SOFT620020.02
Advanced Software Engineering

Bihuan Chen, Pre-Tenure Assoc. Prof.

bhchen@fudan.edu.cn

https://chenbihuan.github.io

Course Outline

Date	Topic	Date	Topic
Sep. 10	Introduction	Nov. 05	Compiler Testing
Sep. 17	Testing Overview	Nov. 12	Mobile Testing
Sep. 24	Holiday	Nov. 19	Delta Debugging
Oct. 01	Holiday	Nov. 26	Presentation 1
Oct. 08	Guided Random Testing	Dec. 03	Bug Localization
Oct. 15	Search-Based Testing	Dec. 10	Automatic Repair
Oct. 22	Performance Analysis	Dec. 17	Symbolic Execution
Oct. 29	**Security Testing**	Dec. 24	Presentation 2

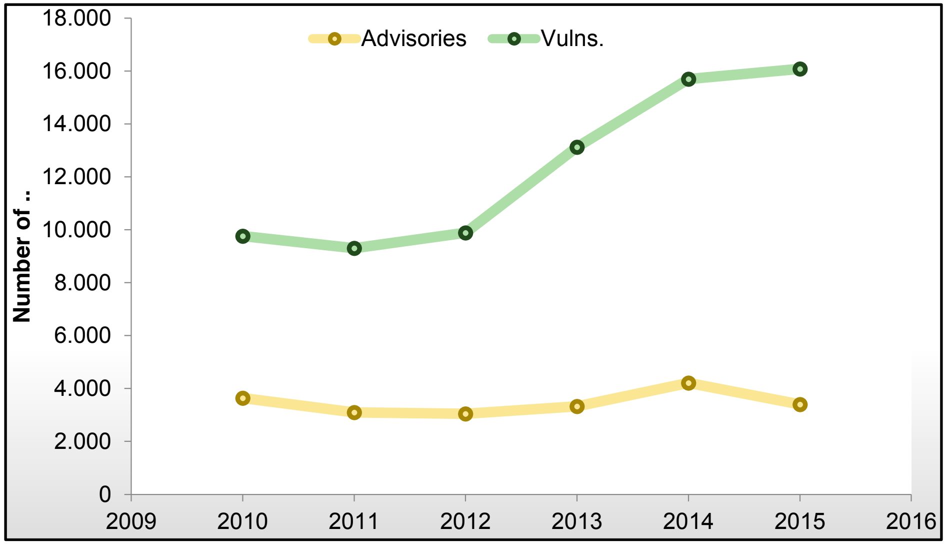
Discussion – What is a Security Bug?

- Security bug is a software bug that can be exploited to gain unauthorized access or privileges on a computer system

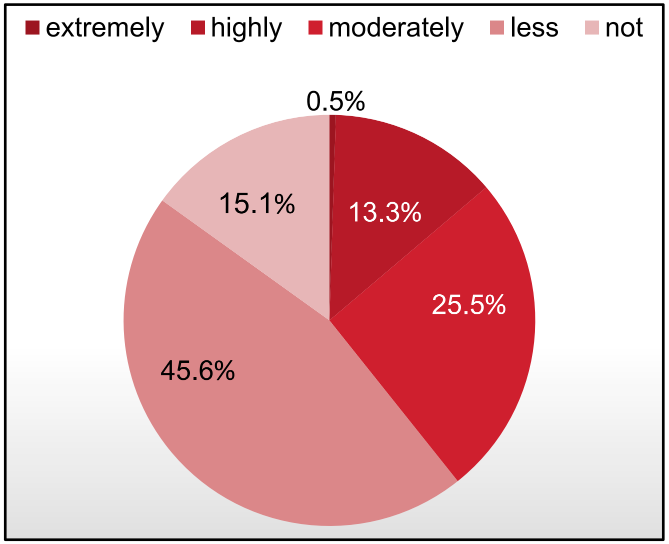
Software Vulnerability

Vulnerability Review in 2016

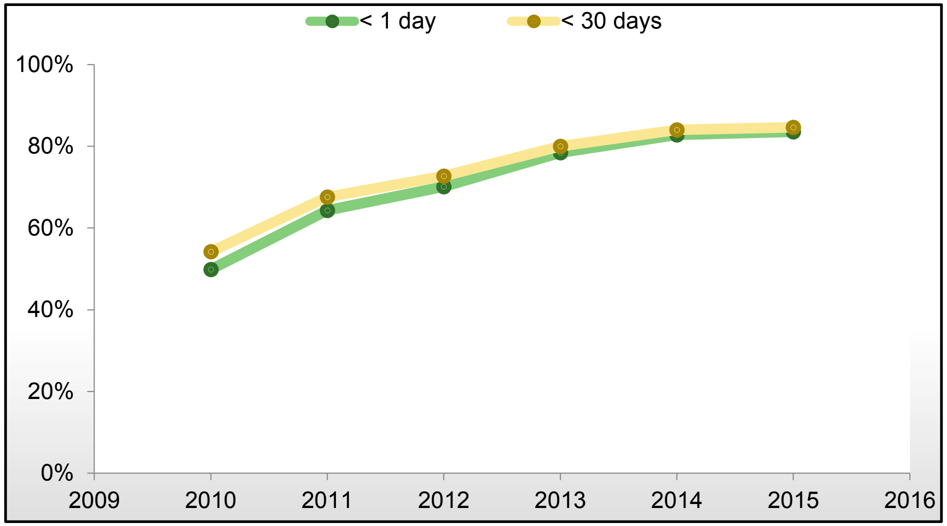
Number



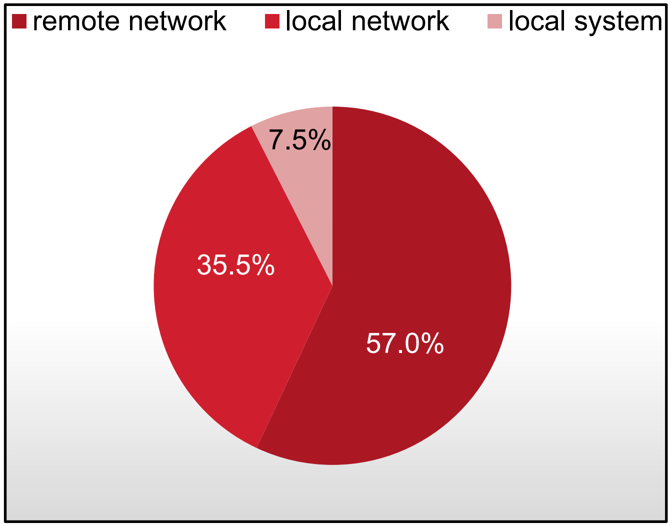
Criticality



Patching Time



Attack Vector



Bug Bounty Programs

- Bug bounty: pay rewards to independent security researchers for finding vulnerabilities in their products
  - Major players: Google, Mozilla, Facebook, PayPal, …
  - What we get: money and fame
  - What the company get: secured applications
  - Rewards can range from $200 to $20,000 or more

Memory Corruptions – Buffer Overflow

- Data written to a buffer corrupts data in memory addresses adjacent to the buffer due to insufficient bounds checking

char A[8] = "";

unsigned short B = 1979;

variable name	A								B	
value	[null string]								1979	
hex value	00	00	00	00	00	00	00	00	07	BB

strcpy(A, "excessive"); → strncpy(A, "excessive", sizeof(A));

variable name	A								B	
value	‘e’	‘x’	‘c’	‘e’	‘s’	‘s’	‘i’	‘v’	25856	
hex value	65	78	63	65	73	73	69	76	65	00

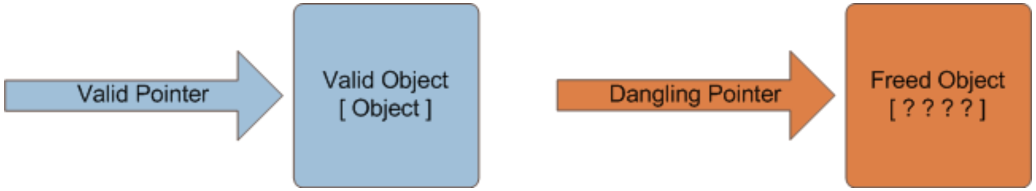
Discussion – Where is the Buffer Overflow?

```
char *lccopy(const char *str) {
    char buf[BUFSIZE];
    char *p;
    strcpy(buf, str);
    for (p = buf; *p; p++) {
        if (isupper(*p)) {
            *p = tolower(*p);
        }
    }
    return strdup(buf);
}
```

```
char buf[64], in[MAX_SIZE];
printf("Enter buffer contents:\n");
read(0, in, MAX_SIZE-1);
printf("Bytes to copy:\n");
scanf("%d", &bytes);
memcpy(buf, in, bytes);
```

Memory Corruptions – Use After Free

- Dereference a dangling pointer storing the address of an object that has been deleted




```
char* ptr = (char*) malloc (SIZE);
if (err) {
    abort = 1;
    free(ptr);
    ptr = null;
}
...
if (abort) {
    logError("operation aborted before commit", ptr);
}
```

Discussion – Where is the Use After Free?

```
char* buf_use = (char*)malloc(32);
printf("buf_use's addr = [0x%08X]\n", buf_use);
strcpy(buf_use, "1");
printf("You have money %s$.\n", buf_use);
free(buf_use);

char* buf_new = (char*)malloc(32);
printf("buf_new's addr = [0x%08X]\n", buf_new);
strcpy(buf_new, "999");

printf("You have money %s$.\n", buf_use);
```

Output:

```
buf_use's addr = [0x00032F98]
You have money 1$.
buf_new's addr = [0x00032F98]
You have money 999$.
```

Input Validation Errors – SQL Injection

- Take advantage of the syntax of SQL to inject commands that can read or modify a database, or compromise the meaning of the original query

```
SELECT UserList.Username FROM UserList
WHERE UserList.Username = 'Username' AND UserList.Password = 'Password'
```

set Password to Password' OR '1'='1

```
SELECT UserList.Username FROM UserList
WHERE UserList.Username = 'Username' AND UserList.Password = 'Password' OR '1'='1'
```

```
SELECT User.UserID FROM User
WHERE User.UserID = 'UserID' AND User.Pwd = 'Password'
```

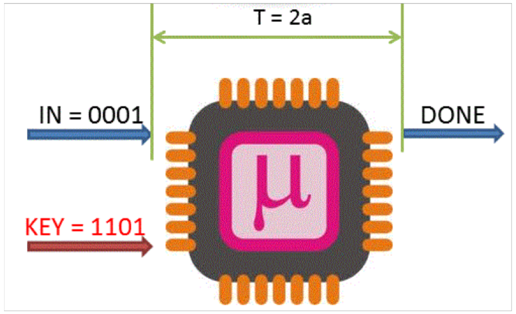
set ';DROP TABLE User; --'

```
SELECT User.UserID FROM User
WHERE User.UserID = '';DROP TABLE User; --'AND User.Pwd = ''OR''='
```

Side Channel Attacks – Timing Attack

- Compromise a cryptosystem by analyzing the time taken to execute cryptographic algorithms

```
MES = IN XOR KEY;
FOR EACH b BIT in MES {
 IF (b == 1) routine();
}
```



User input	Exec. time	Time prediction for KEY_0=0000	Time prediction for KEY_1=0001	Time prediction for KEY_j=XXXX	Time prediction for KEY_{13}=1101
0001	2 ms	1	0	…	2
0010	4 ms	1	2	…	4
0011	3 ms	2	1	…	3
0100	2 ms	1	2	…	2

Fuzzing Overview

Fuzzing (Fuzz Testing)

- Fuzzing is an automated software testing technique
  - Feed malformed inputs to programs to trigger unintended behaviors
  - Trigger crashes and find bugs
  - Widely used by mainstream software companies

- You already know how to fuzz!

Discussion – Fuzzing is Simple?

- How often did you encounter browser crashes, Adobe reader crashes, Microsoft office crashes, video player crashes, etc.?
- Why is the chance of getting program crashes so low?
  - **Feed well-formed/expected inputs to the programs under fuzz**
  - **We need to generate mal-formed/unexpected inputs, but how?**

Mutation Based Fuzzing (Dumb)

- Little or no knowledge of the structure of the inputs is assumed
- Anomalies are added to existing valid inputs via mutation
- Anomalies may be completely random or follow some heuristics

Example: Fuzzing a PDF Viewer

- Google for PDF files (about 1 billion results)
- Crawl pages to build a corpus of PDF files
- Use fuzzing tool (or script to)
  1. Select a PDF file from the corpus
  2. Mutate that file
  3. Feed it to the program under fuzz
  4. Record if it crashed (and input that crashed it)

Mutation Based Fuzzing In Short

- Strengths
  - Super easy to setup and automate
  - Little or no structure knowledge required
  - Very effective to fuzz programs that process compact or unstructured inputs (e.g., images and videos)

- Weaknesses
  - Limited by the initial corpus
  - Less effective to fuzz programs that process highly-structured inputs (e.g., XSL and JavaScript)

Generation Based Fuzzing (Smart)

- Inputs are generated from a specification, e.g., input models that specify the format of data chunks and integrity constraints, and context- free grammars that describe the syntax features

- Structure knowledge should give better results than mutation based fuzzing

Example: Protocol Description

```
//png.spk
//author: Charlie Miller

// Header - fixed.
s_binary("89504E470D0A1A0A");

// IHDRChunk
s_binary_block_size_word_bigendian("IHDR"); //size of data field
s_block_start("IHDRcrc");
s_string("IHDR");  // type
s_block_start("IHDR");
// The following becomes s_int_variable for variable stuff
// 1=BINARYBIGENDIAN, 3=ONEBYE
s_push_int(0x1a, 1);     // Width
s_push_int(0x14, 1);     // Height
s_push_int(0x8, 3);      // Bit Depth - should be 1,2,4,8,16, based on colortype
s_push_int(0x3, 3);      // ColorType - should be 0,2,3,4,6
s_binary("00 00");       // Compression || Filter - shall be 00 00
s_push_int(0x0, 3);      // Interlace - should be 0,1
s_block_end("IHDR");
s_binary_block_crc_word_littleendian("IHDRcrc"); // crc of type and data
s_block_end("IHDRcrc");
...
```

Generation Based Fuzzing In Short

- Strengths
  - Completeness
  - Can deal with complex dependencies, e.g. checksums

- Weaknesses
  - Have to have a specification
  - Writing generator can be labor intensive for complex specifications
  - The specification is not the code

Problem Detection

- See if program crashed
  - Type of a crash can tell a lot (SEGV vs. assertion failure)
- Run program under dynamic memory error detector (e.g., valgrind/purify)
  - Catch more bugs, but more expensive per run
- See if program locks up
- Roll your own checker e.g. valgrind skins

How Much Fuzz Is Enough?

- Mutation based fuzzers can generate an infinite number of test inputs. When has the fuzzer run long enough?
- Generation based fuzzers can generate a finite number of test inputs. What happens when they are all run and no bugs are found?

- Some of the answers to these questions lie in **code coverage**
- Code coverage is a metric which can be used to determine how much code has been executed
- Data can be obtained using various profiling tools, e.g., gcov

Types of Code Coverage

- Line coverage
  - Measure how many lines of source code have been executed
- Branch coverage
  - Measure how many branches in code have been taken
- Path coverage
  - Measure how many paths have been taken

Example

```
if( a > 2 )
  a = 2;
if( b > 2 )
  b = 2;
```

- Requires
  - 1 test case for line coverage, e.g., (3, 3)
  - 2 test cases for branch coverage, e.g., (0, 0), (3, 3)
  - 4 test cases for path coverage, e.g., (0,0), (3,0), (0,3), (3,3)

Code Coverage is Good For Lots of Things

- How good is this initial file?
- Am I getting stuck somewhere?

```
if(packet[0x10] < 7) { //hot path
} else { //cold path
}
```

- How good is fuzzer X vs. fuzzer Y?
- Am I getting benefits from running a different fuzzer?

American Fuzzy Lop (AFL)

Michal Zalewski

http://lcamtuf.coredump.cx/afl/

AFL Can Find Security Bugs

IJG jpeg [1]	libjpeg-turbo [1,2]	libpng [1]
libtiff [1,2,3,4,5]	mozjpeg [1]	PHP [1,2,3,4,5,6,7,8]
Mozilla Firefox [1,2,3,4]	Internet Explorer [1,2,3,4]	Apple Safari [1]
Adobe Flash / PCRE [1,2,3,4,5,6,7]	sqlite [1,2,3,4...]	OpenSSL [1,2,3,4,5,6,7]
LibreOffice [1,2,3,4]	poppler [1,2...]	freetype [1,2]
GnuTLS [1]	GnuPG [1,2,3,4]	OpenSSH [1,2,3,4,5]
PuTTY [1,2]	ntpd [1,2]	nginx [1,2,3]
bash (post-Shellshock) [1,2]	tcpdump [1,2,3,4,5,6,7,8,9]	JavaScriptCore [1,2,3,4]
pdfium [1,2]	ffmpeg [1,2,3,4,5]	libmatroska [1]
libarchive [1,2,3,4,5,6,...]	wireshark [1,2,3]	ImageMagick [1,2,3,4,5,6,7,8,9,...]
BIND [1,2,3,...]	QEMU [1,2]	lcms [1]
Oracle BerkeleyDB [1,2]	Android / libstagefright [1,2]	iOS / ImageIO [1]

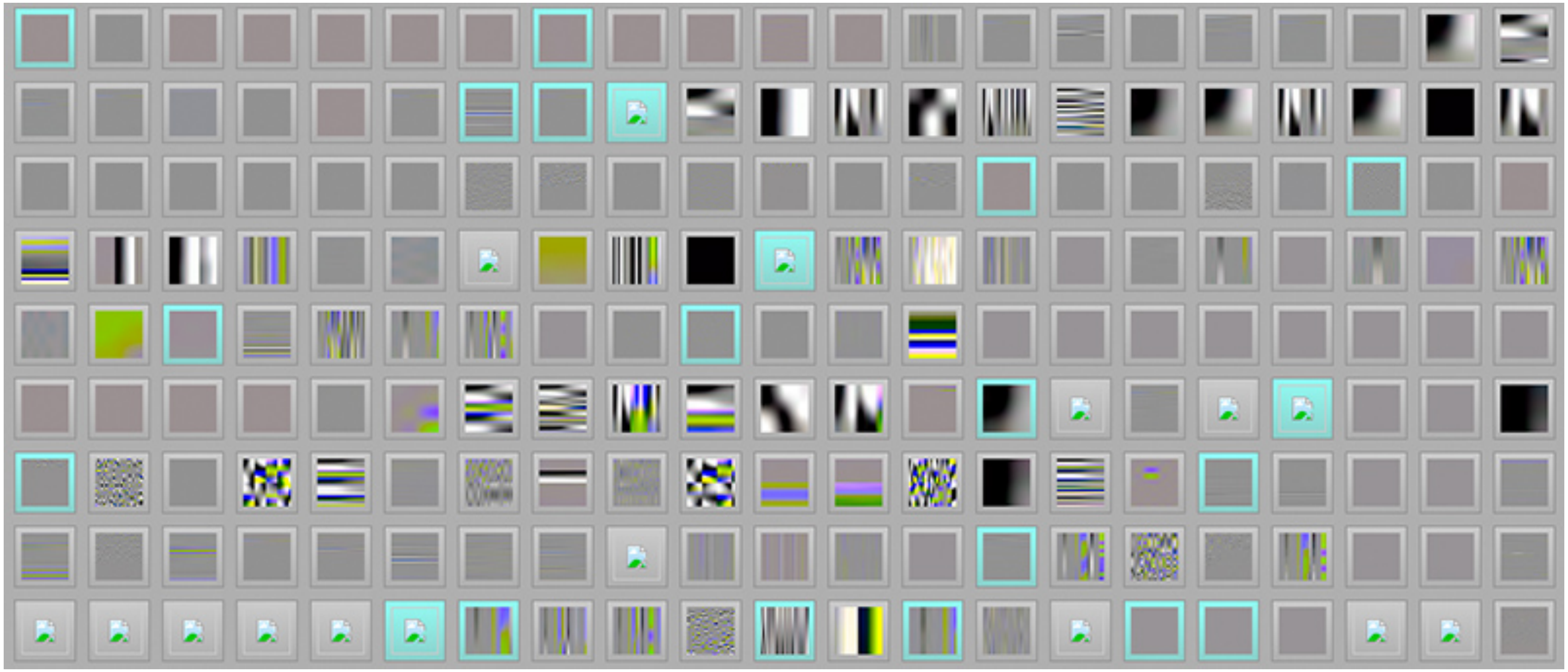
FLAC audio library [1,2]	libsndfile [1,2,3,4]	less / lesspipe [1,2,3]
strings (+ related tools) [1,2,3,4,5,6,7]	file [1,2,3,4]	dpkg [1,2]
rcs [1]	systemd-resolved [1,2]	libyaml [1]
Info-Zip unzip [1,2]	libtasn1 [1,2,...]	OpenBSD pfctl [1]
NetBSD bpf [1]	man & mandoc [1,2,3,4,5,...]	IDA Pro [reported by authors]
clamav [1,2,3,4,5,6]	libxml2 [1,2,4,5,6,7,8,9,...]	glibc [1]
clang / llvm [1,2,3,4,5,6,7,8,...]	nasm [1,2]	ctags [1]
mutt [1]	procmail [1]	fontconfig [1]
pdksh [1,2]	Qt [1,2...]	wavpack [1,2,3,4]
redis / lua-cmsgpack [1]	taglib [1,2,3]	privoxy [1,2,3]
perl [1,2,3,4,5,6,7...]	libxmp	radare2 [1,2]
SleuthKit [1]	fwknop [reported by author]	X.Org [1,2]

FLAC audio library [1,2]	libsndfile [1,2,3,4]	less / lesspipe [1,2,3]
strings (+ related tools) [1,2,3,4,5,6,7]	file [1,2,3,4]	dpkg [1,2]
rcs [1]	systemd-resolved [1,2]	libyaml [1]
Info-Zip unzip [1,2]	libtasn1 [1,2,...]	OpenBSD pfctl [1]
NetBSD bpf [1]	man & mandoc [1,2,3,4,5,...]	IDA Pro [reported by authors]
clamav [1,2,3,4,5,6]	libxml2 [1,2,4,5,6,7,8,9,...]	glibc [1]
clang / llvm [1,2,3,4,5,6,7,8,...]	nasm [1,2]	ctags [1]
mutt [1]	procmail [1]	fontconfig [1]
pdksh [1,2]	Qt [1,2...]	wavpack [1,2,3,4]
redis / lua-cmsgpack [1]	taglib [1,2,3]	privoxy [1,2,3]
perl [1,2,3,4,5,6,7...]	libxmp	radare2 [1,2]
SleuthKit [1]	fwknop [reported by author]	X.Org [1,2]

dhcpcd [1]	Mozilla NSS [1]	Nettle [1]
mbed TLS [1]	Linux netlink [1]	Linux ext4 [1]
Linux xfs [1]	botan [1]	expat [1,2]
Adobe Reader [1]	libav [1]	libical [1]
OpenBSD kernel [1]	collectd [1]	libidn [1,2]
MatrixSSL [1]	jasper [1,2,3,4,5,6,7,...]	MaraDNS [1]
w3m [1,2,3,4]	Xen [1]	OpenH232 [1...]
irssi [1,2,3]	cmark [1]	OpenCV [1]
Malheur [1]	gstreamer [1...]	Tor [1]
gdk-pixbuf [1]	audiofile [1,2,3,4,5,6,...]	zstd [1]
lz4 [1]	stb [1]	cJSON [1]
libpcre [1,2,3]	MySQL [1]	gnulib [1]

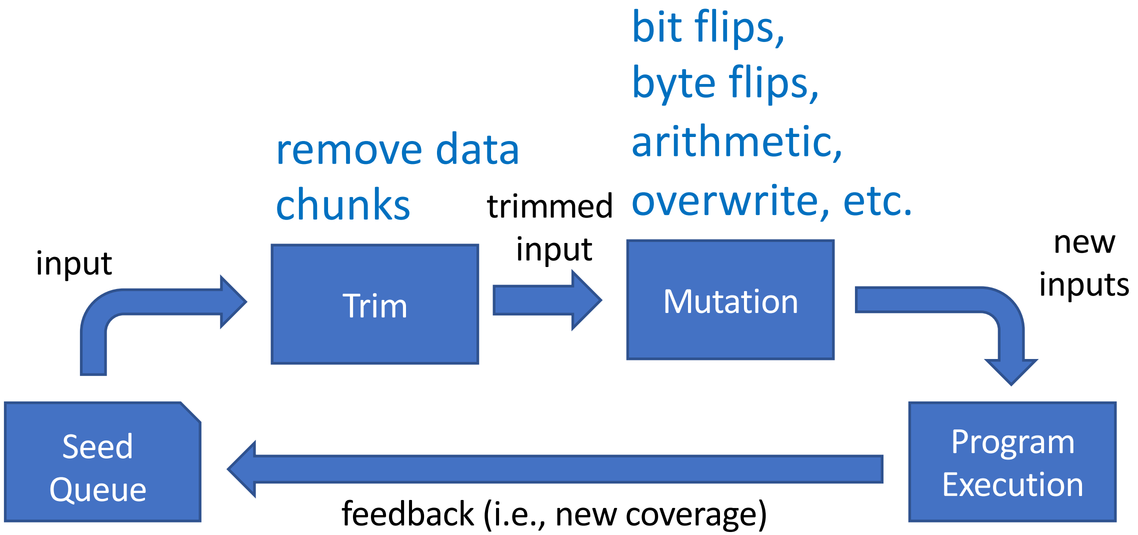
AFL is Spooky

- Fuzz a JPEG image library djpeg with a text file containing just "hello"
- Start to produce valid jpeg files after eight hours

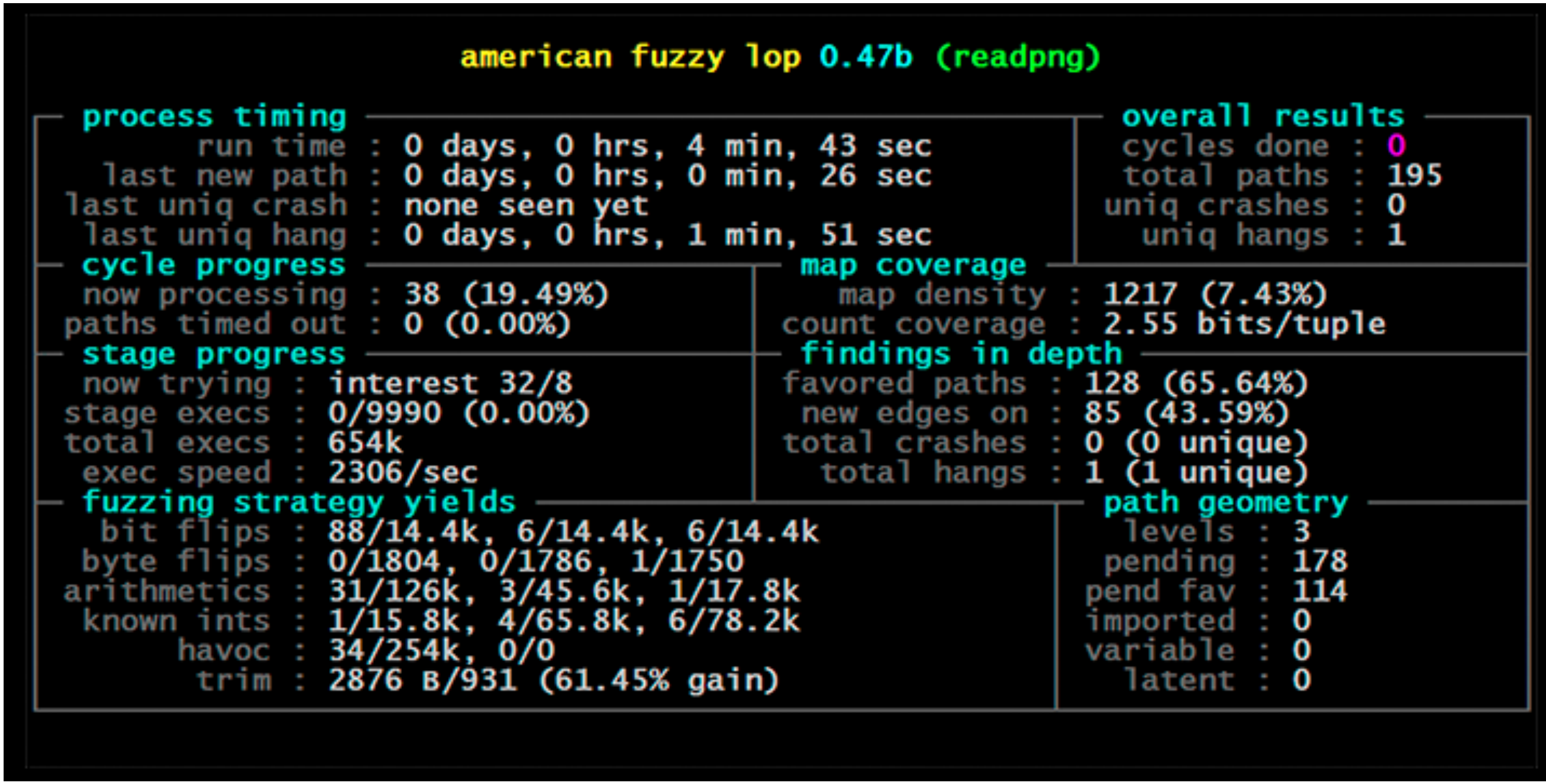


AFL – Coverage-Guided Gray-box Fuzzer

1) load user-supplied initial test cases into the queue
2) take next input file from the queue
3) trim the input to the smallest size that does not change the program behavior
4) repeatedly mutate the input using a variety of traditional fuzzing strategies
5) if any of the generated mutations resulted in a new state transition recorded by the instrumentation, add mutated input as an interesting input in the queue
6) go to 2)



Status Screen of AFL



Discussion – Using AFL

```c
int main(void) {
  if (getchar() == 'M')
    if (getchar() == 'A')
      if(getchar() == 'Z')
        if(getchar() == 'E')
          // trigger the crash
  return 0;
}
```

at most 2^8 (256) mutations

XXXX	→	MXXX	→	MAXX	→	MAZX	→	MAZE
initial input		mutated input		mutated input		mutated input		mutated input

Discussion – Using AFL (cont.)

```c
int main(void) {
  char str[4];
  gets(str);
  if(strcmp(str, "MAZE") == 0)
    // trigger the crash
  return 0;
}
```

- Can AFL trigger the crash?
  - 4 bytes = 1/24*8 (1/4294967296) probability
  - Hard for the fuzzer to ”guess” the bytes correctly all at once

XXXX	→	MXXX
initial input		mutated input

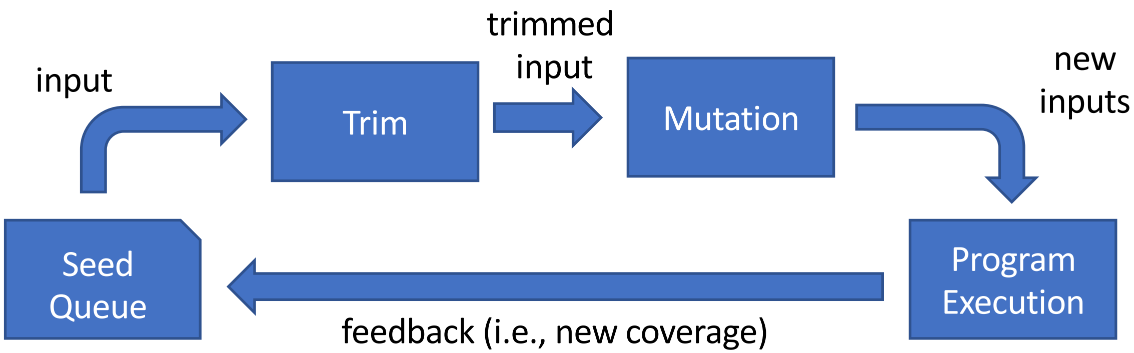
Data-Driven Seed Generation for Fuzzing

Junjie Wang, Bihuan Chen, Lei Wei, and Yang Liu

S&P 2017

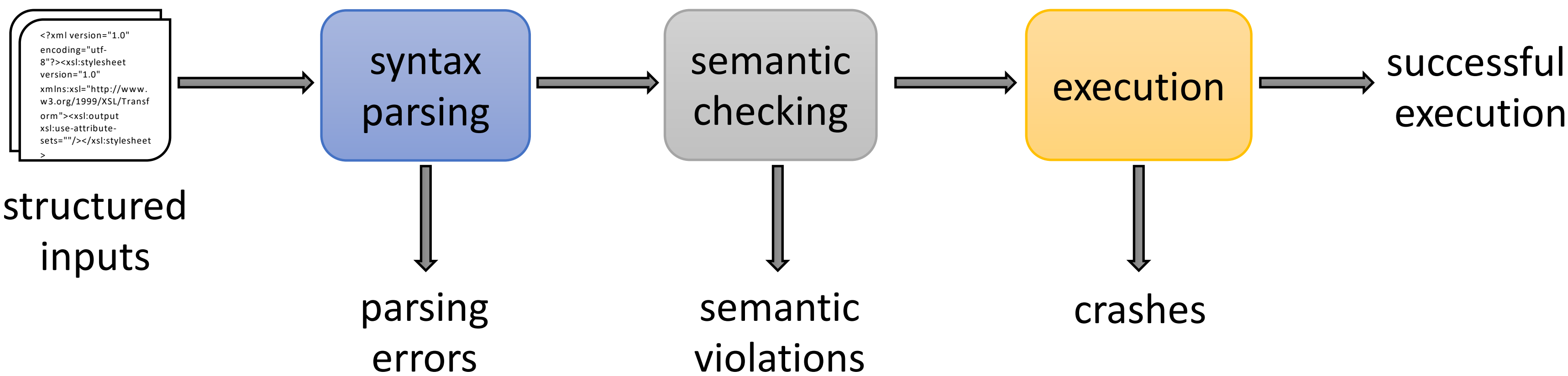
Mutation Based Fuzzing

- Inputs are generated by mutating existing inputs (e.g., bit flips)



- effective for unstructured input formats (e.g., images)
- less suitable for structured inputs (e.g., XSL)

Stages of Processing Structured Inputs



An Example of Semantic Checking in XSL

Attribute match cannot be applied on element xsl:copy; otherwise, an "unexpected attribute name" message will be prompted

<xsl:copy use-attribute-sets="name-list" match="*"></xsl:copy>

Generation Based Fuzzing

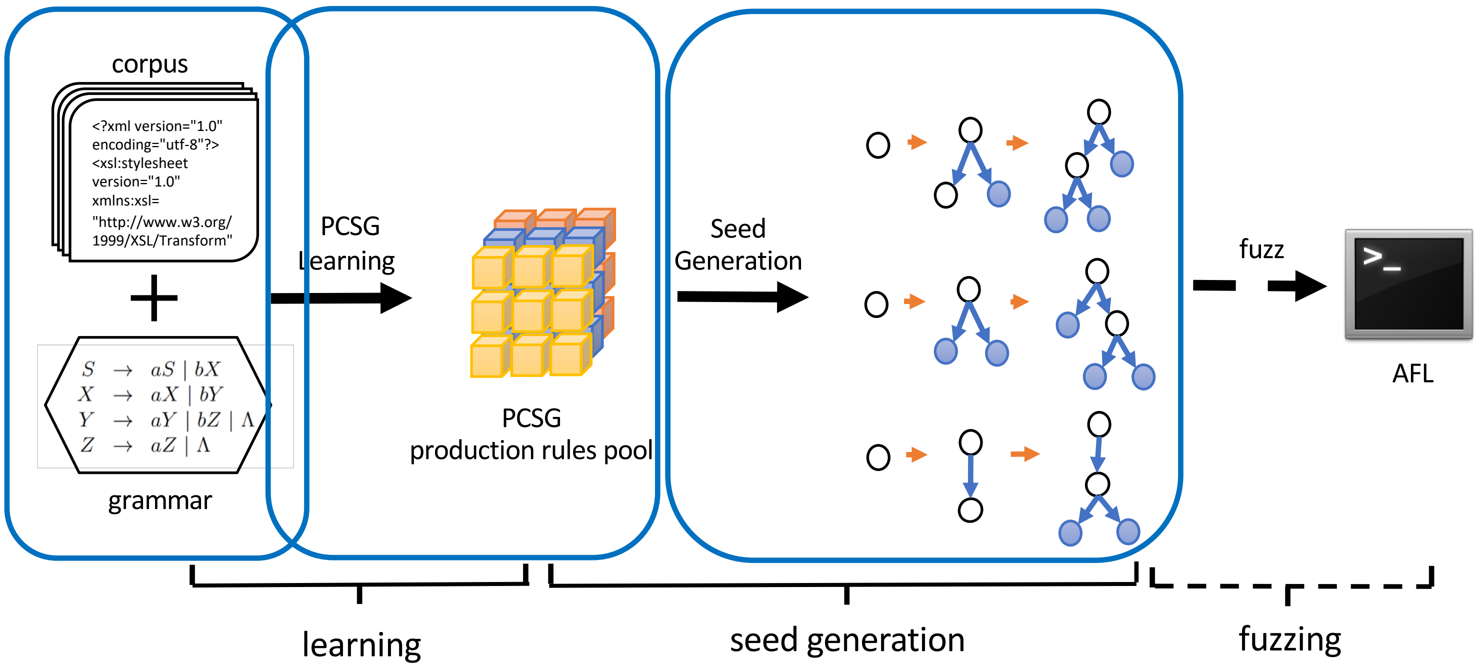
- Inputs are generated from scratch (e.g., following a grammar)

<table>
<tr><th></th><th>Grammar</th><th>Manually-Specified Generation Rules</th></tr>
<tr><td>syntax rules</td><td>easy</td><td rowspan="2">drawbacks
— different programs may implements different sets of semantic rules
— it is labor-intensive, or even impossible to list all semantic rules</td></tr>
<tr><td>semantic rules</td><td>hard</td></tr>
</table>

Skyfire: Data-Driven Seed Generation

- **Goal**: generate **well-distributed** seed inputs for fuzzing programs that process **structured inputs**
- **Solution**: leverage the vast amount of samples to automatically extract the knowledge of grammar and semantic rules

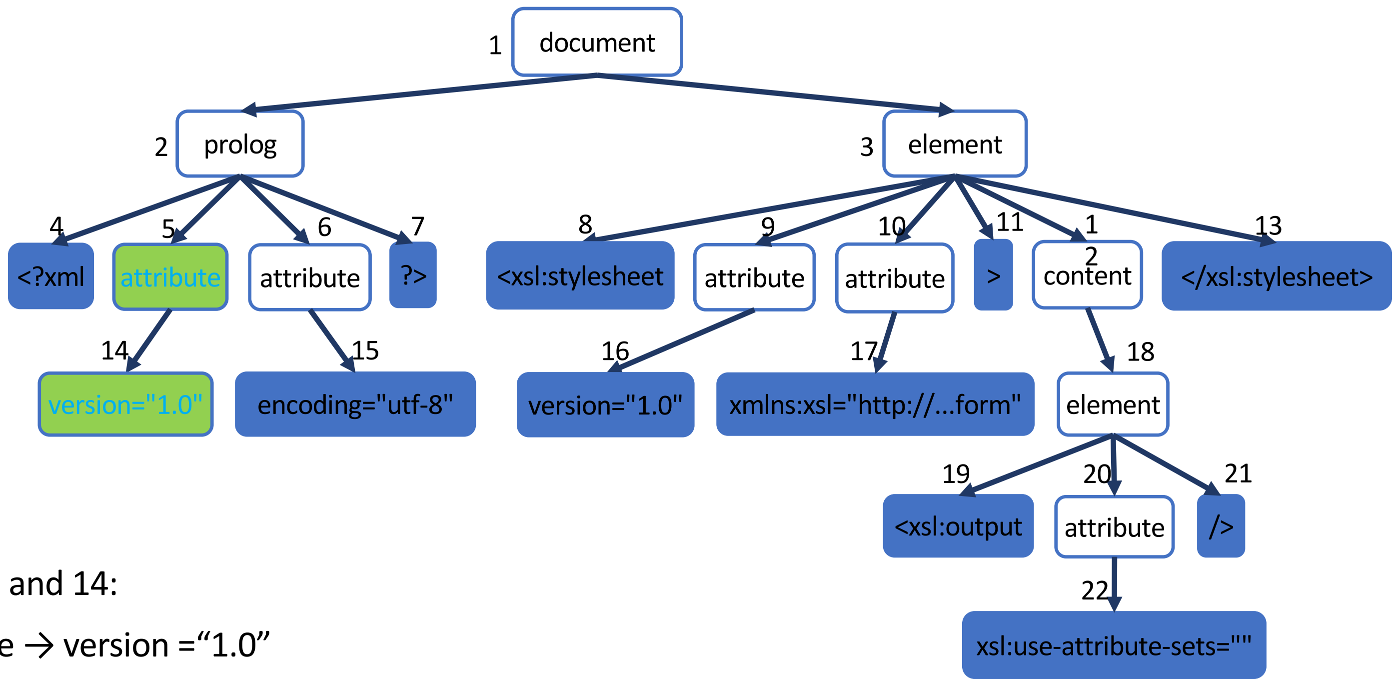
Skyfire: Data-Driven Seed Generation (cont.)



Context-Free Grammar

- **Context-Free Grammar (CFG)** $G_{cf}= (N, \Sigma, R, s)$
  - N is a finite set of non-terminal symbols
  - Σ is a finite set of terminal symbols
  - $s \in N$ is a distinguished start symbol
  - *R* is a finite set of production rules of the form $\alpha \rightarrow \beta_1\beta_2...\beta_n$, $\alpha \in N$, $n\geq 1$, $\beta_i \in (N \cup \Sigma)$ for $i = 1...n$

Example



nodes 5 and 14:

attribute → version ="1.0"

Semantic Rules

- Semantic rules determine whether a production rule can be applied on a non-terminal symbol, i.e., the application context of a rule

#	Error Messages of Violating Semantic Rules	Context
1.	XML declaration not well-formed	parent
2.	The root element that declares the document to be an XSL style sheet is xsl:stylesheet or xsl:transform	parent and first sibling
3.	Unexpected attribute {...}	first sibling
4.	Unbound prefix	first sibling
5.	XSL element xsl:stylesheet can only contain XSL elements	great-grandparent
6.	Required attribute {...} is missing	first sibling and all mandatory attributes
7.	Duplicate attribute	all siblings

Probabilistic Context-Sensitive Grammar

- **Context-Sensitive Grammar (CSG)** $G_{CS} = (N, \Sigma, R, s)$
  - $[c]\ \alpha \rightarrow \beta_1\beta_2\ldots\beta_n$
  - <type of α's great-grandparent, type of α's grandparent, type of α's parent, value of α's first sibling or type if the value is null>

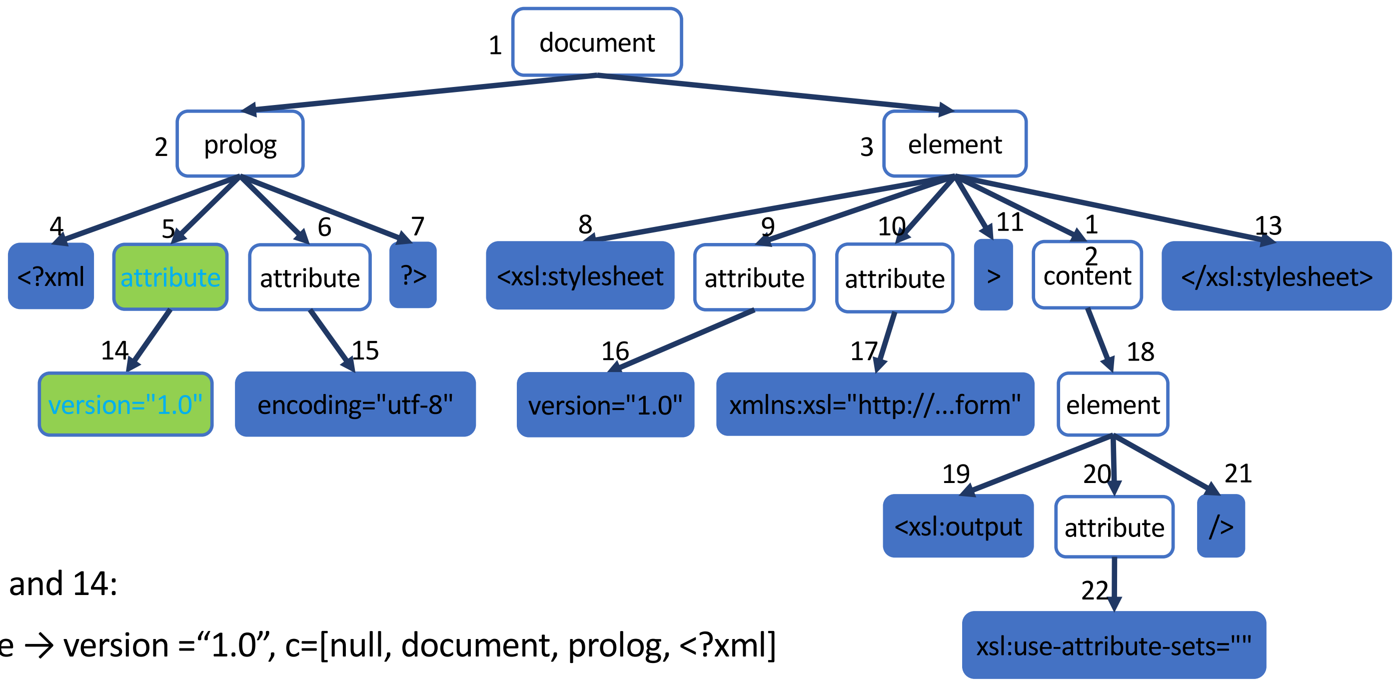
- **Probabilistic Context-Sensitive Grammar (PCSG)** $G_p = (G_{cs}, q)$
  - $q : R \rightarrow R+, \forall \alpha \in N : \sum_{[c]\alpha \rightarrow \beta 1\ \beta 2\ \ldots \beta n\ \in R} q([c]\alpha \rightarrow \beta 1\ \beta 2\ \ldots \beta n) = 1$

PCSG Learning from Corpus

- Parse code samples into parse trees
- Count the occurrence of each parent-children pair under a context
- Calculate the maximum likelihood estimation:

$$q([c]\alpha \rightarrow \beta_1 \beta_2 \ldots \beta_n) = \frac{\text{count}([c]\alpha \rightarrow \beta_1 \beta_2 \ldots \beta_n)}{\text{count}(\alpha)}$$

PCSG Learning from Corpus (cont.)



nodes 5 and 14:

attribute → version ="1.0", c=[null, document, prolog, <?xml]

Learned Production Rules of XSL

Context	Production rule		Prob.
[null,null,null,null]	document	→ prolog element	0.8200
		→ element	0.1800
[null,null,document,null]	prolog	→ <?xml attribute attribute?>	0.6460
		→ <?xml attribute?>	0.3470
		→ ...	
[null,null,document,prolog]	element	→ <xsl:stylesheet attribute attribute attribute>content</xsl:stylesheet>	0.0034
		→ <xsl:transform attribute attribute>content</xsl:transform>	0.0001
		→ ...	
[document,element,content,element]	element	→ <xsl:template attribute>content</xsl:template>	0.0282
		→ <xsl:variable attribute>content</xsl:variable>	0.0035
		→ <xsl:include attribute/>	0.0026
		→ ...	
[null,document,prolog,<?xml]	attribute	→ version="1.0"	0.0056
		→ encoding="utf-8"	0.0021
		→ ...	

Left-Most Derivation

t0=document → start with the start symbol, randomly choose a production rule whose left-side is document

t1=prolog element → choose a left-most non-terminal symbol , randomly choose a production rule whose left-side is prolog

t2=<?xml attribute attribute?> element

t3=<?xml version="1.0" attribute?> element

t4=<?xml version="1.0" encoding="utf-8"?>element

t5=<?xml version="1.0" encoding="utf-8"?><xsl:stylesheet attribute>content</xsl:stylesheet>

t6=<?xml version="1.0" encoding="utf-8"?><xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">content</xsl:stylesheet>

t7=<?xml version="1.0" encoding="utf-8"?><xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">element</xsl:stylesheet>

t8=<?xml version="1.0" encoding="utf-8"?><xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"><xsl:output attribute/></xsl:stylesheet>

t9=<?xml version="1.0" encoding="utf-8"?><xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"><xsl:output xsl:use-attribute-sets=""/></xsl:stylesheet>

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:output xsl:use-attribute-sets=""/>
</xsl:stylesheet>
```

Heuristic-Based Left-Most Derivation

- Heuristic Rules
  - Satisfy context
  - Favor low-probability production rules
  - Restrict the application number of the same production rule
  - Favor low-complexity production rules
  - Restrict the total number of rule applications

Experiment Setup - Samples

Language	XSL	XML
number of unique samples crawled	18,686	19,324
number of distinct samples crawled (afl-cmin)	671	732
number of distinct seeds generated by Skyfire (afl-cmin)	5,017	5,923

Experiment Setup – Target Programs

Sablotron (XSL engine)

- Adobe PDF Reader, and Acrobat

Libxslt (XSL engine)

- Chrome browser, Safari browser, and PHP 5

Libxml2 (XML engine)

- Linux, Apple iOS/OS X, and tvOS

Experiment Setup - Approaches

Crawl

- samples crawled

Skyfire

- inputs generated by Skyfire

Crawl+AFL

- feed the samples crawled as seeds to AFL

Skyfire+AFL

- feed the inputs generated by Skyfire as seeds to AFL

Bugs Found in XSL and XML Engines

Unique Bugs (#)	XSL						XML		
	Sablotron 1.0.3			libxslt 1.1.29			libxml2 2.9.2/2.9.3/2.9.4		
	Crawl+AFL	Skyfire	Skyfire+AFL	Crawl+AFL	Skyfire	Skyfire+AFL	Crawl+AFL	Skyfire	Skyfire+AFL
Memory Corruptions (New)	1	5	8§	0	0	0	6	3	11¶
Memory Corruptions (Known)	0	1	2†	0	0	0	4	0	4‡
Denial of Service(New)	8	7	15	0	2	3	2	1	3⊕
Total	9	13	25	0	2	3	12	4	18

§ CVE-2016-6969, CVE-2016-6978, CVE-2017-2949, CVE-2017-2970, and one pending report.
¶ CVE-2015-7115, CVE-2015-7116, CVE-2016-1835, CVE-2016-1836, CVE-2016-1837, CVE-2016-1762, and CVE-2016-4447; pending reports include GNOME bugzilla 766956, 769185, 769186, and 769187.
†CVE-2012-1530, CVE-2012-1525.
‡CVE-2015-7497, CVE-2015-7941, CVE-2016-1839, and CVE-2016-2073.
⊕GNOME bugzilla 759579, 759495, and 759675.

19 new memory corruptions bugs (16 vulnerabilities, 11 CVEs, and 33.5K USD)
21 new denial of service bugs

Line and Function Coverage

program			line coverage (%)				function coverage (%)			
name	lines	functions	crawl	crawl+AFL	Skyfire	Skyfire+AFL	crawl	crawl+AFL	Skyfire	Skyfire+AFL
Sablotron 1.0.3	10,561	2,230	34.0	39.0	**65.2**	**69.8**	29.8	32.6	**48.1**	**50.1**
libxslt 1.1.29	14,418	778	29.6	38.1	**57.4**	**62.5**	30.0	34.2	**51.9**	**53.1**
libxml2 2.9.4	67,420	3,235	13.5	15.3	**22.0**	**23.8**	15.7	16.3	**24.1**	**25.9**

20%/15% line/function coverage improvement

Effectiveness of Context

Percentage of generated inputs that pass semantic checking

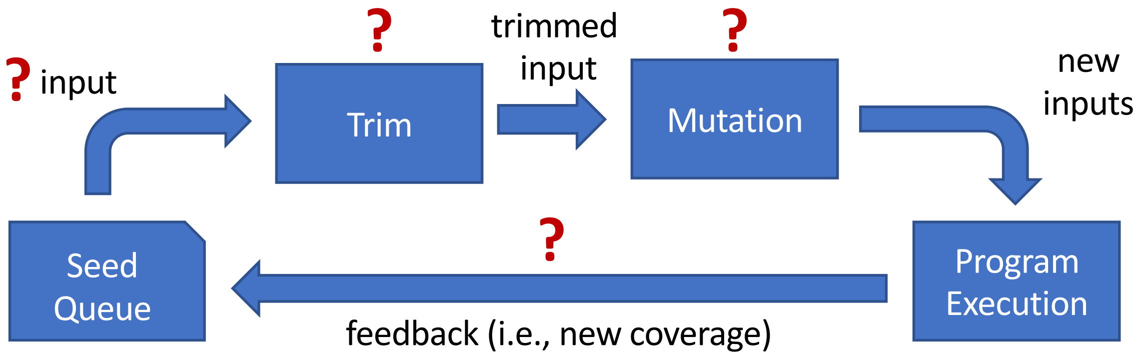
Approach	XSL	XML
CFG-Based	0	34
PCSG-Based	85	63

Performance Evaluation

Time	XSL	XML
Learning (h)	1.5	1.6
Generation (s)	20.3	20.6

Conclusions

- Data-driven seed generation approach to generate well-distributed seed inputs for fuzzing programs that process structured inputs



Reading Materials

- J. Wang, B. Chen, L. Wei, and Y. Liu, “Skyfire: Data-driven seed generation for fuzzing,” in SP, 2017, pp. 579–594.
- C. Holler, K. Herzig, and A. Zeller, “Fuzzing with code fragments,” in USENIX Security, 2012, pp. 445–458.
- S. Veggalam, S. Rawat, I. Haller, and H. Bos, “Ifuzzer: An evolutionary interpreter fuzzer using genetic programming,” in ESORICS, 2016, pp. 581–601.
- Y. Li, B. Chen, M. Chandramohan, S.-W. Lin, Y. Liu, and A. Tiu, “Steelix: Program-state based binary fuzzing,” in ESEC/FSE, 2017, pp. 627–637.
- S. Rawat, V. Jain, A. Kumar, L. Cojocar, C. Giuffrida, and H. Bos, “Vuzzer: Application-aware evolutionary fuzzing,” in NDSS, 2017.
- N. Stephens, J. Grosen, C. Salls, A. Dutcher, R. Wang, J. Corbetta, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, “Driller: Augmenting fuzzing through selective symbolic execution,” in NDSS, 2016.
- M. Bö hme, V.-T. Pham, M.-D. Nguyen, and A. Roychoudhury, “Directed greybox fuzzing,” in CCS, 2017.
- H. Chen, Y. Xue, Y. Li, B. Chen, X. Xie, X. Wu, and Y. Liu, “Hawkeye: Towards a desired directed grey-box fuzzing,” in CCS, 2018.
- M. Bö hme, V.-T. Pham, and A. Roychoudhury, “Coverage-based greybox fuzzing as markov chain,” in CCS, 2016, pp. 1032–1043.

Q&A?

Bihuan Chen, Pre-Tenure Assoc. Prof.

bhchen@fudan.edu.cn

https://chenbihuan.github.io