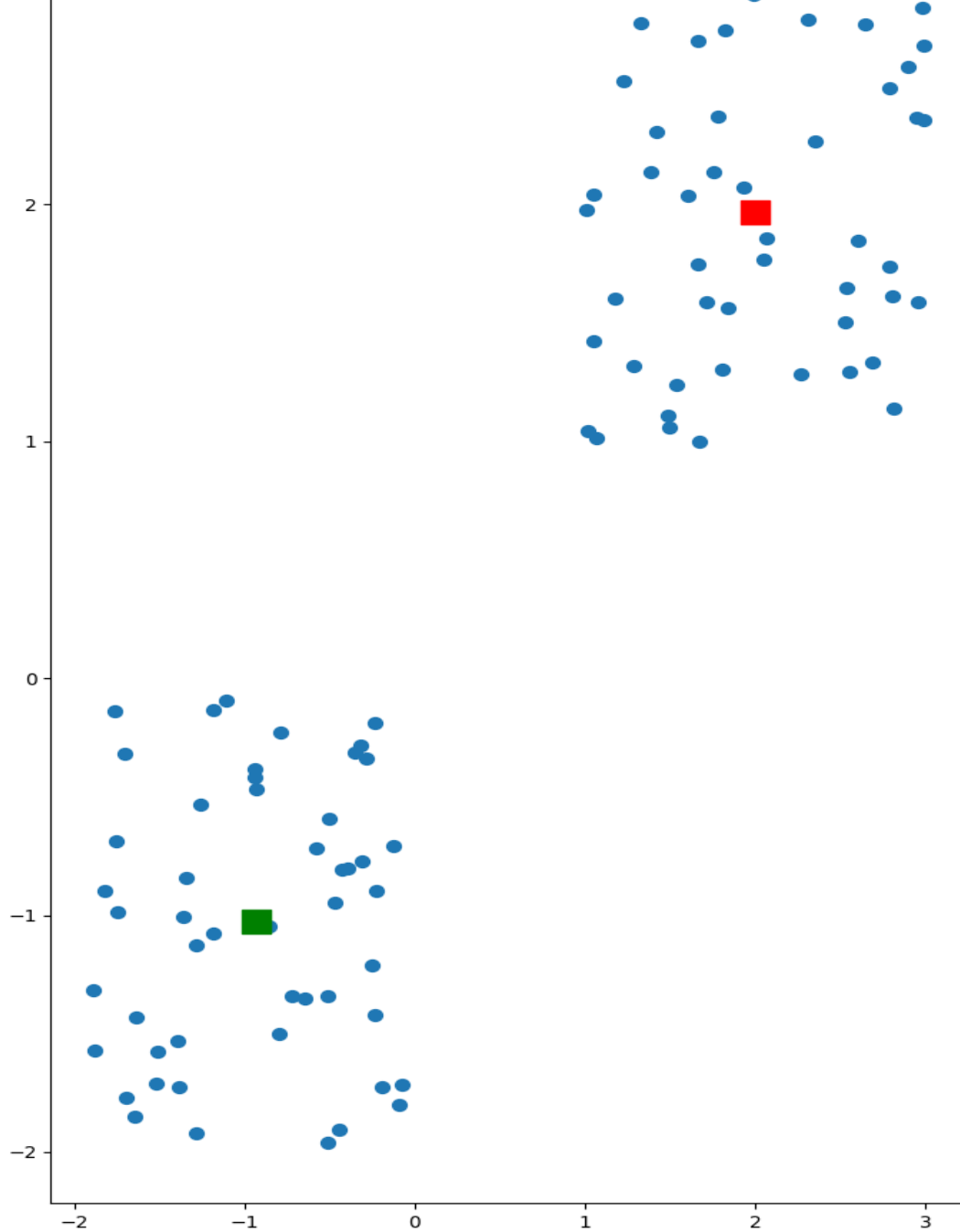
```



FUNCTION SIMILARITY

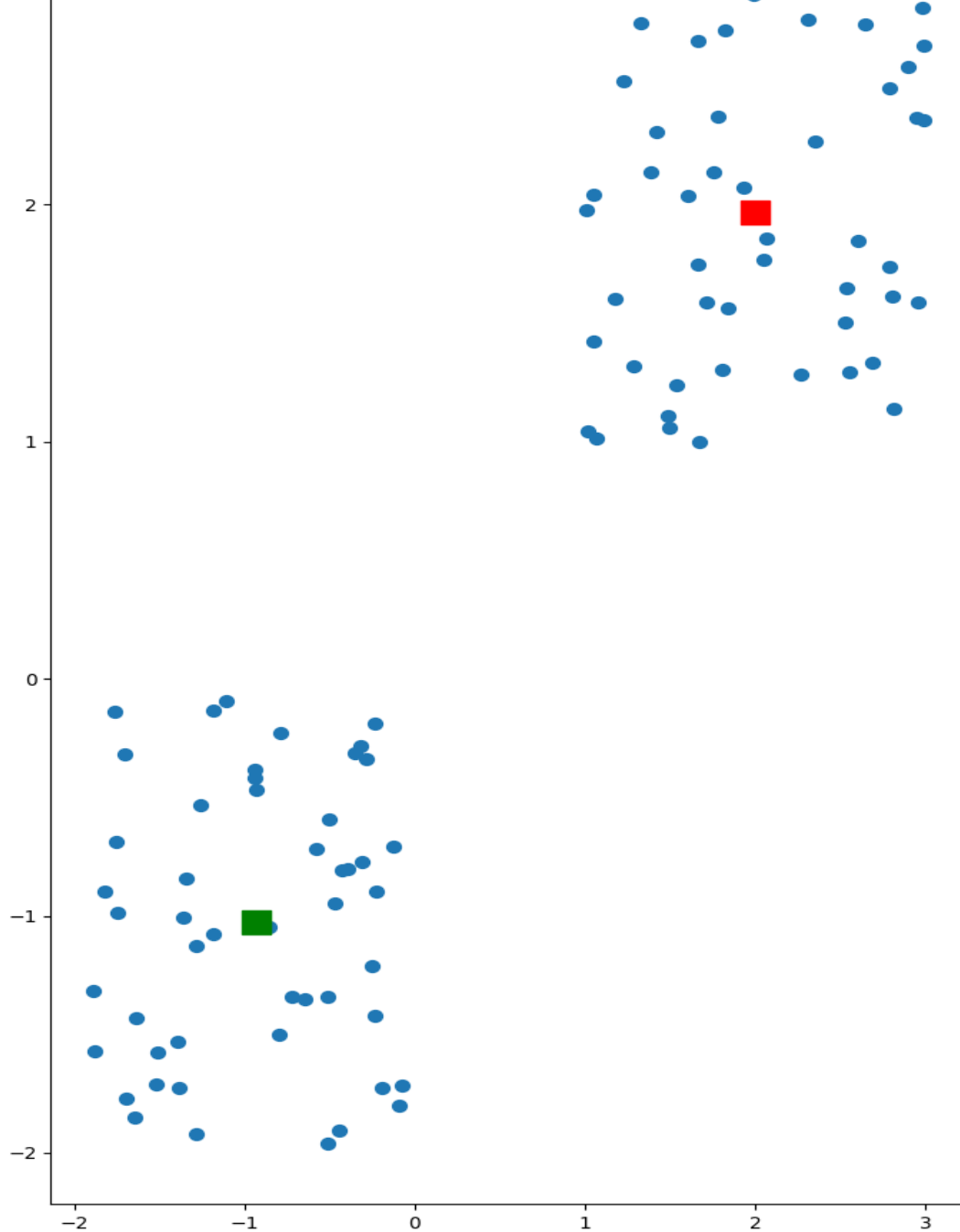
- **Bindiff** and **diaphora** are the standard for binary diffing.
- They help us find what code was actually patched when a CVE and a patch is published.
- Uses a set of heuristics to build a signature for every function in a binary
  - Basic block count
  - Basic block edges
  - Function references

- 
- The background is dark grey with white circuit-like lines in the corners. These lines consist of straight segments and small circles, resembling a stylized PCB or network diagram. They are located in the top-left, top-right, bottom-left, and bottom-right corners.
- Both of these tools are tied to **IDA**
  - The workflow is built around **one-off** comparisons



CLUSTERING

- Helps us understand how **similar** are two things?
- Extract **features** from each thing
  - For dots on a grid it can be:
    - X location
    - Y location



K-MEANS CLUSTERING

Extract features

Pick two random points

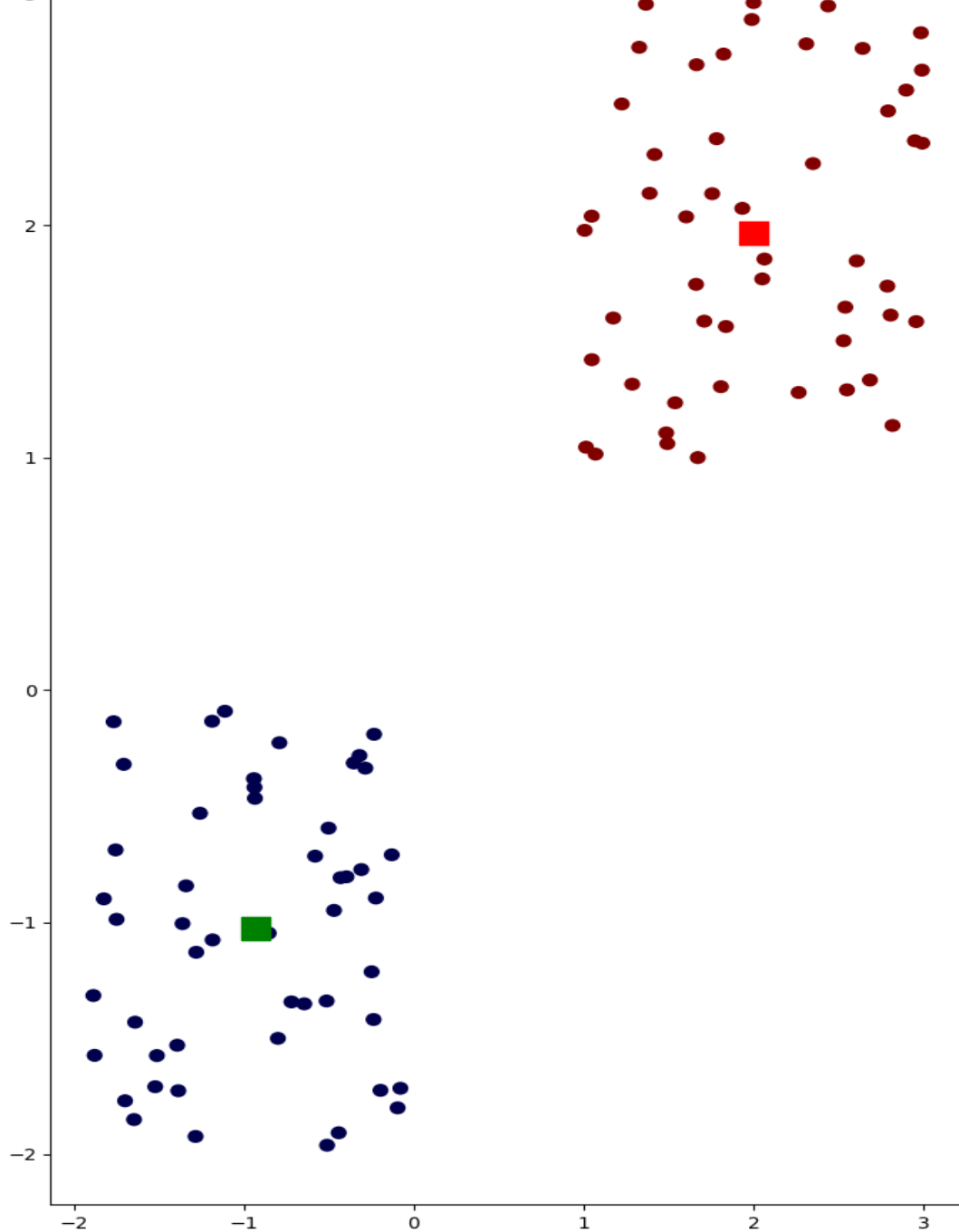
Categorize each point to one of those random points

- Use Euclidian or cosine distance to find which is closest

Pick new cluster center by averaging each category by feature and using the point closest.

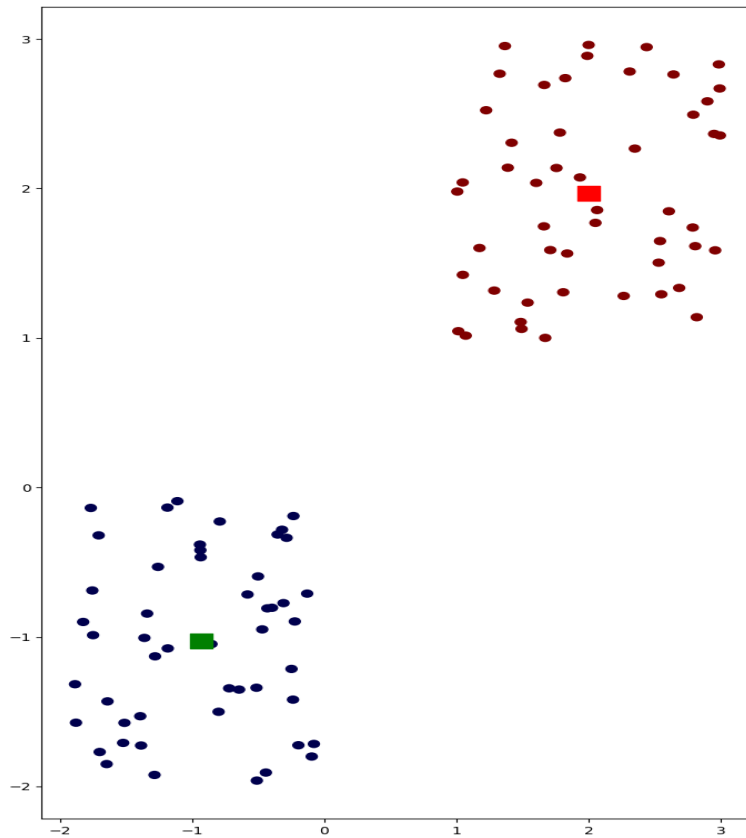
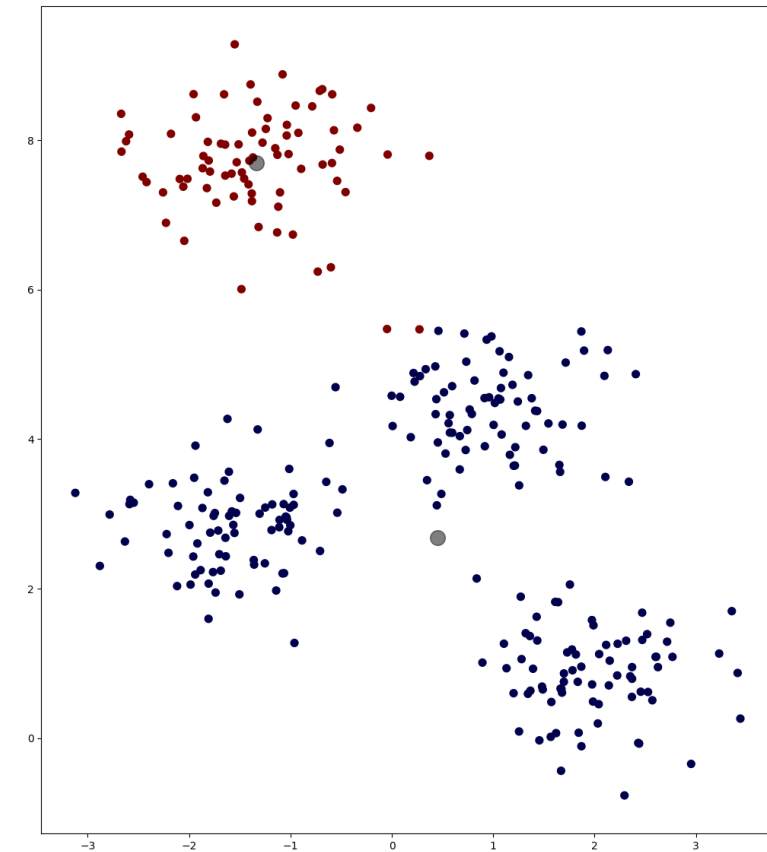
Recategorize all the points into categories.

- Rinse and repeat until points don't move!



CLUSTERING – WHY THIS WORKS

- **Features** don't have to be numbers...
- They can be the **existence** (0 or 1) of:
  - String references
  - Data references
  - Function arguments
  - Basic block count
- All of these features can be extracted from reverse engineering tools like...
- **Ghidra**, **Radare2**, or **Binary Ninja**



IT ONLY WORKS IF YOU GUESS THE RIGHT NUMBER OF CLUSTERS

SUPERVISED CLUSTERING

- Supervised anything machine learning uses **KNOWN** values to cluster data
- We also know how many clusters there **should** be
- Our functions inside our binaries could be supervised if every function was known to be **vulnerable** or **benign**
- Embedded systems programming gives us no assurances.

SEMI-SUPERVISED CLUSTERING

- Semi-Supervised clustering uses **SOME KNOWN** values to cluster data
- If we use public CVE information to find which functions in a binary are **KNOWN** vulnerable, we can guess that really **similar** functions might also be **vulnerable**.
- We can set our **cluster count** to the number of **known vulnerable functions** in a binary

```

'allFixup': None,
'CallingConvention': 'None',
'CallingConventionName': 'unknown',
'CallingFunctions': [{'Addr': '00400de0', 'Name': 'main'}],
'DefaultCallingConventionName': '__stdcall',
'EntryPoint': {'offset': '4207200', 'physicalAddress': '00403260'},
'ExternalLocation': None,
'HiFuncProto': 'void set_vpnpass(undefined4 uParm1);',
'HiParameterCount': 1,
'HiParameters': [{'dataType': 'undefined4',
  'name': 'uParm1',
  'size': '4',
  'slot': '0',
  'storage': 'a0:4'}],
'HiRetType': 'void',
'Key': 208,
'LocalVariables': [{'dataType': 'undefined4',
  'firstUseOffset': '0',
  'length': '4',
  'name': 'local_4',
  'register': 'None',
  'registerVariable': 'False',
  'registers': 'None',
  'source': 'DEFAULT',
  'stackOffset': '-4',
  'stackVariable': 'True',
  'uniqueVariable': 'False',
  'valid': 'True',
  'variableStorage': 'Stack[-0x4]:4'},
{'dataType': 'undefined4',
  'firstUseOffset': '0',
  'length': '4',
  'name': 'local_8',
  'register': 'None',
  'registerVariable': 'False',
  'registers': 'None',
  'source': 'DEFAULT',
  'stackOffset': '-8',
  'stackVariable': 'True',
  'uniqueVariable': 'False',
  'valid': 'True',
  'variableStorage': 'Stack[-0x8]:4'},
{'dataType': 'undefined4',
  'firstUseOffset': '0',
  'length': '4',
  'name': 'local_10',
  'register': 'None',
  'registerVariable': 'False',
  'registers': 'None',
  'source': 'DEFAULT',
  'stackOffset': '-16',
  'stackVariable': 'True',
  'uniqueVariable': 'False',

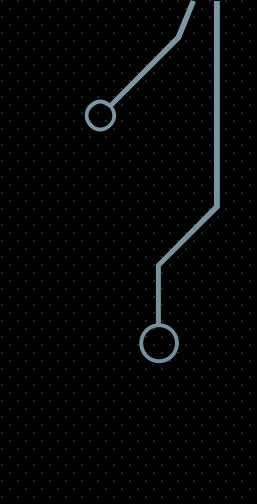
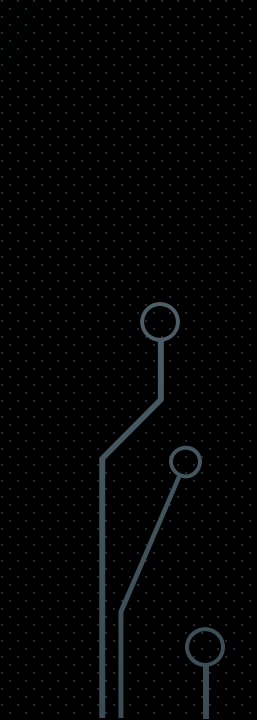
```

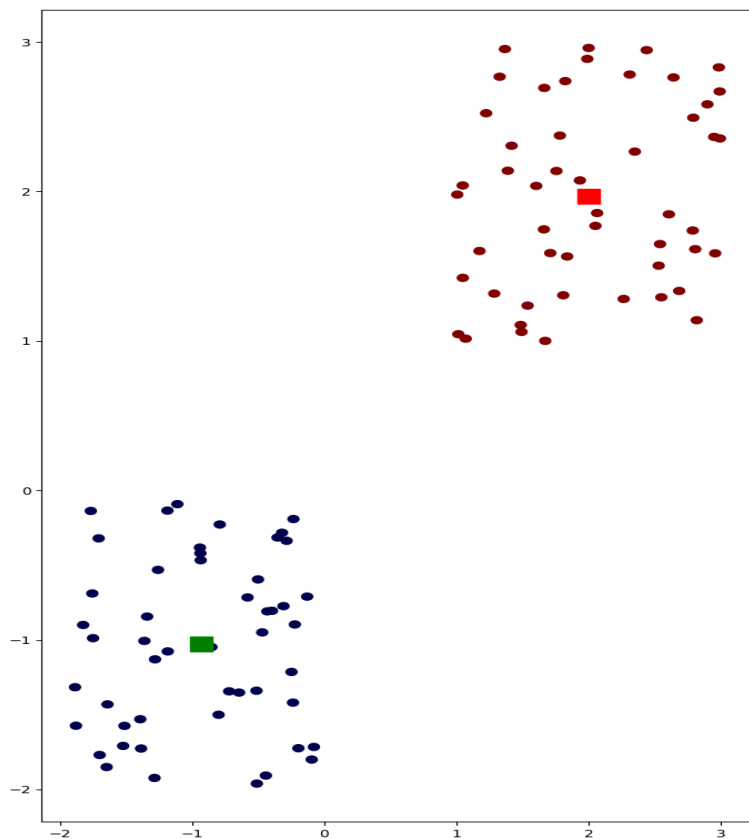
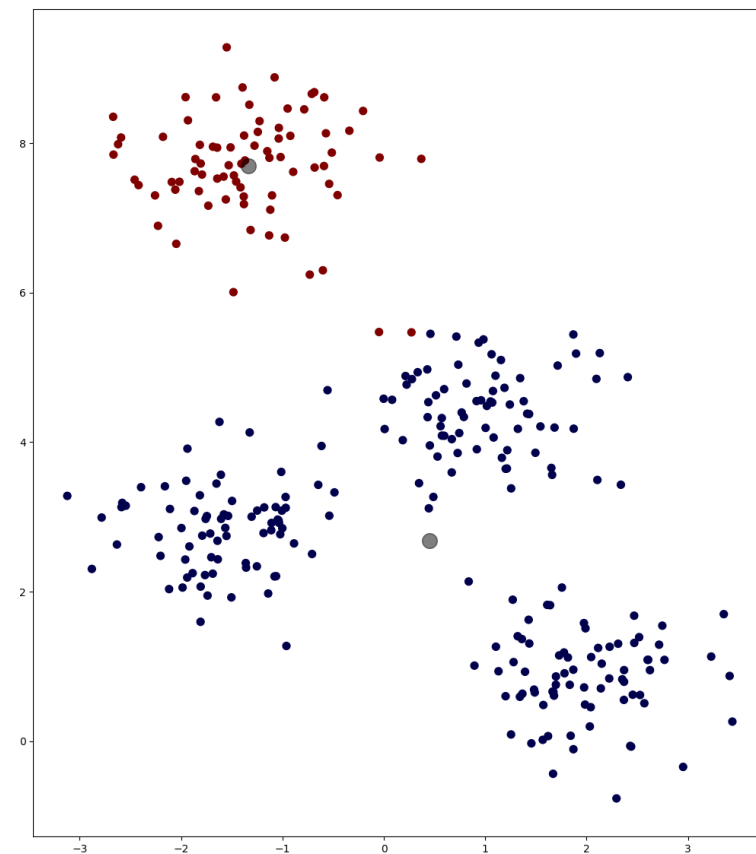
- Finding **features** in binaries to cluster
  - Wrote a **Ghidra** headless plugin to dump all function information
  - Data/String/Call **references** are changed to binary (0/1) it exists or it doesn't
  - All numbers are **normalized**
    - Being at offset **0x80000000** shouldn't matter more then having **2 function arguments**.
  - Throw away useless information
    - A **Chi² squared test** is used to see how much a feature defines an item.
    - If every function has the **same** calling convention, the Chi² squared test will **throw it away**.

- Taking it further...

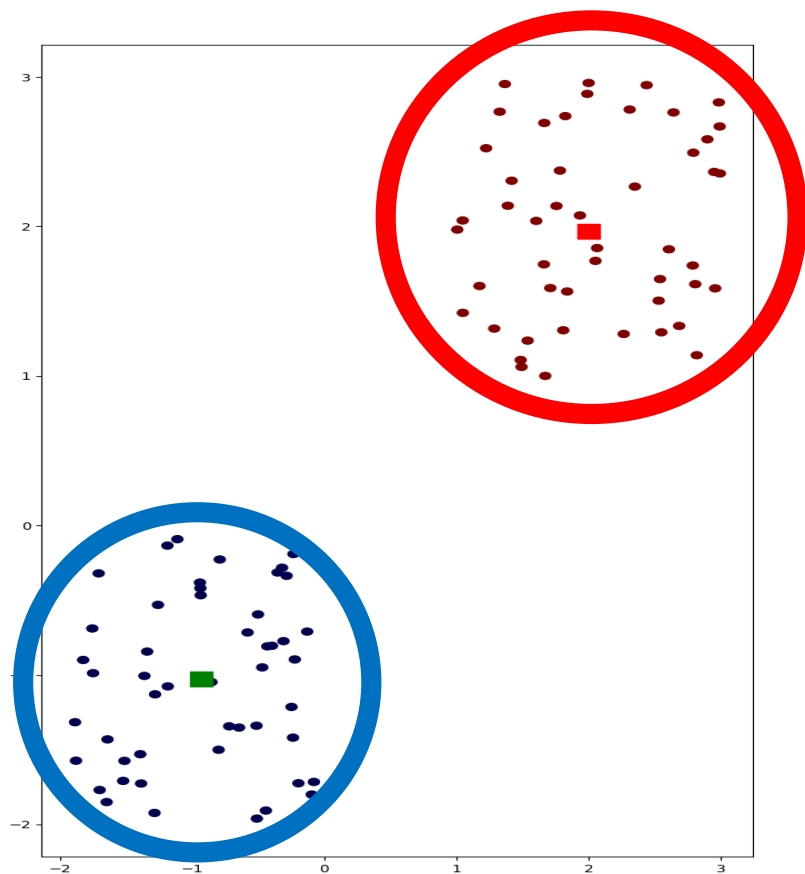
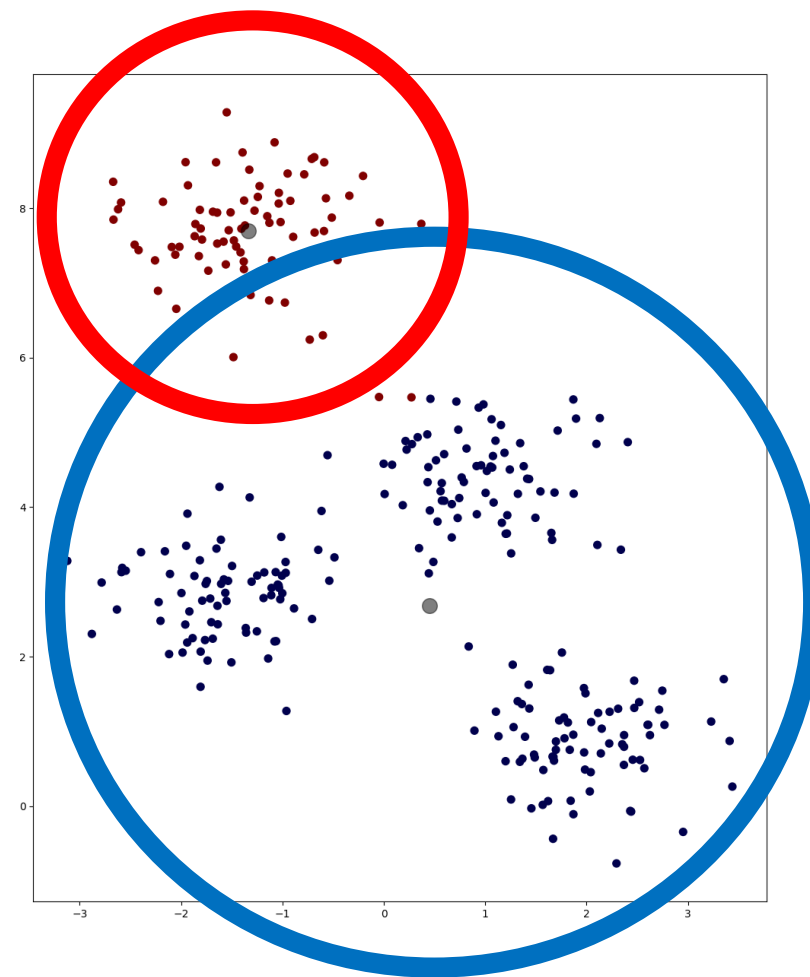
- Selecting a better number of clusters through **cluster scoring**
- **Silhouette score** ranks how similar each cluster of functions are

- This separates functions into clusters of similar tasks

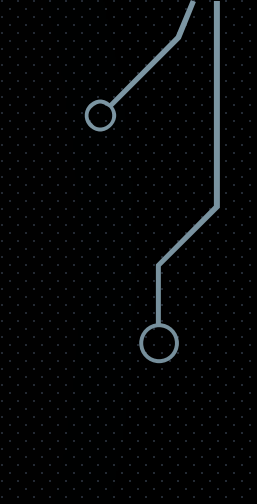
- String operation functions
  - Destructors/Constructors
  - File manipulation
  - Web request handling
  - etc..
- 
- 
- 



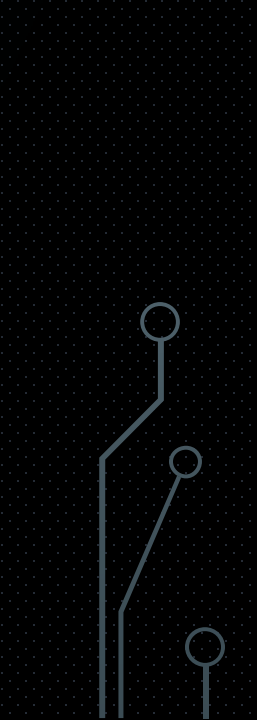
SILHOUETTE SCORE



SILHOUETTE SCORE



DATA MINING + CONCOLIC ANALYSIS

- Demo
  - CVE-2019-13087
- 
- 

Similar Functions With Distances

```
sym.mtd_write_firmware      in upload_bootloader.cgi
> 0.0
sym.mtd_write_firmware      in upload.cgi
> 1.868202467580904e-05
sym.mtd_write_bootloader    in upload_bootloader.cgi
> 0.00015272111900810348
sym.mtd_write_bootloader    in upload.cgi
> 0.00016208252348248742
sym.write_flash_kernel_version in upload.cgi
> 0.0013812828298550572
sym.write_flash_kernel_version in upload_bootloader.cgi
> 0.001381969125964222
sym.access_policy_handle    in wireless.cgi
> 0.0030522430530255384
sym.compile_regex           in wireless.cgi
> 0.003158924286679299
```

Findings

- Code clones
- Calling patterns
- Similar function calls
- Similar data references
- Similar file access

```

int mtd_write_firmware(int param_1,int param_2,char *param_3)
{
    uint uVar1;
    char sys_buffer [516];

    snprintf(sys_buffer,0x200,
        "/bin/mtd_write -o %d -l %d write %s Kernel",param_2,
    uVar1 = system(sys_buffer);
    return uVar1;
}

```

```

void mtd_write_bootloader(int param_1,int param_2,char *param_3)
{
    char sys_buf [516];

    snprintf(sys_buf,0x200,
        "/bin/mtd_write -o %d -l %d write %s Bootloader",param_2,
    printf("write bootloader");
    system(sys_buf);
    return;
}

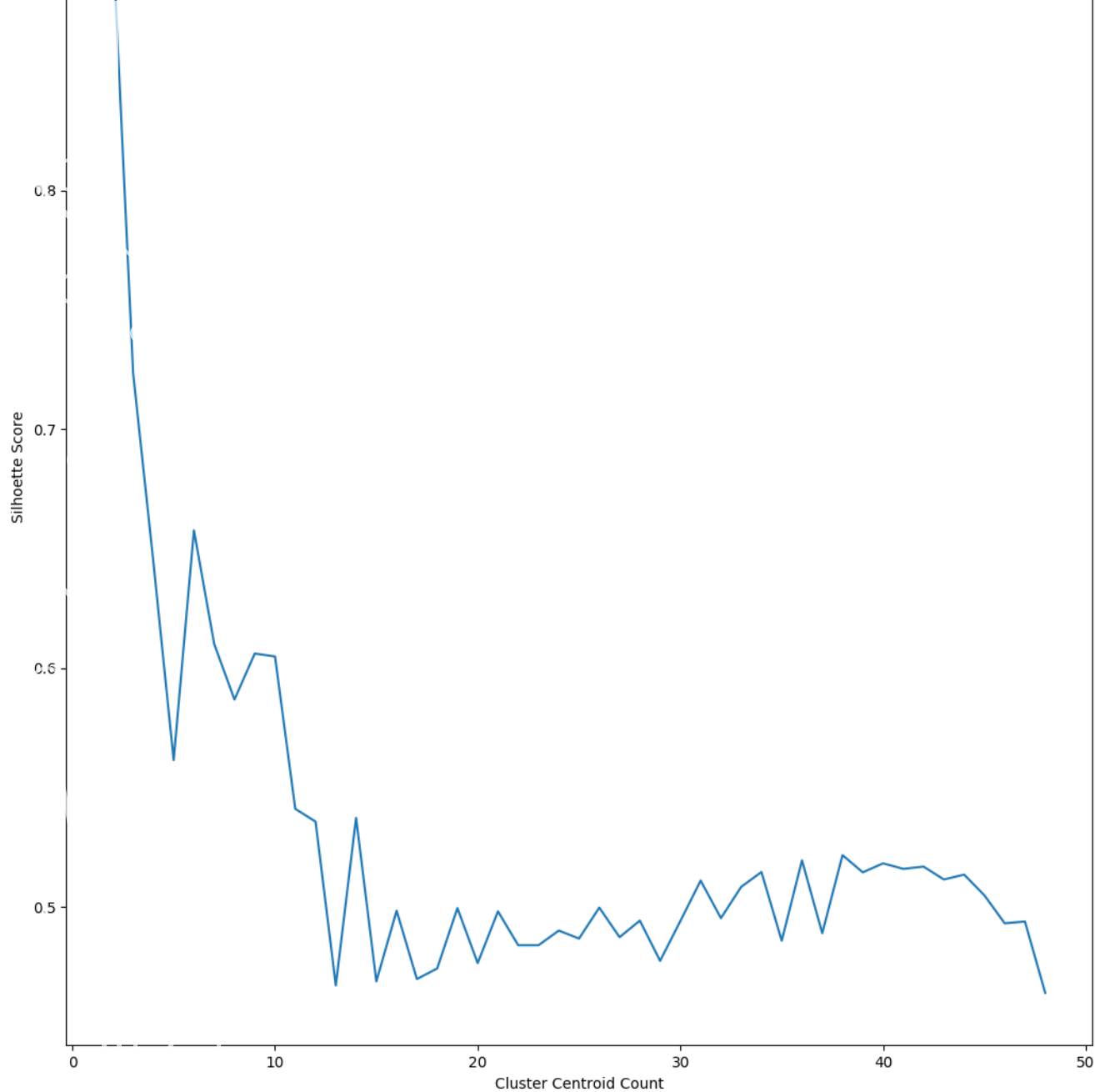
```

Similar Functions With Distances

```

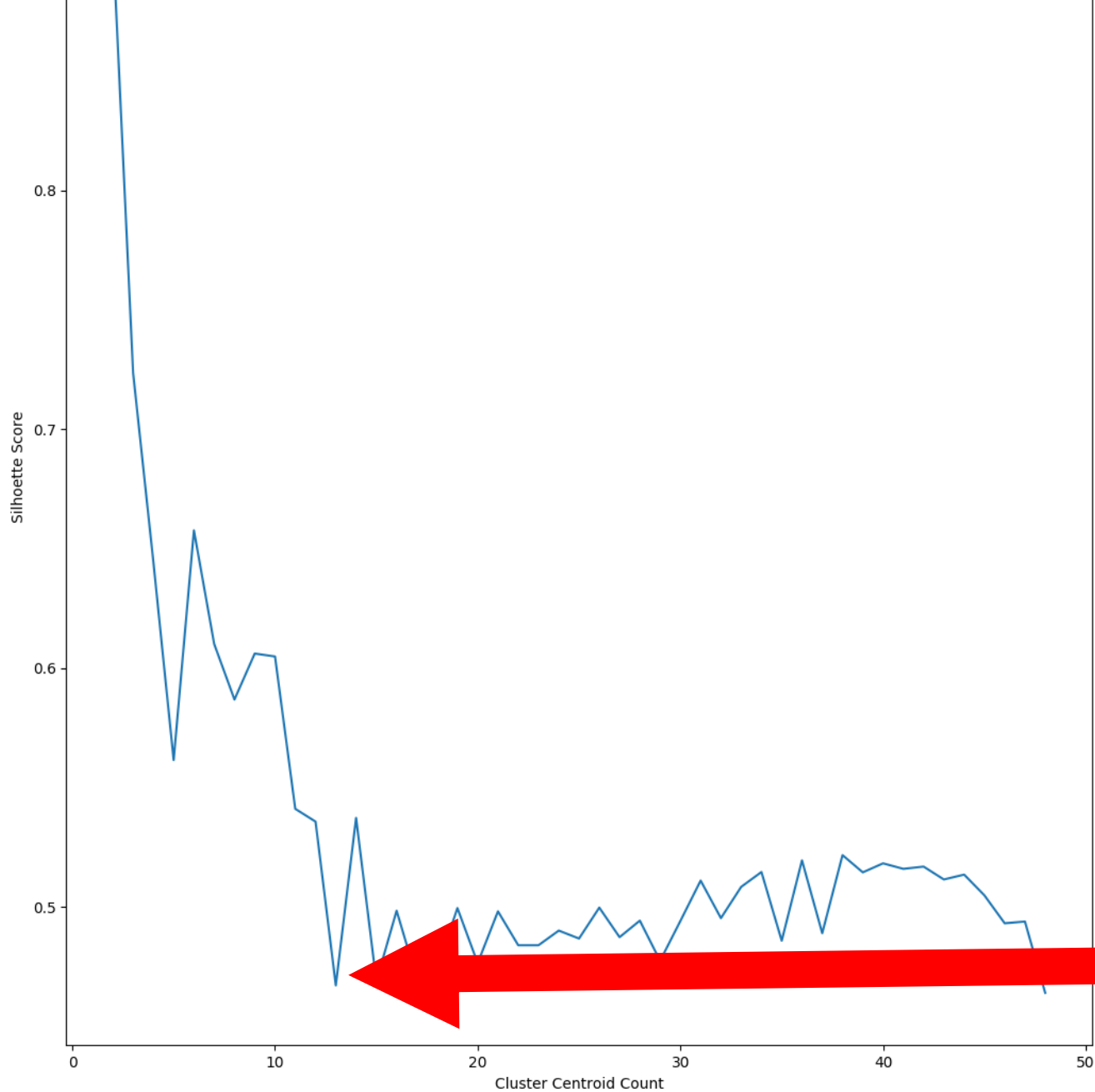
sym.mtd_write_firmware      in
    > 0.0
sym.mtd_write_firmware      in
    > 1.868202467580904e-05
sym.mtd_write_bootloader    in
    > 0.00015272111900810348

```



FINDING THE FUNCTION CLUSTER COUNT

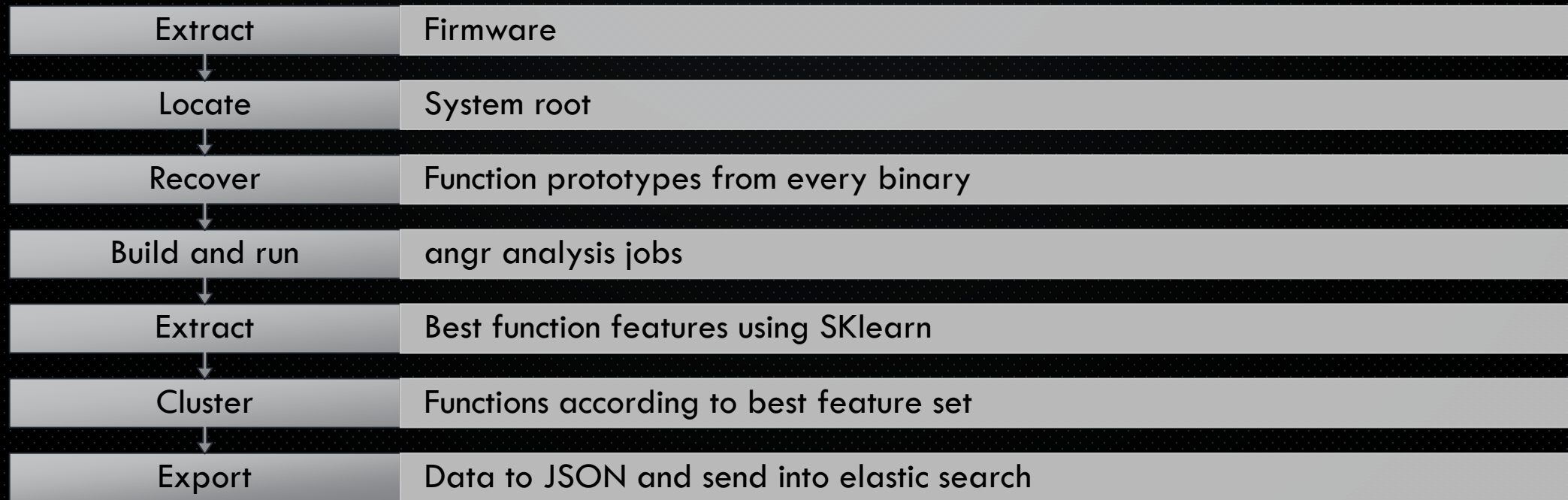
The trick is generally finding the biggest “drop” and choosing the count before that

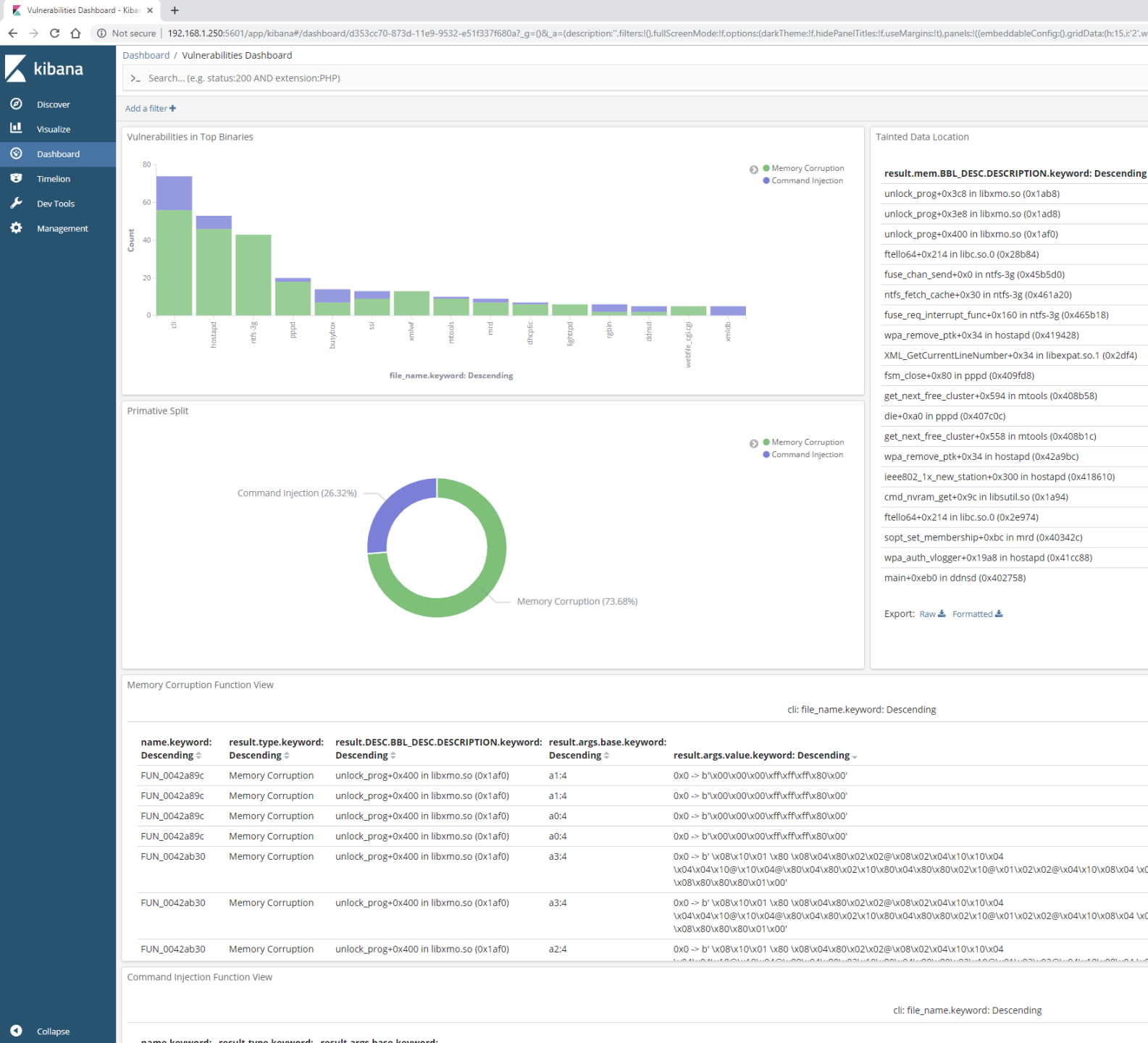


FINDING THE FUNCTION CLUSTER COUNT

- The trick is generally finding the biggest “drop” and choosing the count before that

FIRMWARE SLAP



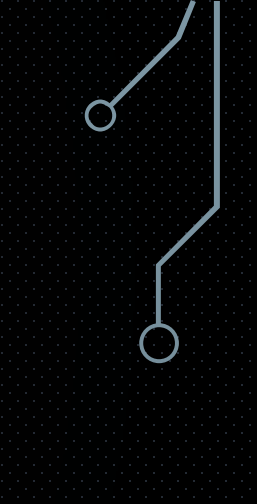
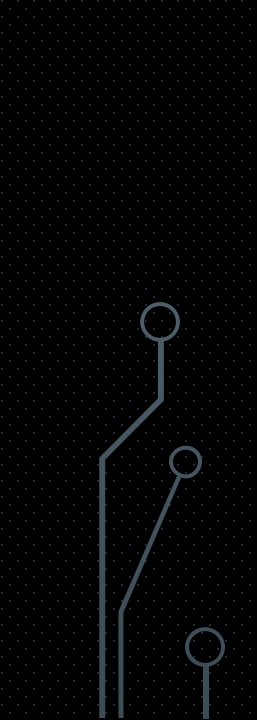


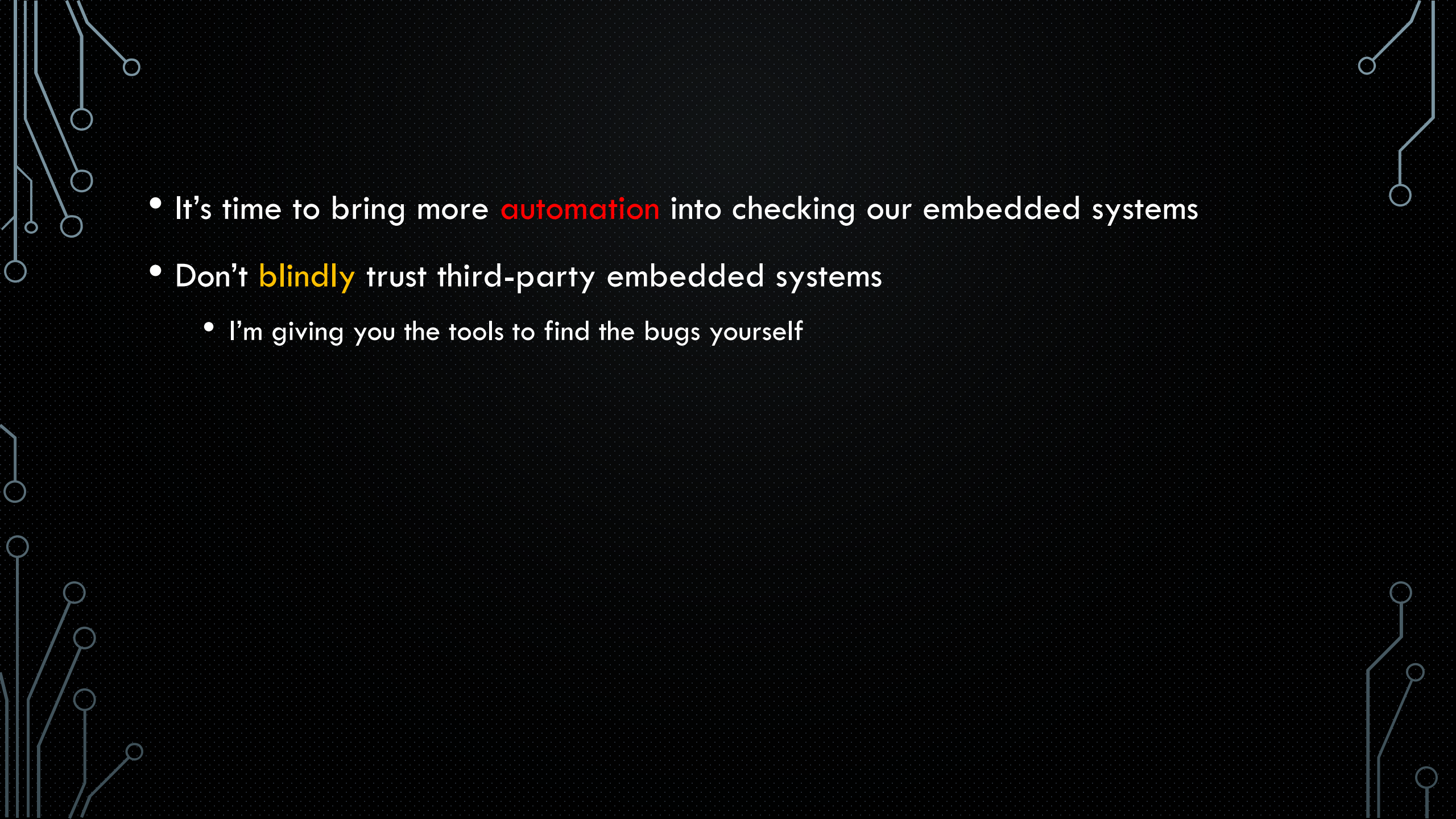
VISUALIZING VULNERABILITY RESULTS

- All information generated as **JSON** from both concolic and data mining pass
- Includes script to load information into **Elasticsearch** and **Kibana**



MITIGATIONS

- Use **compile time** protections
  - Enable your operating system's **ASLR**
  - Buy a better **router**
- 
- 
- 

- 
- The slide features a dark background with decorative circuit-like lines in the corners. These lines are composed of thin, light-colored segments that form a network of paths, some ending in small circles, resembling a stylized electronic circuit board.
- It's time to bring more **automation** into checking our embedded systems
  - Don't **blindly** trust third-party embedded systems
    - I'm giving you the tools to find the bugs yourself



RELEASING

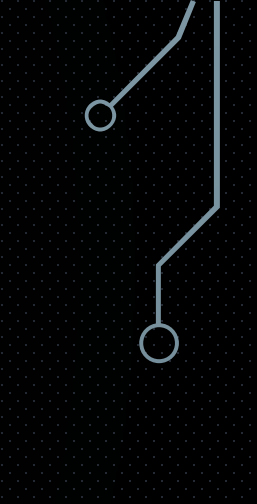
- **Firmware Slap** – The tool behind the demos
  - The **Ghidra** function dumping plugin
  - The cleaned-up **PoCs**
    - CVE-2019-13087 - CVE-2019-13092
- 



- Code:

- https://github.com/ChrisTheCoolHut/Firmware_Slap

- Feedback? Questions?

- @0x01_chris
- 
- 
- 