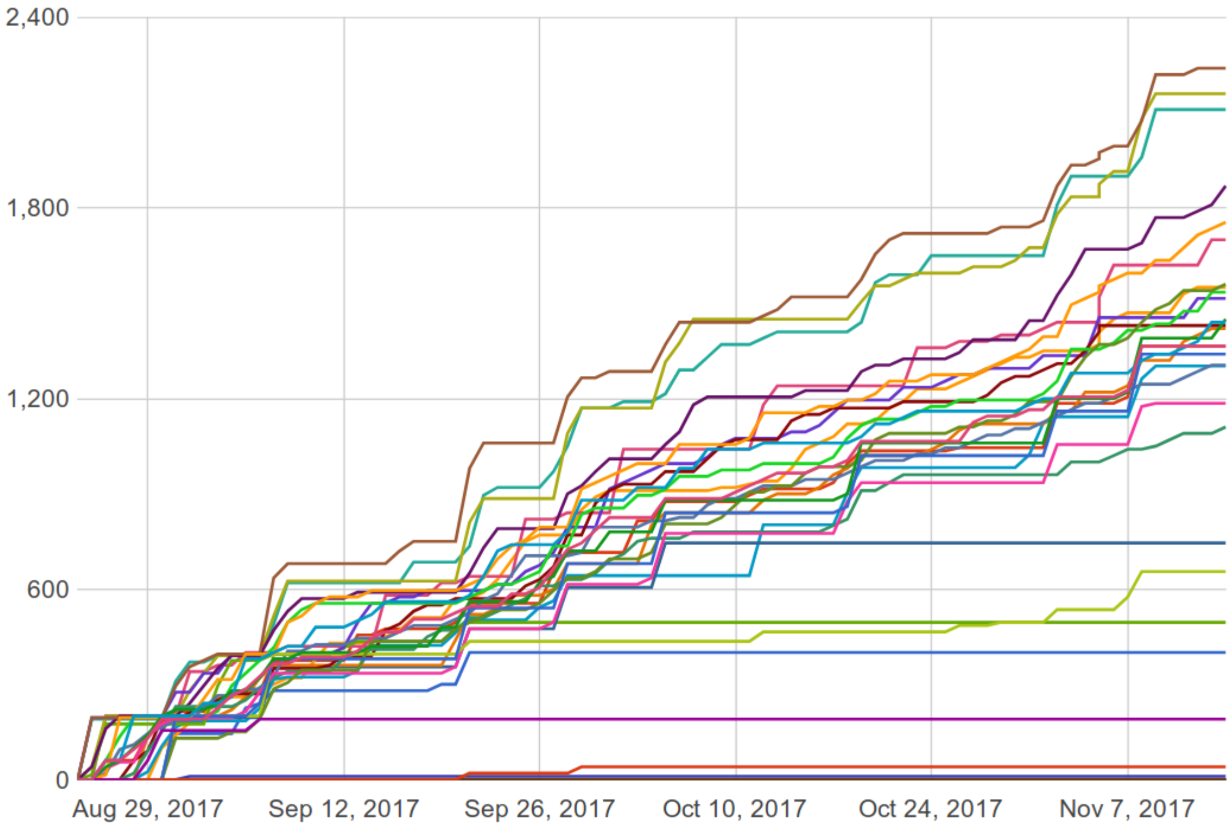
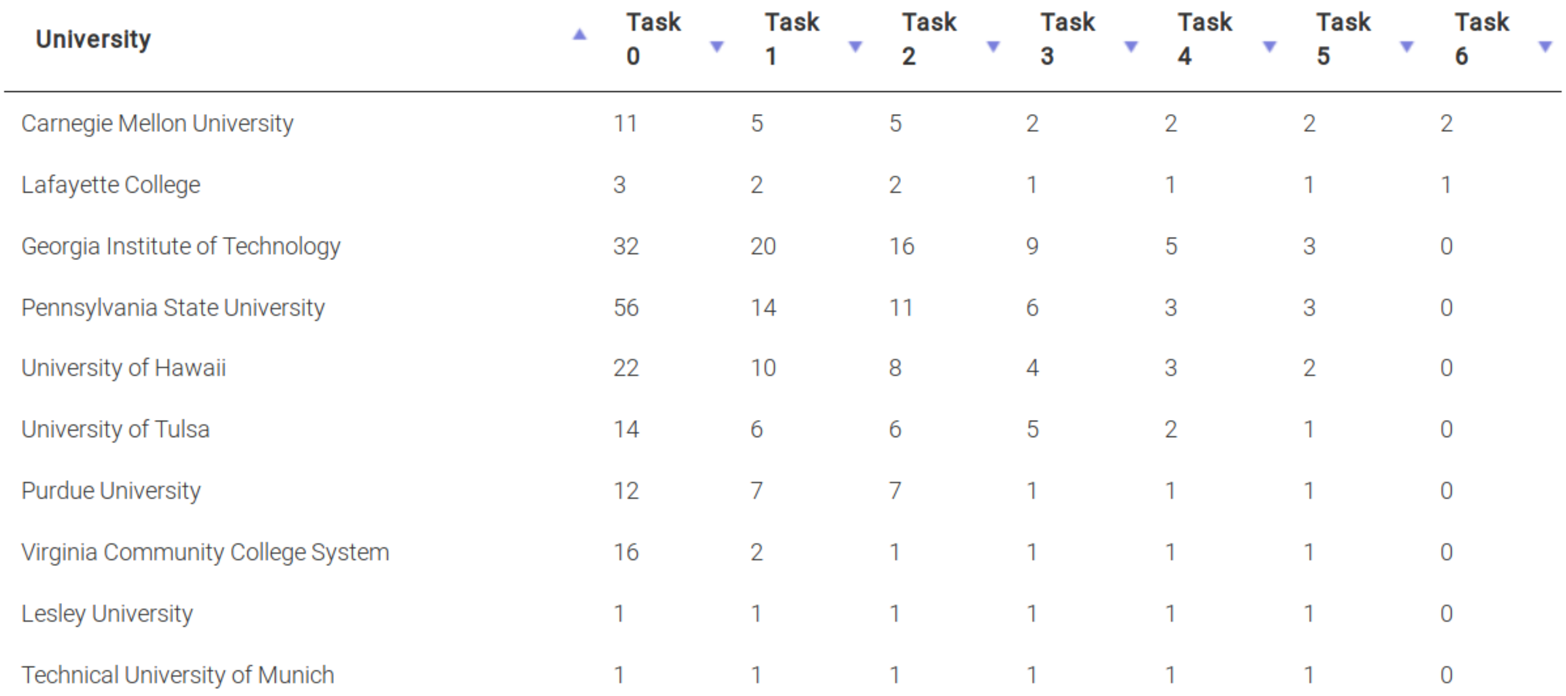
Lec11: Fuzzing

Taesoo Kim

Scoreboard


2,400
1,800
1,200
600
0
Aug 29, 2017
Sep 12, 2017
Sep 26, 2017
Oct 10, 2017
Oct 24, 2017
Nov 7, 2017

NSA Codebreaker Challenges



University	Task 0	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6
Carnegie Mellon University	11	5	5	2	2	2	2
Lafayette College	3	2	2	1	1	1	1
Georgia Institute of Technology	32	20	16	9	5	3	0
Pennsylvania State University	56	14	11	6	3	3	0
University of Hawaii	22	10	8	4	3	2	0
University of Tulsa	14	6	6	5	2	1	0
Purdue University	12	7	7	1	1	1	0
Virginia Community College System	16	2	1	1	1	1	0
Lesley University	1	1	1	1	1	1	0
Technical University of Munich	1	1	1	1	1	1	0

Administrivia

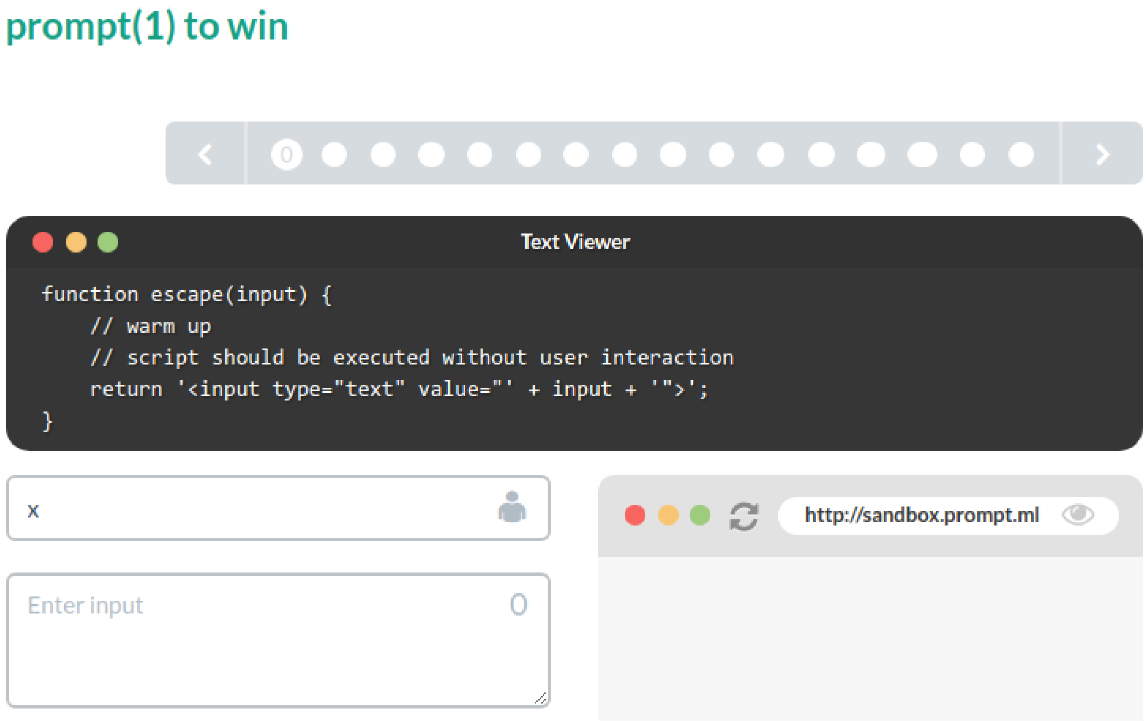
- Welcome to the last lab!
- Due: Lab10 on **Nov 23** (one week extension)
- Due: Lab04 / Lab11 on **Nov 30**
- Last lecture (Dec 1)
    - How to find bugs (by Insu)
    - Linux kernel UAF exploit (by Wen)
- Let you know your grade on Dec 1 in class

Lab this week

- Two options (same rules)
  - Sandboxing/kernel
  - Web exploitation

Web exploitation

- http://prompt.ml/



prompt(1) to win

```
function escape(input) {
    // warm up
    // script should be executed without user interaction
    return '<input type="text" value="' + input + '">';
}
```

x

Today: Fuzzing

- intro
- DEMO: fuzzing

So far, focuses are more on “exploitation”

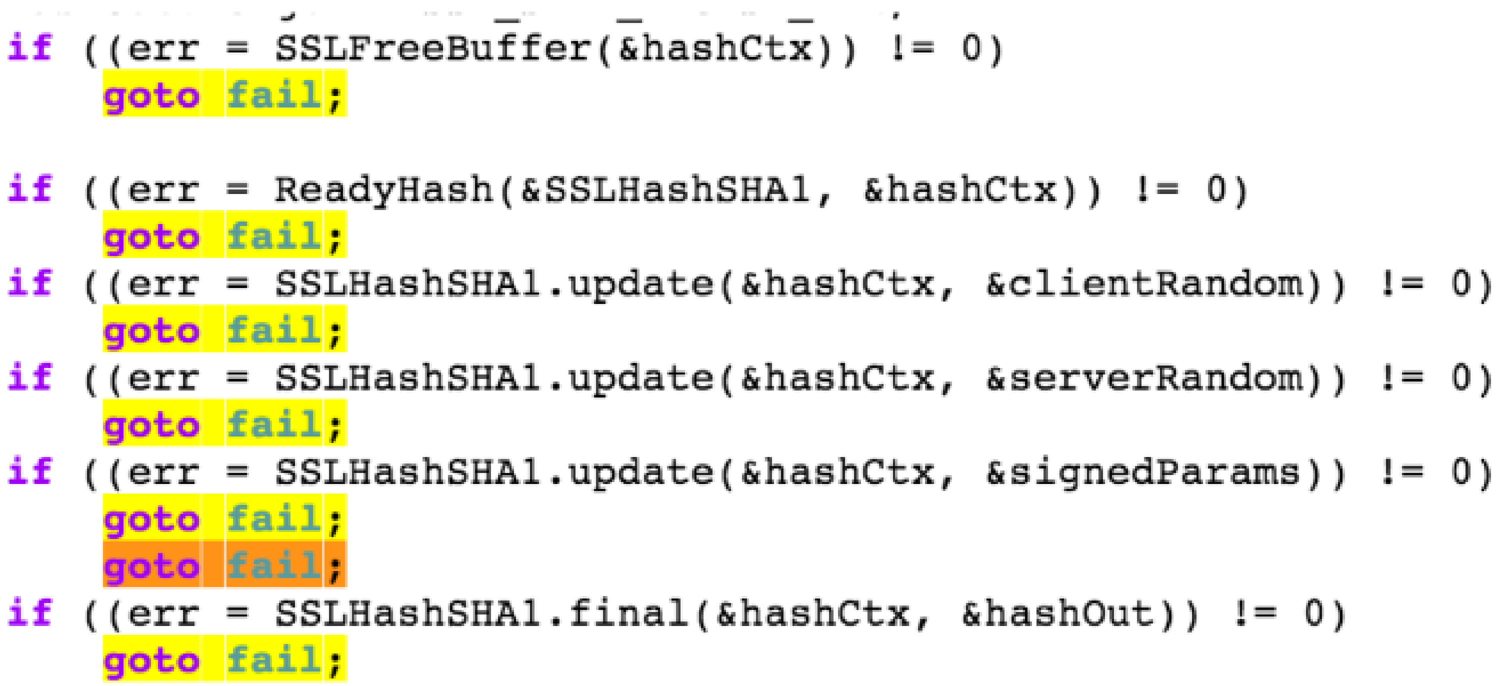
- More important question: how to find bugs?
  - often, with source code (we will see in the last lecture)
  - but mostly, with only binary

Two pre-conditions (often much difficult!)

- Locating a bug (i.e., bug finding)
- Triggering the bug (i.e., reachability)

```
if (magic == 0xdeadbeef)
    memcpy(dst, src, len)
```

Solution 1: Code Auditing (w/ code)



```
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;

if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
```

Solution 2: Static Analysis (on binary)

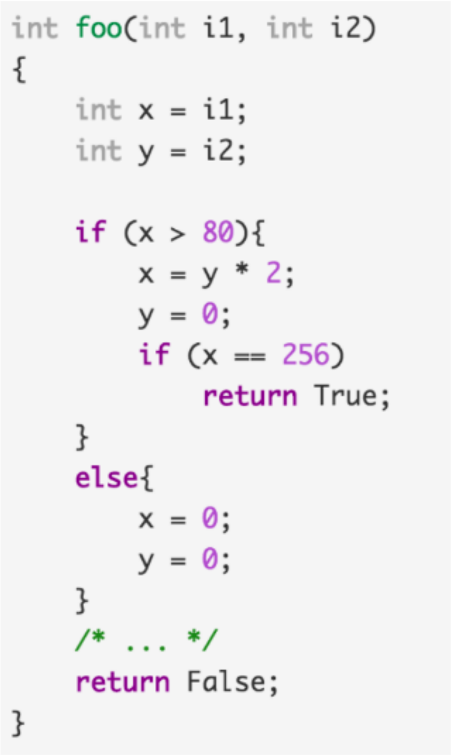
- Reverse Engineering (e.g., IDA)

Problem: Too Complex (e.g., browser)

Two Popular Directions

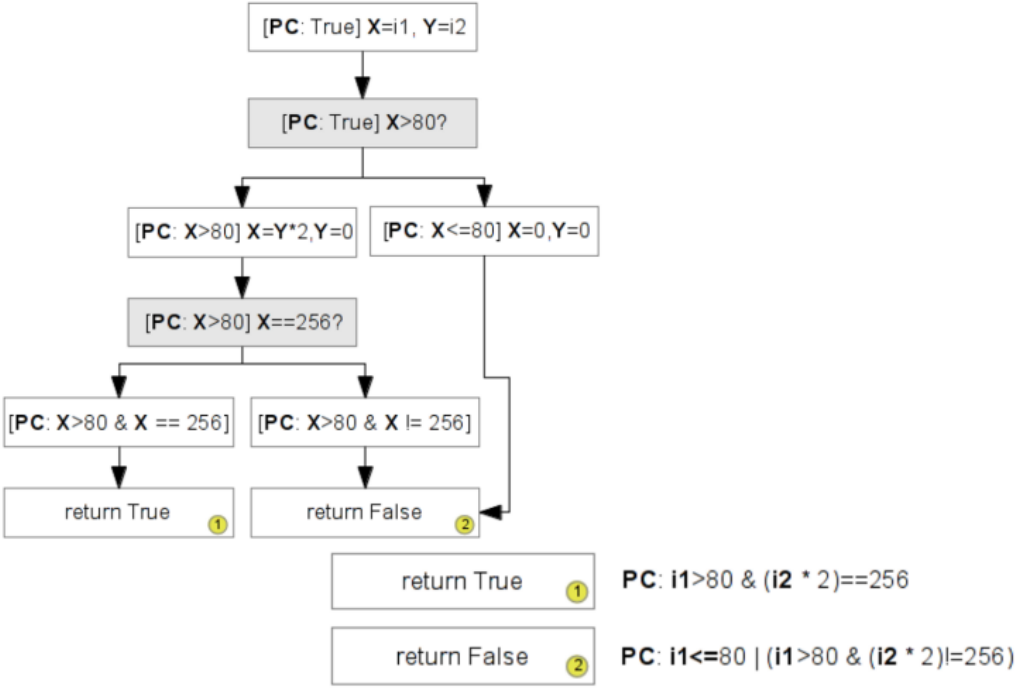
- Symbolic Execution (also static)
- Fuzzing (dynamic)

Symbolic Execution



```
int foo(int i1, int i2)
{
    int x = i1;
    int y = i2;

    if (x > 80){
        x = y * 2;
        y = 0;
        if (x == 256)
            return True;
    }
    else{
        x = 0;
        y = 0;
    }
    /* ... */
    return False;
}
```



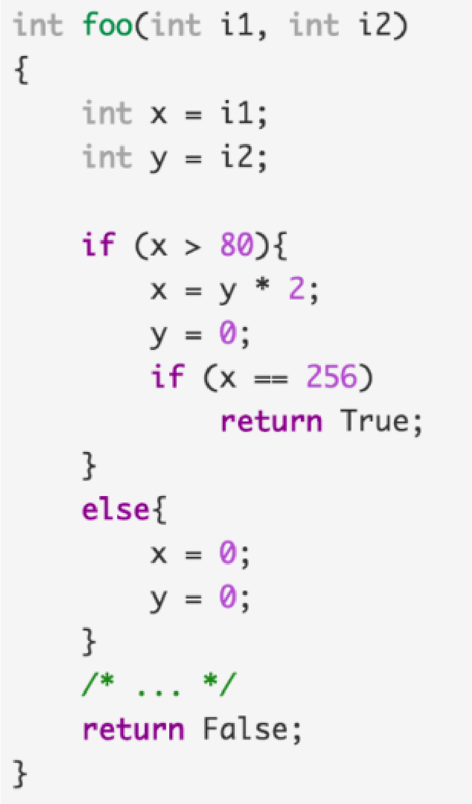
Problem: State Explosion

- Too many path to explore (e.g., strcmp(“hello”, input))
- Too huge state space (e.g., browser? OS?)
- Solving constraints is a hard problem

Today's Topic: Fuzzing

- Two key ideas
  - Reachability is given (since we are executing!)
  - Focus on quickly exploring the path/state
    - How? mutating inputs
    - How well? e.g., coverage

Example: How well fuzzing can explore all paths?



```
int foo(int i1, int i2)
{
    int x = i1;
    int y = i2;

    if (x > 80){
        x = y * 2;
        y = 0;
        if (x == 256)
            return True;
    }
    else{
        x = 0;
        y = 0;
    }
    /* ... */
    return False;
}
```

Game Changing Fact: Speed

- In this example,
  - Symbolic execution explores/checks just two conditions
  - Fuzzing requires 256 times (by scanning values from 0 to 256)
- But, what if fuzzer is an order of magnitude faster (say, 10k times)?

Importance of High-quality Corpus

- In fact, fuzzing is really bad at exploring paths
    - e.g., if (a == 0xdeadbeef)
- So, paths should be (or mostly) given by corpus (sample inputs)
    - e.g., pdf files utilizing full features
    - but, not too many! (do not compromise your performance)
- A fuzzer will trigger the exploitable state
    - e.g., len in malloc()

AFL (American Fuzzy Lop)

- VERY well-engineered fuzzer w/ lots of heuristics

Examples of Mutation Techniques

- interest: -1, 0x8000000, 0xffff, etc
- bitflip: flipping 1,2,3,4,8,16,32 bits
- havoc: random tweak in fixed length
- extra: dictionary, etc
- etc

Key Idea: Mapping Input to State Transitions

- Input → [IPs] (problem?)

Key Idea: Mapping Input to State Transitions

- Input → [IPs] (problem?)
- Input → map[IPs % len] (problem? A→B vs B→A)

Key Idea: Mapping Input to State Transitions

- Input → [IPs] (problem?)
- Input → map[IPs % len] (problem? A→B vs B→A)
- Input → map[(prevIP >> 1 ^ curIP) % len] (problem?)

Key Idea: Mapping Input to State Transitions

- Input → [IPs] (problem?)
- Input → map[IPs % len] (problem? A→B vs B→A)
- Input → map[(prevIP >> 1 ^ curIP) % len] (problem?)
- Input → map[(rand1 >> 1 ^ rand2) % len]

Key Idea: Avoiding Redundant Paths

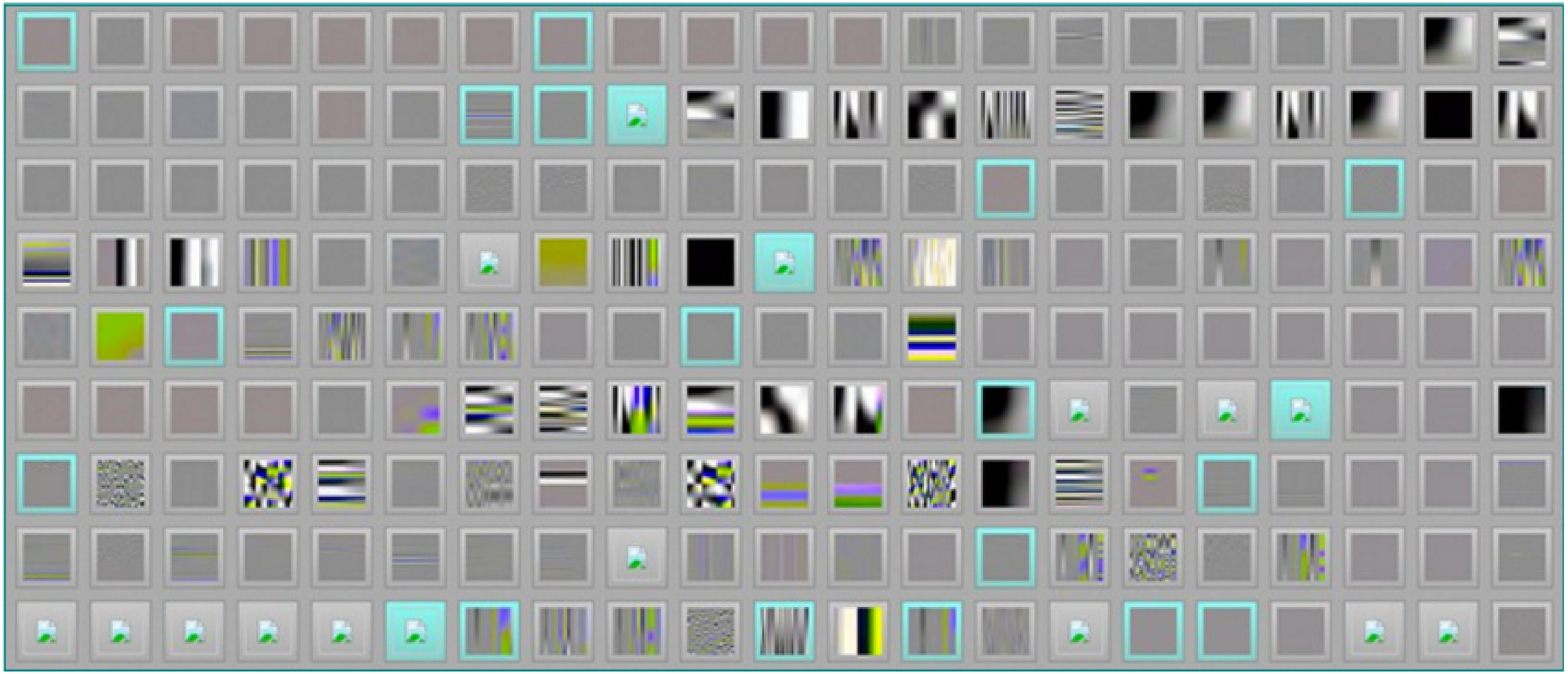
- If you see the duplicated state, throw out
  - e.g., i1 = 1, 2, 3
- If you see the new path, keep it for further exploration
  - e.g., i1 = 81

How to Create Mapping?

- Instrumentation
  - Source code → compiler (e.g., gcc, clang)
  - Binary → QEMU

```
if (block_address > elf_text_start && block_address < elf_text_end) {
  cur_location = (block_address >> 4) ^ (block_address << 8)
  shared_mem[cur_location ^ prev_location] ++;
  prev_location = cur_location >> 1;
}
```

AFL Arts



Other Types of Fuzzer

- Radamsa: syntax-aware fuzzer
- Cross-fuzz: function syntax for Javascript
- langfuzz: fuzzing program languages
- Driller: fuzzing + symbolic execution

Today’s Tutorial

- In-class tutorial:
  - Fuzzing with source code
  - Fuzzing on binary
  - Fuzzing a real-world program

In-class Tutorial

```
$ git git@clone tc.gtisc.gatech.edu:seclab-pub cs6265
```

or

```
$ cd tut/lec11
$ cat README
```

Problem: Too Complex (e.g., browser)