Skyfire: Data-Driven Seed Generation for Fuzzing

Junjie Wang, Bihuan Chen, Lei Wei, and Yang Liu
Nanyang Technological University
wang1043, bhchen, l.wei, yangliu@ntu.edu.sg

Agenda

Fuzzing / motivation

Skyfire: data-driven seed generation

Application of Skyfire

Case study / future work

Fuzzing

Fuzzing is an automated software testing technique

- provide invalid and unexpected inputs to programs
- then monitor for crashes, failing assertions or potential memory leaks

Fuzzing approaches

Generation-based fuzzing

– inputs are generated from scratch

Mutation-based fuzzing

– inputs are generated by modifying existing inputs

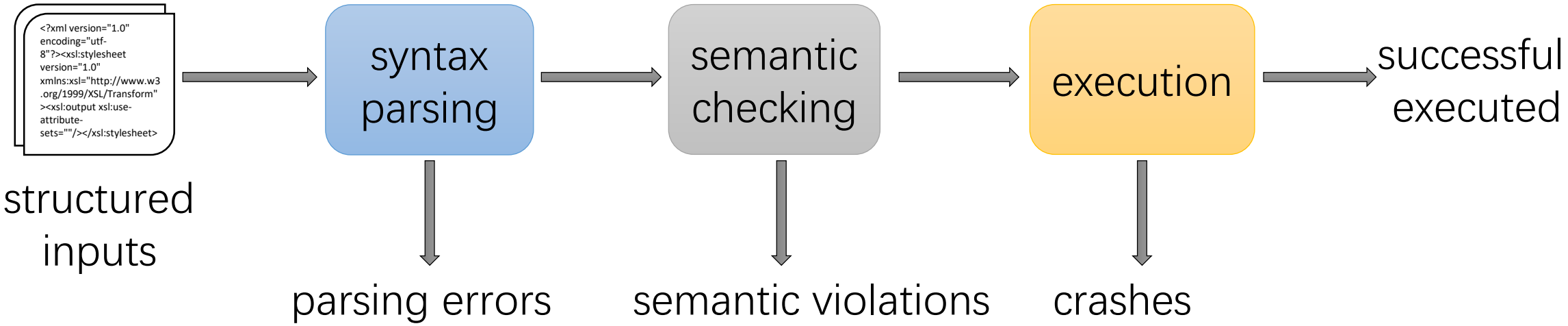
Dumb

– unaware of input structure

Smart

– aware of input structure

Stages of processing structured inputs



Passing syntax parsing/semantic checking

	Grammar	**manually-specified generation rules**
syntax rules	**easy**	**drawbacks:** — different programs may implements different sets of semantic rules, — labor-intensive, or even impossible
semantic rules	**hard**	

An example of semantic checking

Attribute “match” cannot be applied on element “xsl:copy”

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

– prompts “unexpected attribute name” message

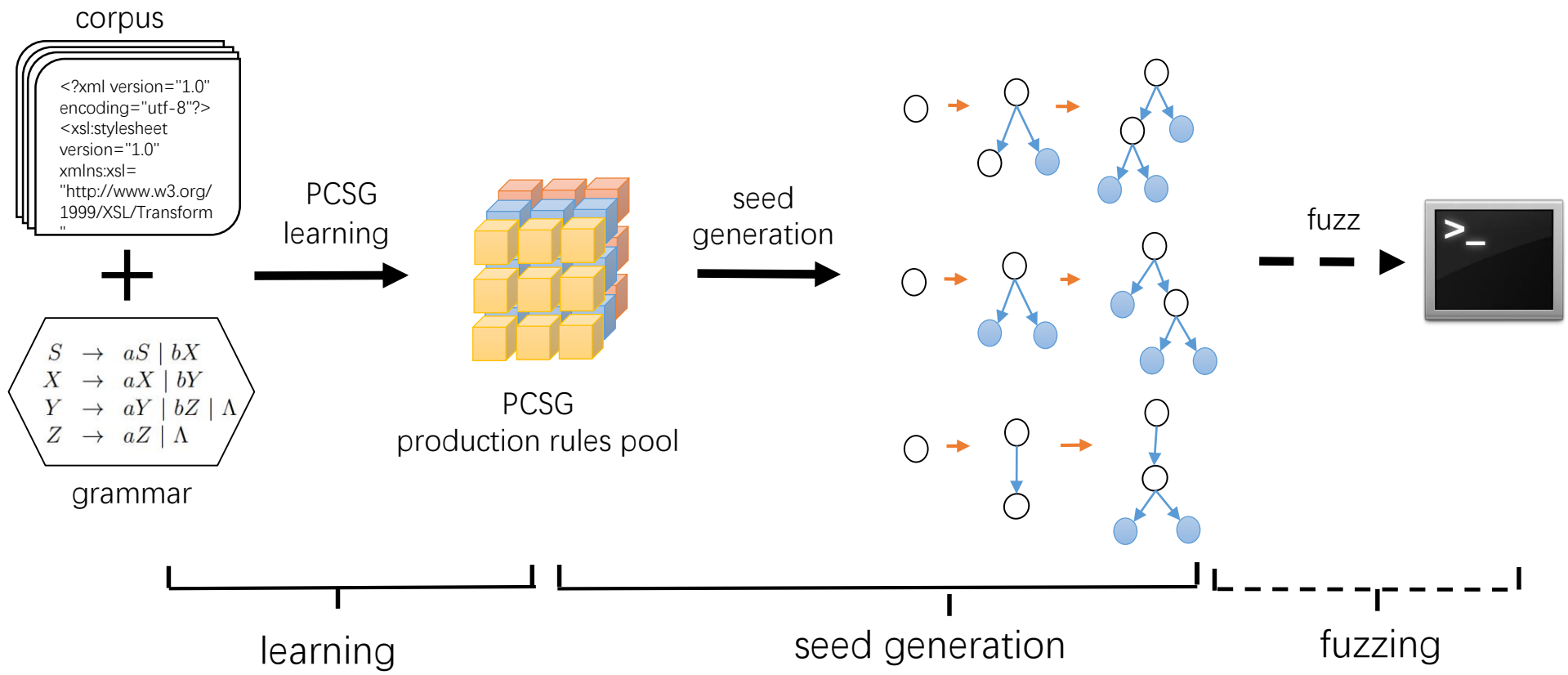
– rejected

An effective fuzzer should be able to generate inputs most of them can reach execution stage.

Skyfire: data-driven seed generation

- Extend context-free grammar (CFG) to model semantic rules
- Target programs processing highly-structured inputs
- Design goal: generates well-distributed test cases

Skyfire: data-driven seed generation



Probabilistic context-sensitive 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 $\alpha \rightarrow \beta_1\beta_2...\beta_n$, $\alpha \in N$, $n \geqslant 1$, $\beta_i \in (N \cup \Sigma)$ for i = 1...n,

Context-sensitive grammar(CSG) $G_{CS} = (N, \Sigma, R, s)$:

- $[c]\alpha \rightarrow \beta_1\beta_2...\beta_n$,

<type of α 's great-grandparent, type of α 's grandparent, type of α 's parent, value of α 's first sibling or type the value is null>

Probabilistic context-sensitive grammar(PCSG) : $G_p = (G_{cs}, q)$,

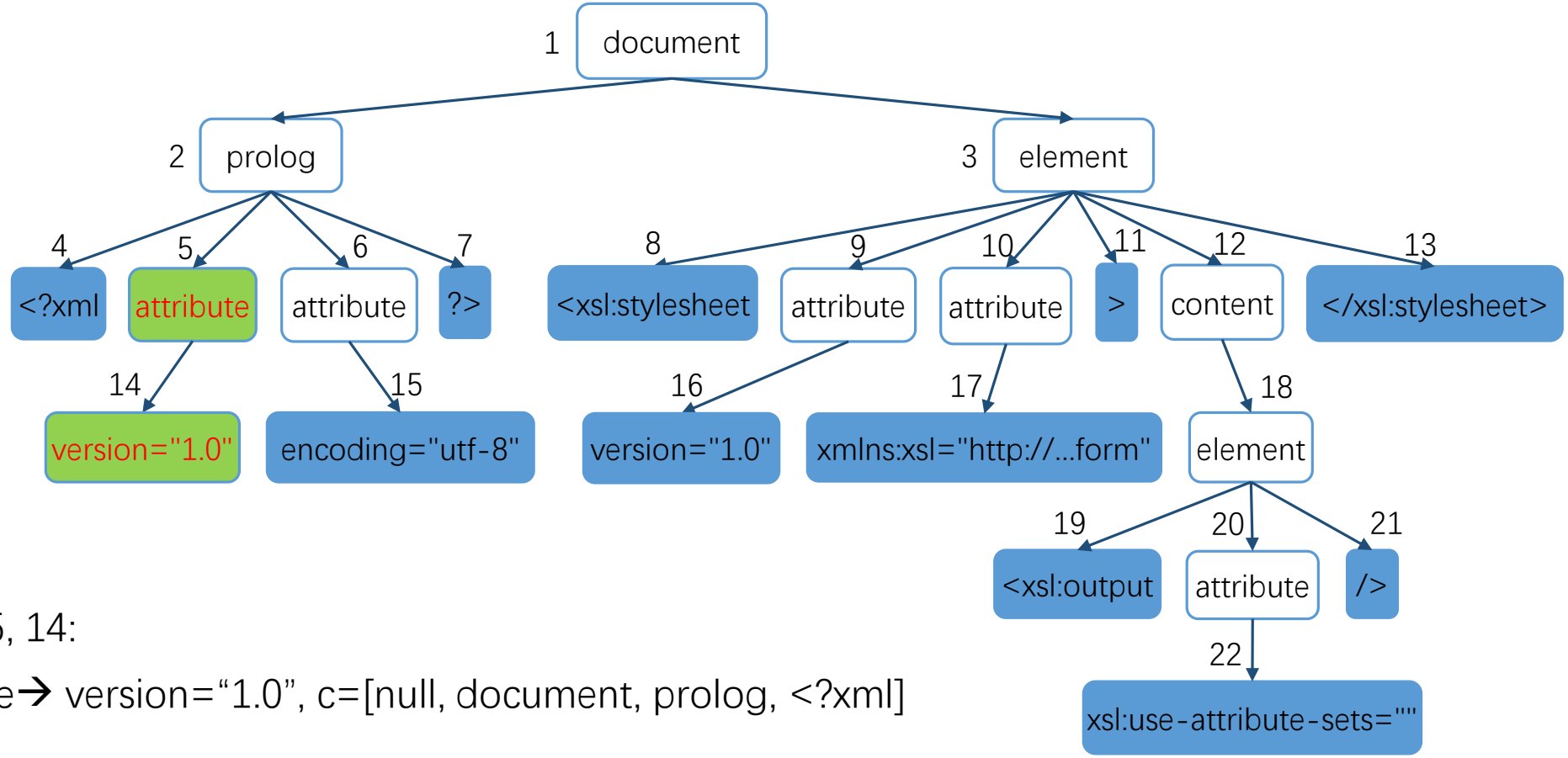
$$q : R \rightarrow R+, \forall \alpha \in N : \sum_{[c]\alpha \rightarrow \beta 1\ \beta 2\ ...\beta n\ \in R} q([c]\alpha \rightarrow \beta 1\ \beta 2\ ...\beta n\ ) = 1.$$

PCSG learning from corpus

- parse code samples to parse trees,
- count occurrence of each parent-children pair and the 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



nodes 5, 14:

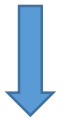
attribute→ version=“1.0”, c=[null, document, prolog, <?xml]

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
t1=prolog element
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>
```

Seed generation

Input: the PCSG Gp = ((N, Σ, R, s), q)
Output: the set of seeds generated T
T := ∅
repeat
  t := s // set the seed input to the start symbol
  c := null // set the context to null
  num := 0 // set the times of applying rules to 0
  repeat
    l := the left-most non-terminal symbol in t
    update c according to l
    R_l:= the set of rules in R whose left-side is l given c
    if random() < 0.9 **then**
      heuristically choose a less-frequently applied and less complexity rule r from low-probability rules in R_l
    else heuristically choose a less-frequently applied and less complexity rule r from high-probability rules in R_l
    end if
    replace l in t with the right side of r
    num := num + 1
  until there is no non-terminal symbol in t , or num == 200
  T := T ∪ {t}
until time budget is reached, or enough seed inputs are generate

Seed generation

Heuristic rules:

- 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

Heritrix to collect samples and establish the corpus.

Language	XSL	XML
number of unique samples crawled	18,686	19,324
number of distinct samples crawled (afl-cmin)	671	732
number of unique seeds generated by Skyfire	20,000	20,000
number of distinct seeds generated by Skyfire (afl-cmin)	5,017	5,923

ANTLR to automatically generate the corresponding lexer and parser.

Experiment setup

Sablotron

- Adobe PDF Reader and Acrobat.

libxslt

- Chrome browser, Safari browser, and PHP 5

libxml2

- Linux, Apple iOS/OS X, and tvOS

Experiment setup

Crawl
- samples crawled

Skyfire
- inputs generated by Skyfire

Crawl+AFL
- Fed the samples crawled as seeds to AFL

Skyfire+AFL
- the inputs generated by Skyfire as seeds to AFL

Bugs found in XSLT and XML/HTML engines

Unique Bugs (#)	XSL						XML/HTML		
	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.

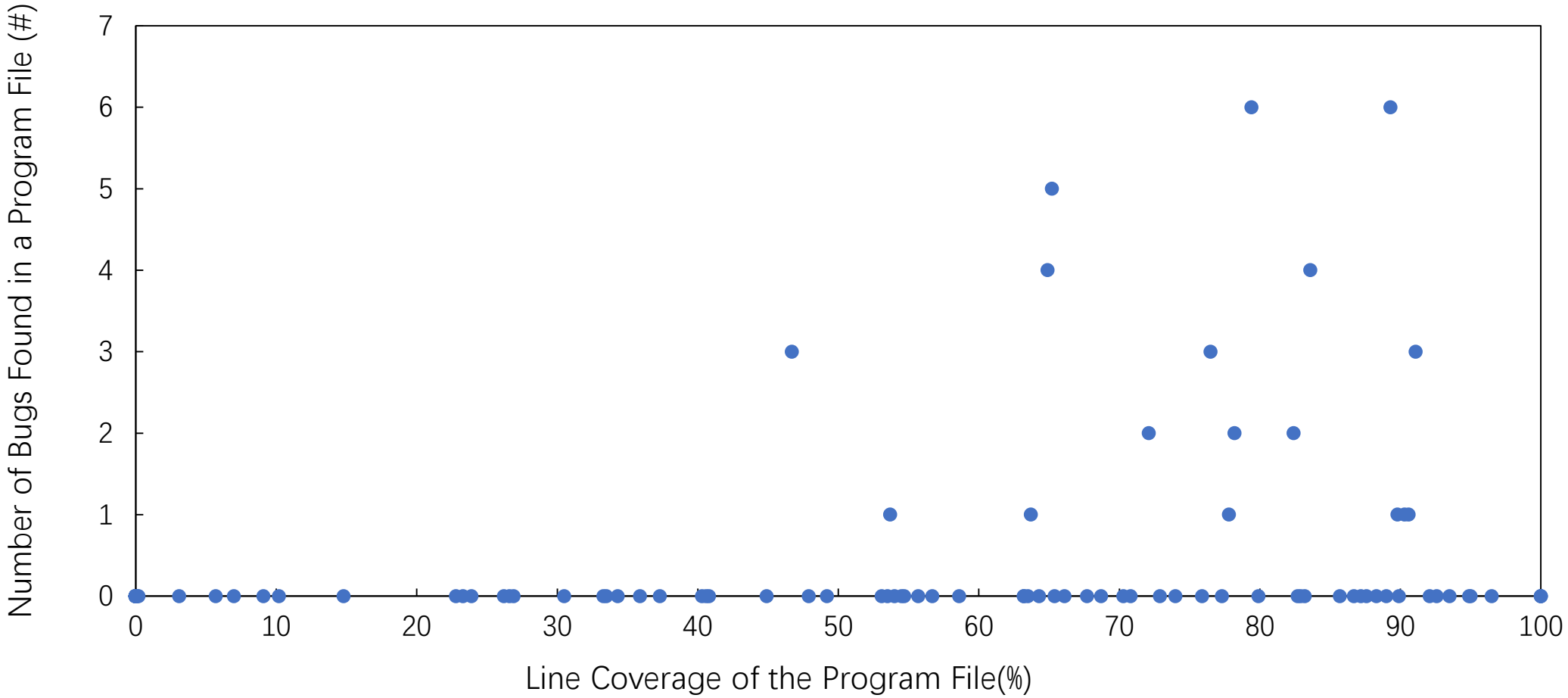
Vulnerabilities and Types

Vulnerability	Type
CVE-2016-6978	Out-of-bound read
CVE-2016-6969	Use-after-free
Pending advisory 1	Double-free / UAF
CVE-2017-2949	Out-of-bound write
CVE-2017-2970	Out-of-bound write
CVE-2015-7115	Out-of-bound read
CVE-2015-7116	Out-of-bound read
CVE-2016-1762	Out-of-bound read
CVE-2016-1835	Use-after-free
CVE-2016-1836	Use-after-free
CVE-2016-1837	Use-after-free
CVE-2016-4447	Out-of-bound read
Pending advisory 2	Out-of-bound read
Pending advisory 3	Out-of-bound read
Pending advisory 4	Use-after-free
Pending advisory 5	Out-of-bound read

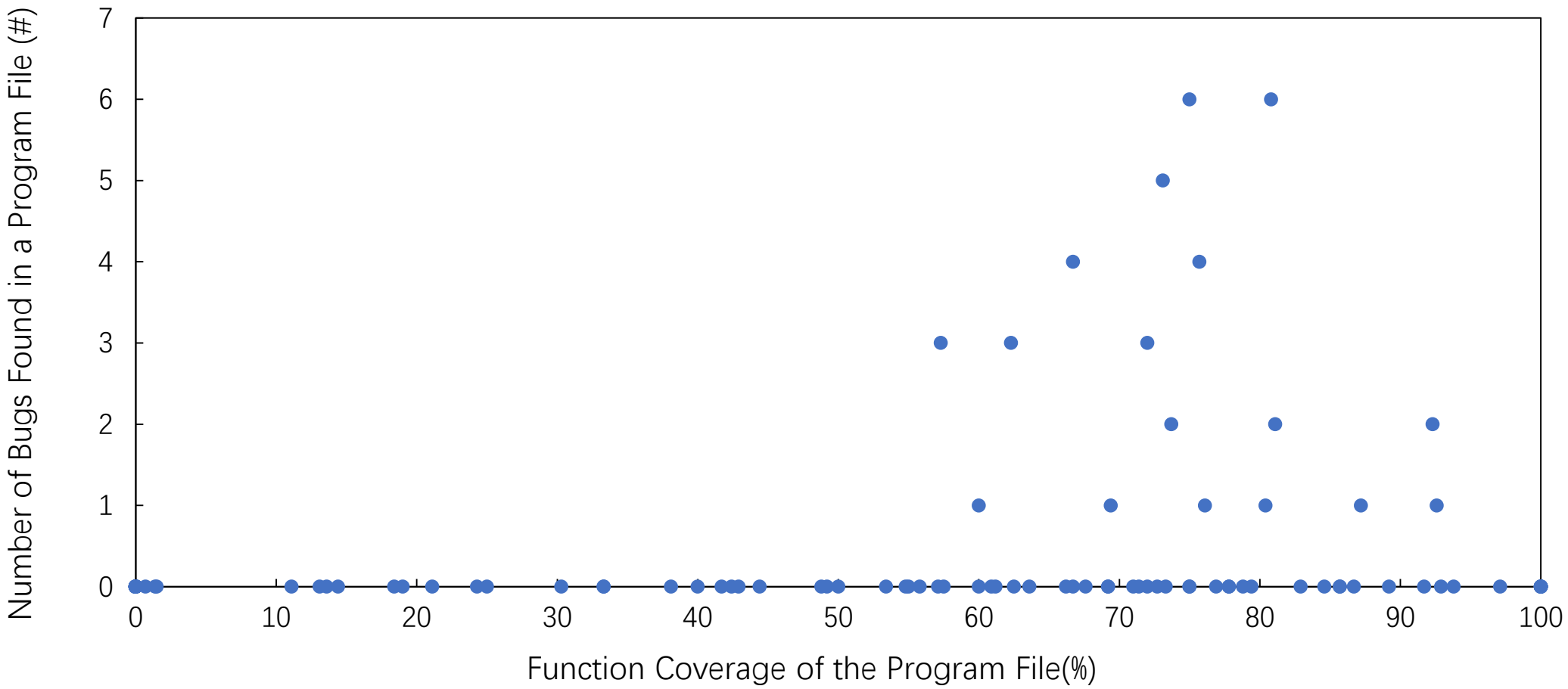
Line and function coverage

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**

Line coverage vs. the number of bugs



Function coverage vs. the number of bugs



Case study-CVE-2016-6978

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0“ xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <xsl:value-of select="substring-before('F','-')"/>
    </xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0“ xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <xsl:value-of select="substring-before('F')"/>
    </xsl:template>
</xsl:stylesheet>
```

Case study

```
934  case EXFF_SUBSTRING_BEFORE:
935  case EXFF_SUBSTRING_AFTER:
936  {
937          Str strg;
938          Str theBigger, theSmaller;
939          E( atoms[ 0 ] -> tostring( S, theBigger ) );
940          E( atoms[ 1 ] -> tostring( S, theSmaller ) );
941          checkArgsCount( 2 );
942          checkIsString2( 0, 1 );
943          int where = firstOccurence( theBigger, theSmaller );
944          if ( where == -1 ) strg.empty();
945          else{
946                  if ( functor == EXFF_SUBSTRING_BEFORE ){
947                          if ( where == 0 ) strg.empty();
948                          else getBetween( strg, theBigger, 0, where - 1 );
949                  }
950                  else getBetween( strg, theBigger,
951                          where + utf8StrLength( theSmaller ), -1 );
952          };
953          retxpr.setAtom( strg );
954  }; break;
```

JavaScript / future work

JavaScript is more complicated than XML/XSL.

IE

- 11 null pointers

Webkit

- 7 unique crashes / 2 bug bounty

Thanks

Questions?