With a Case Study on Binary Vulnerability Analysis

Cyber-security: the Journey from Formal Methods, Program Analysis to Data Analytics

LIU, Yang

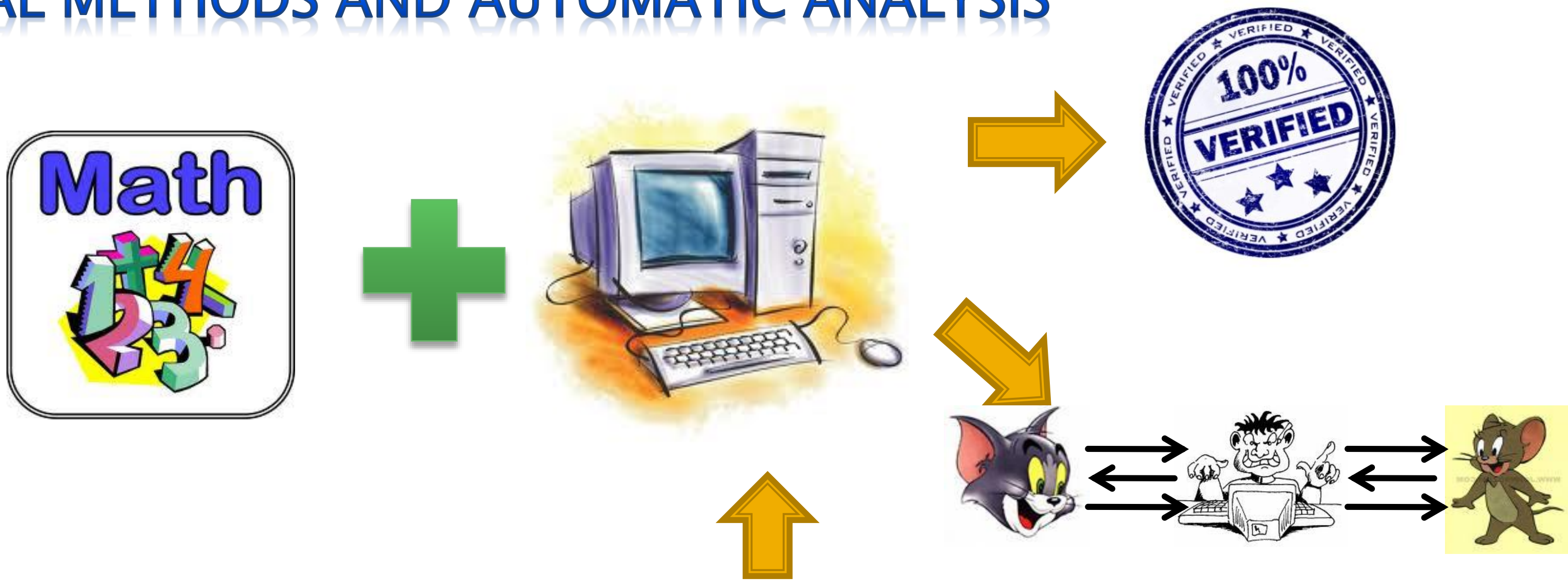
ICECCS 2017 (8 Nov 2017)

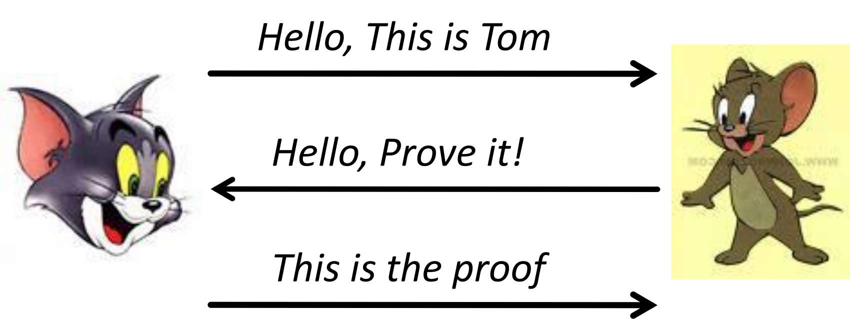
Since 2005

Formal Methods

Formal Analysis

- FORMAL METHODS AND AUTOMATIC ANALYSIS

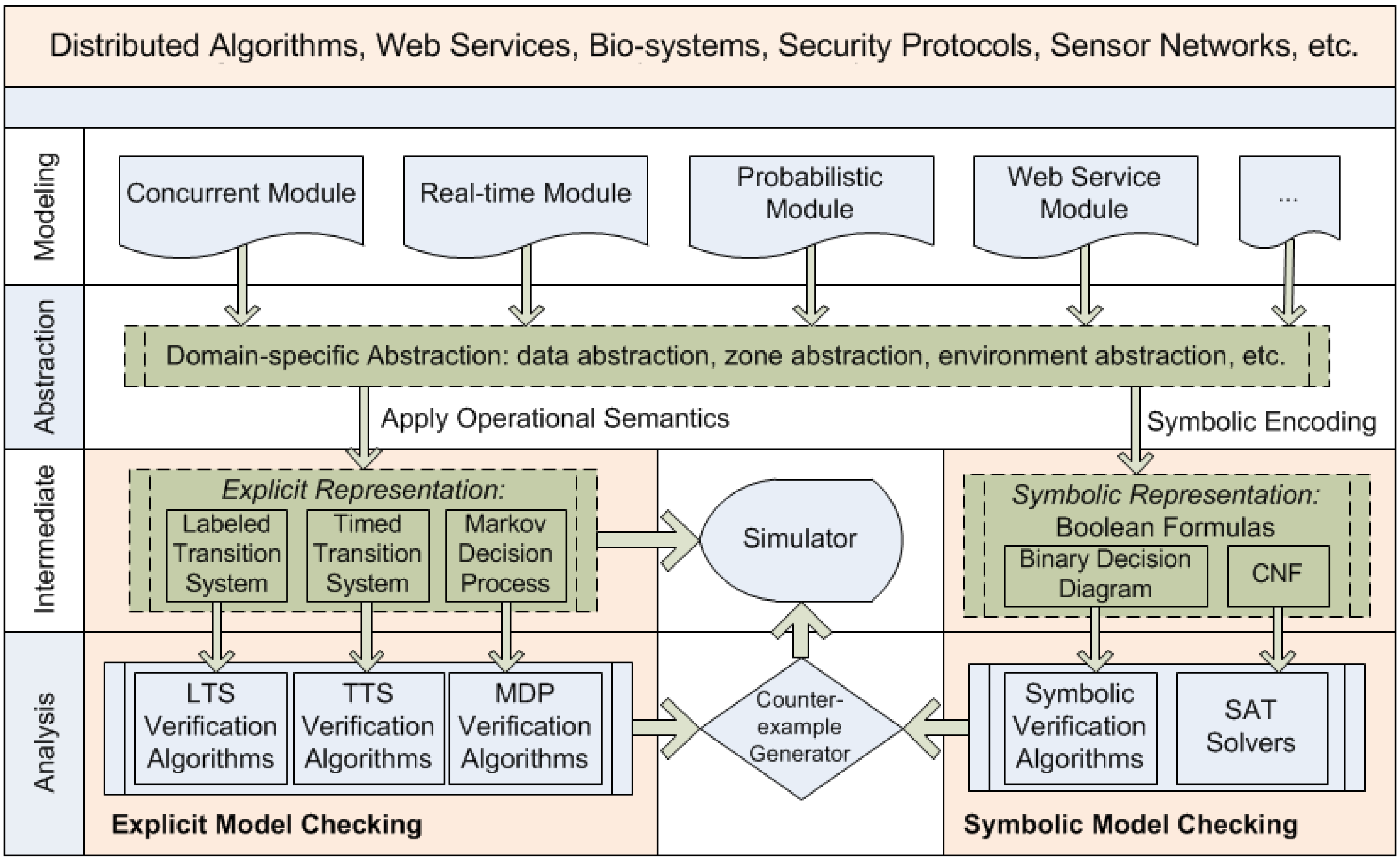




$$
\begin{array}{llcllcl}
\alpha 1. & A & \rightarrow & I & : & A.I.\{N_a.A\}_{K_I} \\
\beta 1. & I(A) & \rightarrow & B & : & A.B.\{N_a.A\}_{K_B} \\
\beta 2. & B & \rightarrow & I(A) & : & B.A.\{N_a.N_b\}_{K_A} \\
\alpha 2. & I & \rightarrow & A & : & I.A.\{N_a.N_b\}_{K_A} \\
\alpha 3. & A & \rightarrow & I & : & A.I.\{N_b\}_{K_I} \\
\beta 3. & I(A) & \rightarrow & B & : & A.B.\{N_b\}_{K_B}
\end{array}
$$

PAT (Process Analysis Toolkit), 2007

- A self-contained framework to support the development of formal verification tools



Applying Formal Methods in Cyber-security Systems (since 2012)

- Design verification
  - (Timed) Security Protocol (FM15,TSE17)
  - TPM Verification (FM14)
- Implementation verification
  - Authentication Protocols (NDSS13)
  - Android App for Malware (TSE17)
- Assembly code verification
  - Vulnerability Verification

Design

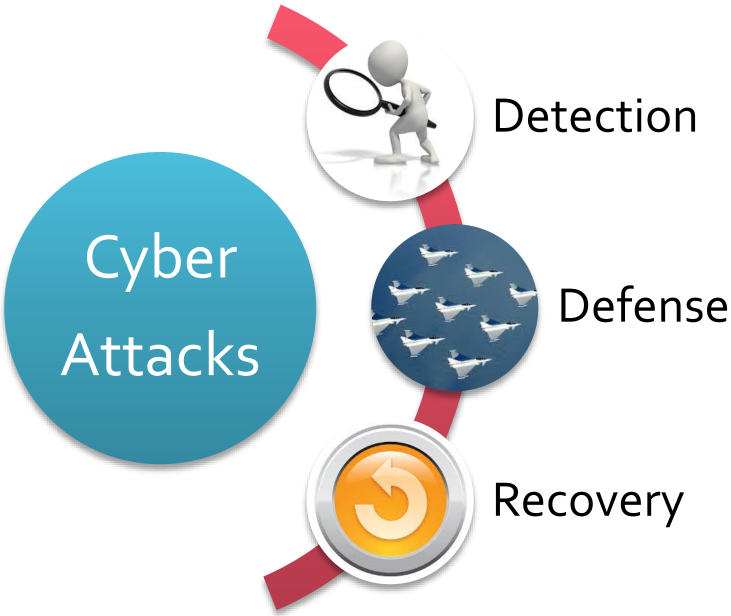
Source Code

Machine Code

Since 2015

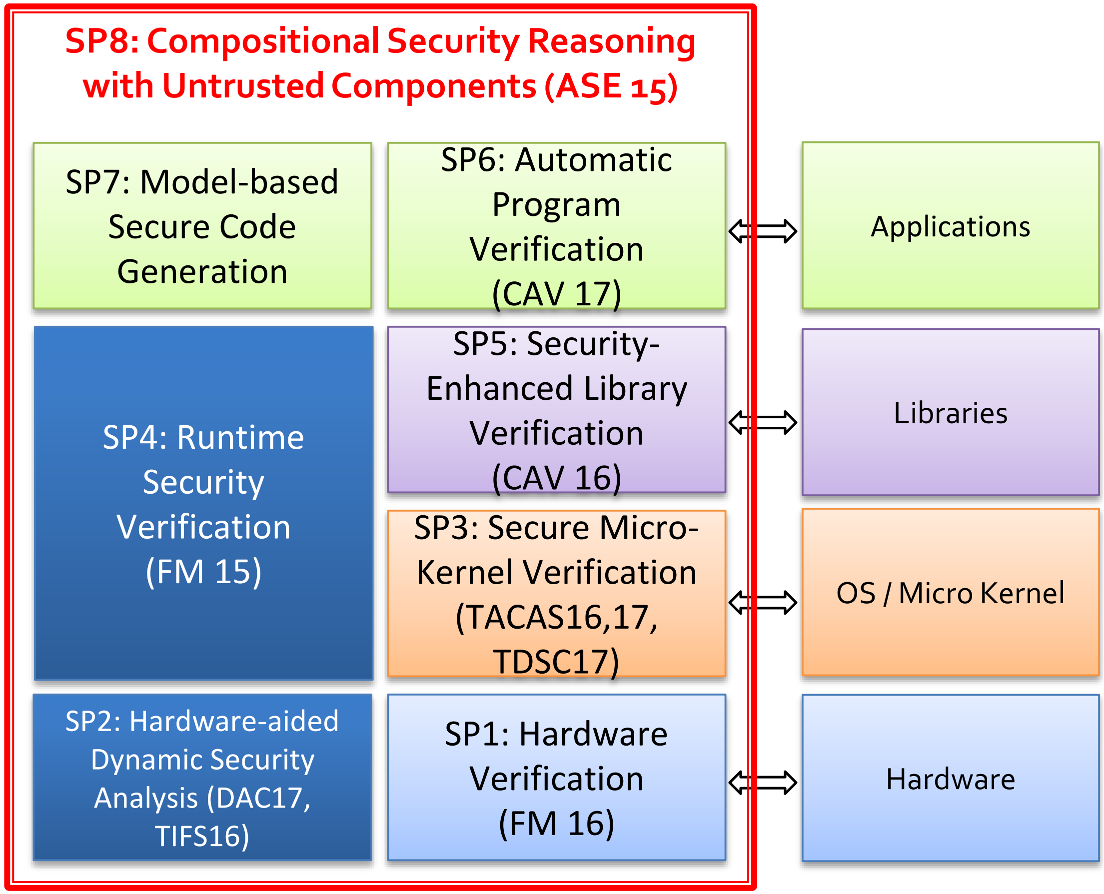
Securify: A Compositional Approach of Building Security Verified System

Introduction

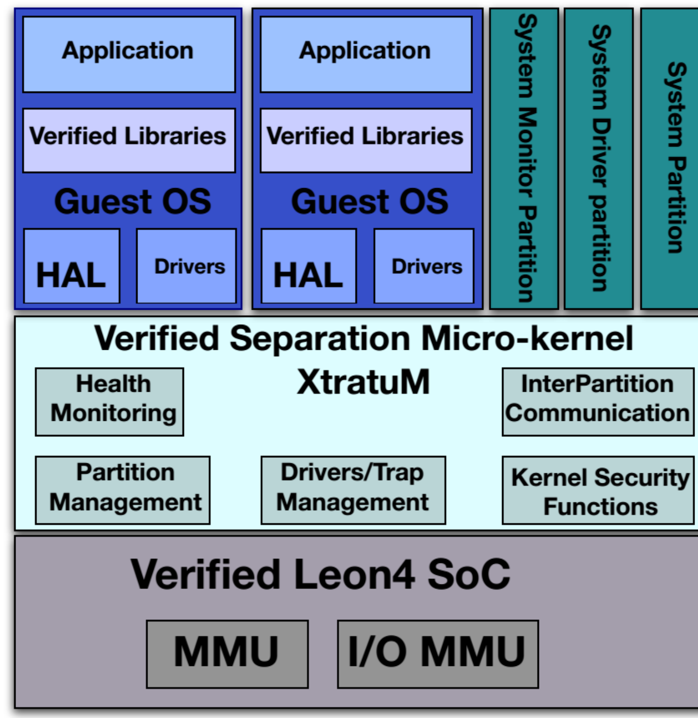


- **Verification**
  - To overcome security issues by deploying composition based techniques to realize security-verified systems
  - To introduce runtime verification to further improve robustness

Proposed Approach



Securify Architecture



Team Members

SP8: Compositional Security Reasoning with Untrusted Components

SP7: Model-based Secure Code Generation	SP6: Automatic Program Verification
SP4: Runtime Security Verification	SP5: Security-Enhanced Library Verification
	SP3: Secure Micro-Kernel Verification
SP2: Hardware-aided Dynamic Security Analysis	SP1: Hardware Verification

ST Electronics
Info-Security

WINCOR NIXDORF
EXPERIENCE MEETS VISION.

Srikanthan Thambipillai
NTU

Ralf Huuck
NICTA
SP6,8

Liu Yang
NTU

David Basin
ETH
SP4

Alwen Tiu
NTU

Kenny Paterson
RHUL
SP5

Chin Wei Ngan
NUS

Sjouke Mauw
Uni. of Lux.
SP3,7

Sun Jun
SUTD

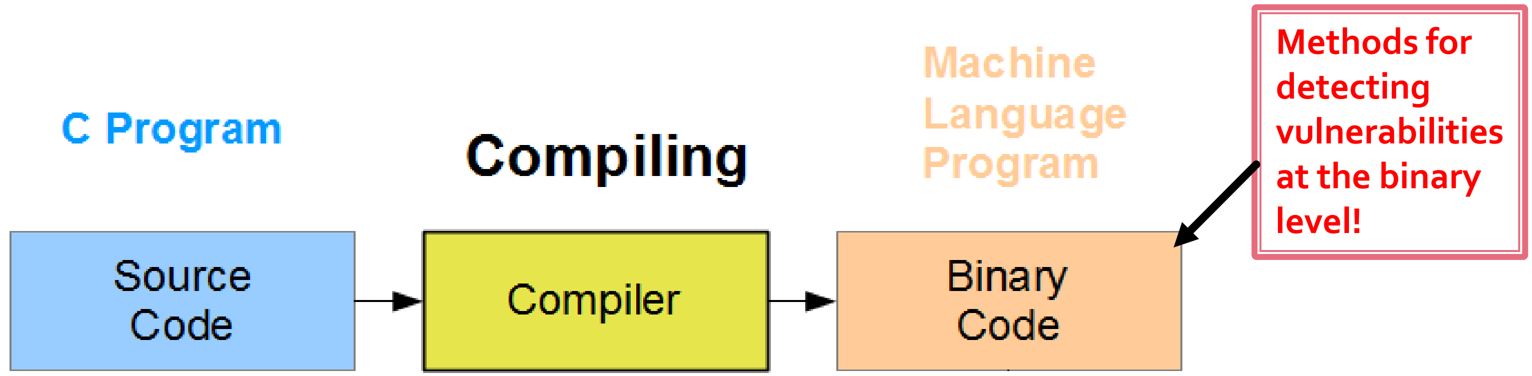
Luke Ong
Oxford
SP3,5

Dong Jin Song
NUS

Wei Zhang
HKUST
SP1,2

Is FM really useful for solving real-world security problems?

Case Study: Binary Vulnerability Detection?



- No need to rely on compilation process
- Vendors need not release source code
- Cross-architecture and cross platform analysis

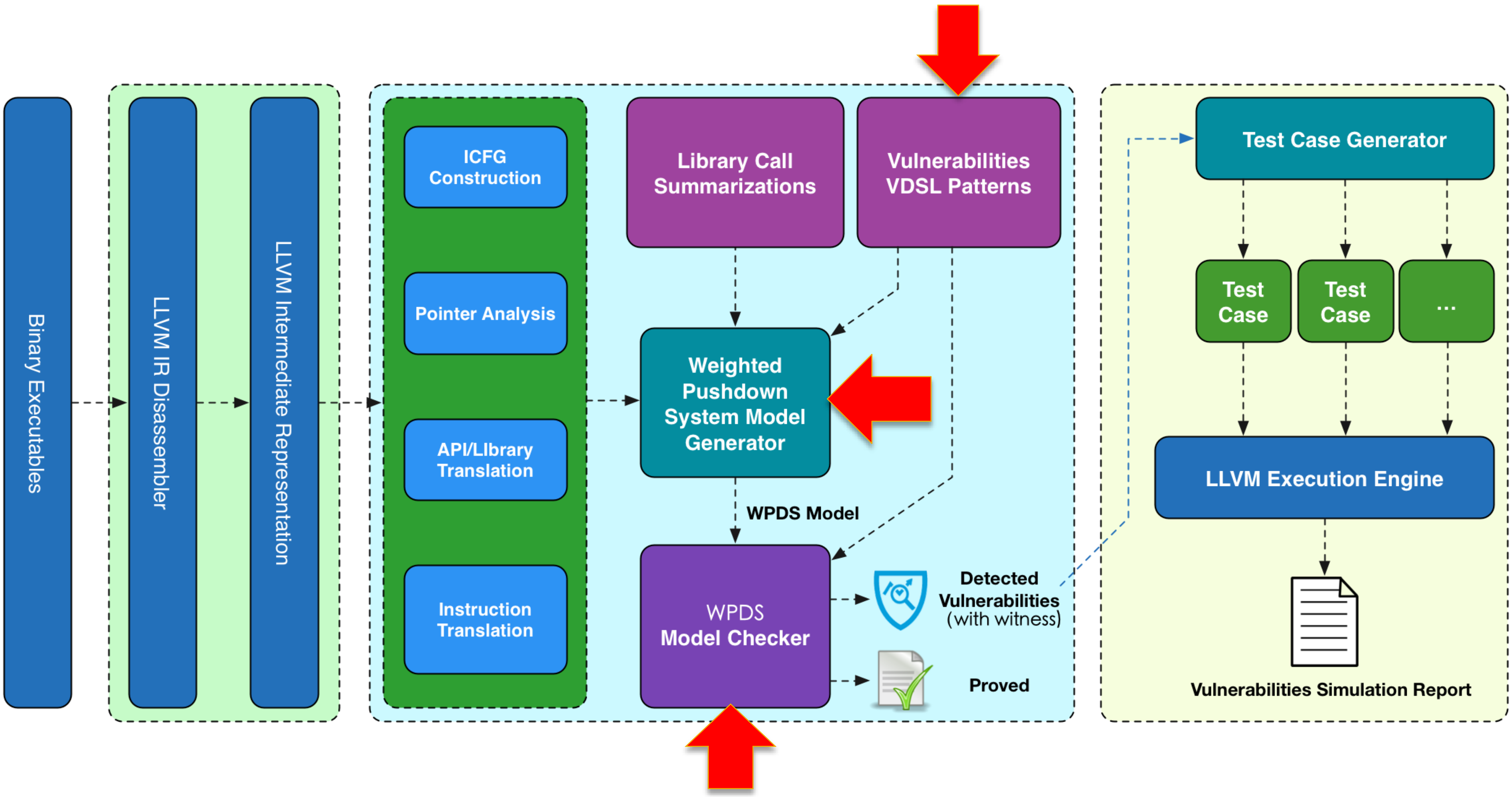
intel ARM Microsoft Windows Linux android

LLVV: Automatic Vulnerability Verification for LLVM's Intermediate Representation, 2014

Key Insights

- Detection of vulnerabilities sometimes relies upon partial semantics of variables rather than their values.
- Examples:
  - strcpy(des, src) → Len(src) + 1 > size(des)
  - memcpy(des,src, num) → value(num) > size(des)
- LLVV: Automatic Vulnerability Verification on LLVM IR
  - Context- and path-sensitive
  - Language independent

Approach Overview



VDSL (Vulnerability Domain-Specific Language)

Let **Attr** = $\{Size, Len, Val, Type\}$ denote size, length, value and type attributes. Other demanded attributes can be easily introduced. *VDSL* formulas are defined by the following syntax:

$$
\begin{aligned}
\phi &::= f(\$1, ..., \$k) \wedge \psi \\
\psi &::= exp \bowtie exp \ | \ \psi \wedge \psi \ | \ \psi \vee \psi \\
\bowtie &::= \leq \ | \ \geq \ | \ < \ | \ > \ | \ == \ | \ \neq \\
exp &::= attr(\$i) \ | \ i \ | \ exp \ \mathbf{op} \ exp \\
\mathbf{op} &::= + \ | \ - \ | \ \times \ | \ / \ | \ \%
\end{aligned}
$$

where f is a function, i, k are integers, $attr \in$ **Attr**.

```
int main(int argc, char **argv){
        char a[3]={'\0','\0','\0'};
        if( argc>1)
                f(a, argv[1]);
        return 0;
}

int f(char *des, char *src){
        if( strlen(src) >0)
                strcpy(des,src);
        else
                des[0]='b';
        return 0;
}
```

E.g., \$1 => a, \$2 => argv[1]

strcpy(\$1, \$2) ∧ Size(\$1) < Len(\$2)

Specifying constraints on vulnerabilities over *attributes*

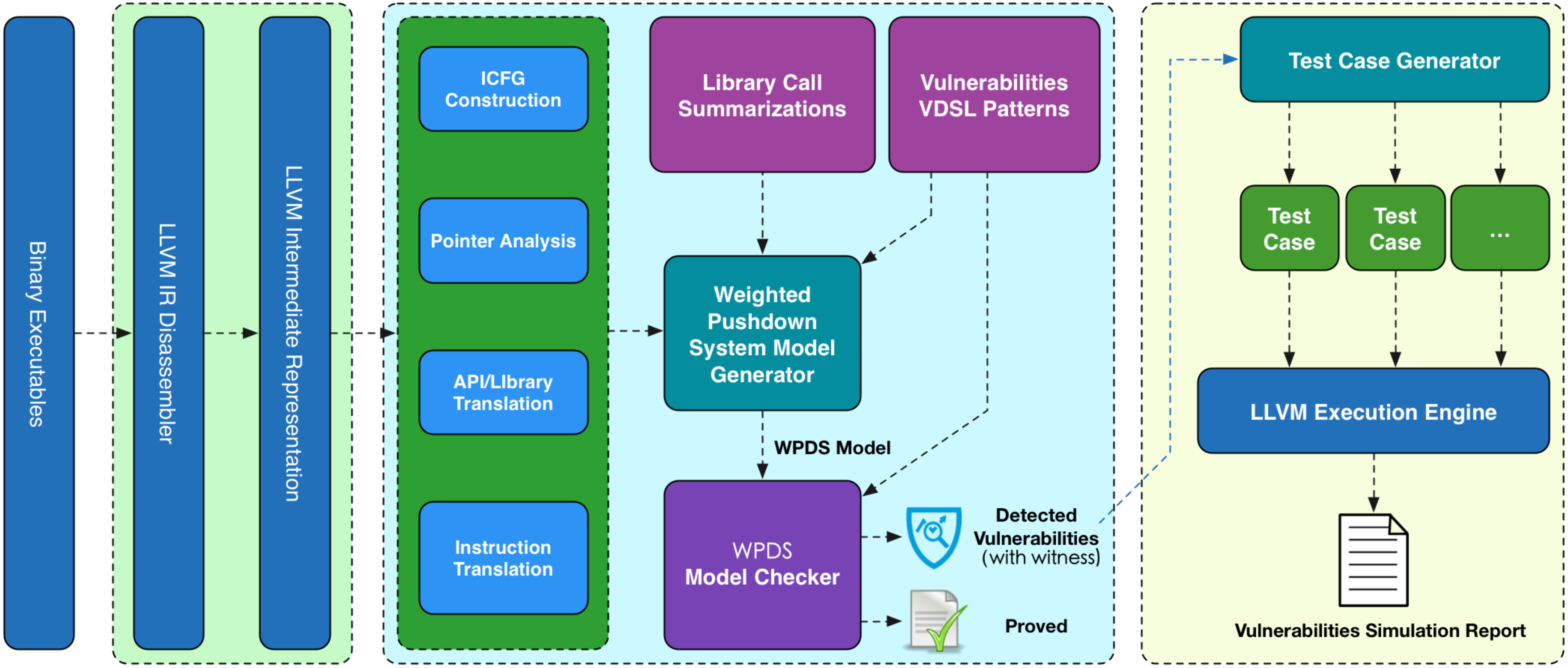
Null Pointer Dereference Vulnerability Pattern

$$Assign_1(\$1, *\$2) \wedge Val(\$2) = 0.$$

Division by Zero Vulnerability Pattern

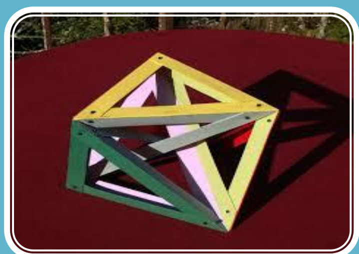
$$Assign_3(\$1, \$2, \$3) \wedge Val(\$3) = 0.$$

Not working



Binary Executables
LLVM IR Disassembler
LLVM Intermediate Representation
ICFG Construction
Pointer Analysis
API/LIbrary Translation
Instruction Translation
Library Call Summarizations
Vulnerabilities VDSL Patterns
Weighted Pushdown System Model Generator
WPDS Model
WPDS Model Checker
Detected Vulnerabilities (with witness)
Proved
Test Case Generator
Test Case
Test Case
...
LLVM Execution Engine
Vulnerabilities Simulation Report

Three main problems in applying FM



Models

Scalability

Domain Knowledge

Since 2015

Program Analysis

- Taint Analysis
- Symbolic Execution
- Dynamic Analysis
- Machine Learning
- …

Cross-Architecture Cross-OS Binary Search

- Why?
  - Plagiarism Detection, Clone detection, Vulnerability extrapolation , Search engine for machine code, Code property inference, Type inference, Partial decompilation, Malware signature generation, Vulnerability signature generation

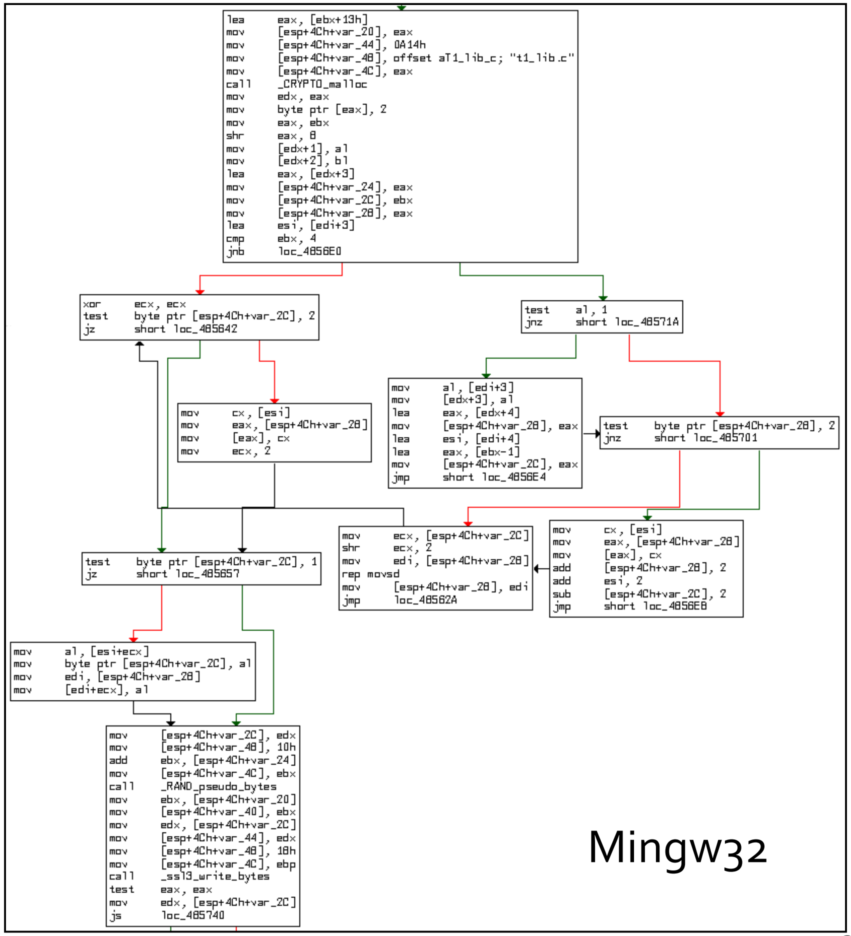
Mahinthan Chandramohan, Yinxing Xue, Zhengzi Xu, Yang Liu, Chia Yuan Cho and Hee Beng Kuan Tan. **BinGo: Cross-Architecture Cross-OS Binary Search**, *ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE 2016)* , Seattle, WA, USA. November 13-18, 2016.

Reality: The Heartbleed Example

- Same program (at source code level) might look totally different at machine code level

```
lea     edx, [ebx+13h]
add     edi, 3
mov     [esp+4Ch+var_20], edx
mov     [esp+4Ch+dest], edx
mov     [esp+4Ch+n], 0A14h
mov     [esp+4Ch+src], offset aT1_lib_c; "t1_lib.c"
call    CRYPTO_malloc
mov     byte ptr [eax], 2
mov     ebp, eax
mov     eax, ebx
shr     eax, 8
mov     [ebp+2], bl
lea     edx, [ebp+3]
mov     [ebp+1], al
mov     [esp+4Ch+n], ebx; n
mov     [esp+4Ch+dest], edx; dest
mov     [esp+4Ch+var_24], edx
mov     [esp+4Ch+src], edi; src
call    _memcpy
mov     edx, [esp+4Ch+var_24]
mov     [esp+4Ch+src], 10h
add     ebx, edx
mov     [esp+4Ch+dest], ebx
call    RAND_pseudo_bytes
mov     edx, [esp+4Ch+var_20]
mov     [esp+4Ch+n], ebp
mov     [esp+4Ch+src], 18h
mov     [esp+4Ch+dest], esi
mov     [esp+4Ch+var_40], edx
call    ssl3_write_bytes
test    eax, eax
js      short loc_80B7920
```

GCC



Desired Properties

P1. Resilient to the syntax and structural gaps introduced due to architecture, OS and compiler differences.

P2. Accurate by considering the complete function semantics.

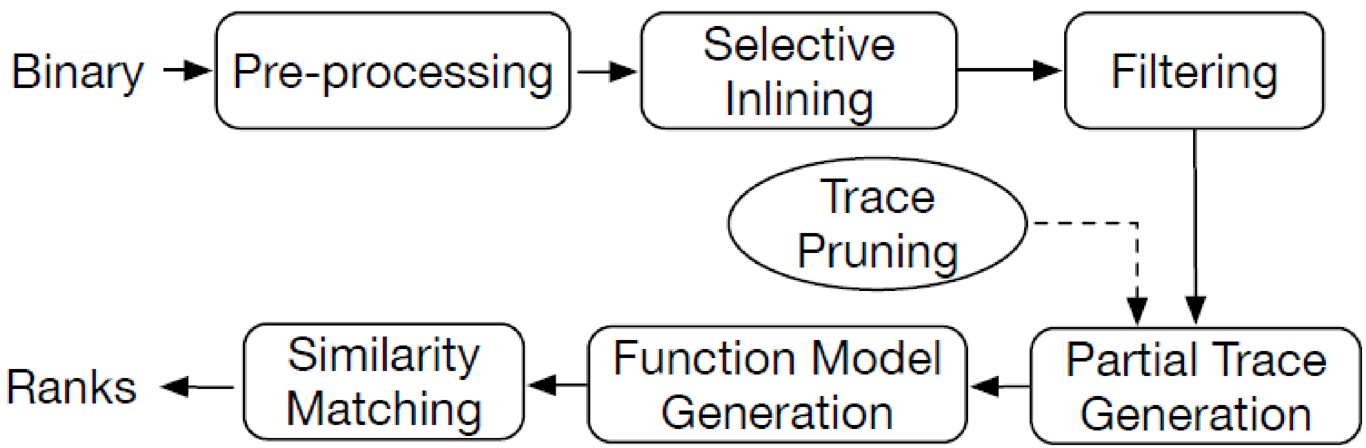
P3. Scalable to large size real world binaries.

Table 1: Comparison of existing techniques. Here, ✘ denotes *limited* support, while ✓ and ✗ represent *full* and *no* support, respectively.

Tool	Technique	P1 Resilient	P2 Accurate	P3 Scalable
Tracy [10]	Static	✗	✗	✓
CoP [26]	Static	✗	✘	✗
Bug search [29]	Static	✗	✗	✗
DISCOVRE [35]	Static	✗	✗	✓
BLEX [14]	Dynamic	✓	✓	✗
BinGo	Static	✓	✓	✓

Cross-architecture Cross Platform Vulnerability Mining

Smart Mining Learning + Deep Program Analysis



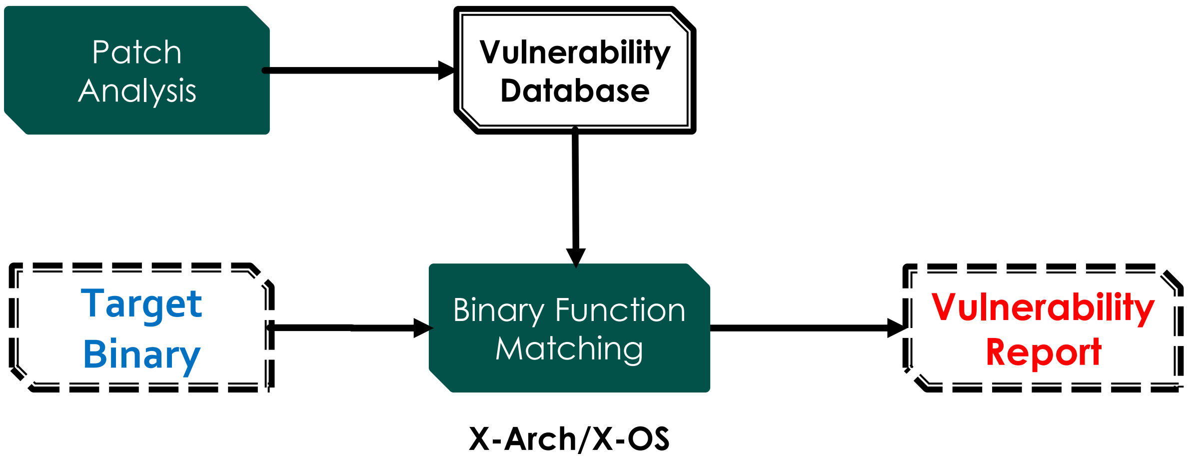
Successfully Tested on Commercial Software

Two zero-day vulnerabilities (RCE),

Nine zero-day vulnerabilities (DoS, info leakage)

zero-day vul.: unknown vul.
DoS: Deny of Service
RCE: remote code execution

Binary Vulnerability Analysis

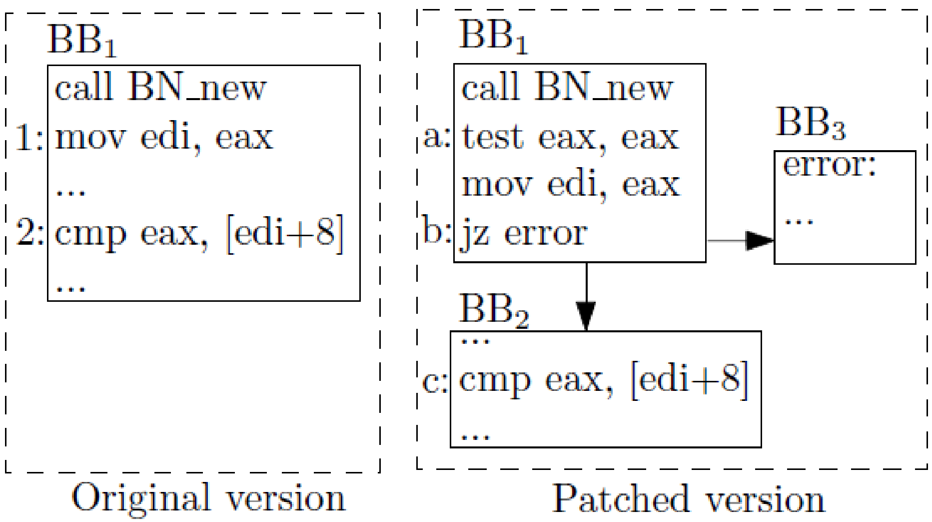


Towards Understanding the Pain and Pills

SPAIN: Security Patch Analysis for Binaries

Zhengzi Xu, Bihuan Chen, Mahinthan Chandramohan, Yang Liu and Fu Song
ICSE 2017

Motivating Example



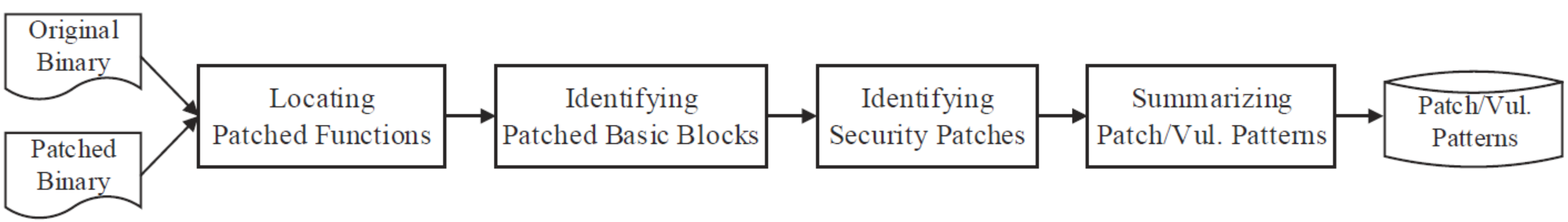
a: Additional instruction to test the value of a register "eax"

b: Additional flow branch to error

Goal: to capture the patch location and summarize patch/vulnerability patterns at binary level

SPAIN Overview

- A scalable binary-level patch analysis framework, SPAIN, to automatically identify security patches and summarize patch patterns and their corresponding vulnerability patterns



Step 3: Identifying Security Patches

- Assumption: a security patch is less unlikely to change the semantics

- Trace sem summary = post-state – pre-state

```
mov   eax, 0x04
sub   ebx, eax
```

(a) Sample Code Segment

Pre-state:

$\texttt{Reg} = \{\texttt{eax} = 0, \texttt{ebx} = 0, ..\}$

$\texttt{Flag} = \{\texttt{zf} = 0, \texttt{sf} = 0, ..\}$

$\texttt{Mem} = \{0, 0 \ldots 0\}$

Post-state:

$\texttt{Reg}' = \{\texttt{eax}' = \texttt{0x4}, \texttt{ebx}' = -\texttt{0x4}, ..\}$

$\texttt{Flag}' = \{\texttt{zf}' = 0, \texttt{sf}' = 1, ..\}$

$\texttt{Mem}' = \{0, 0 \ldots 0\}$

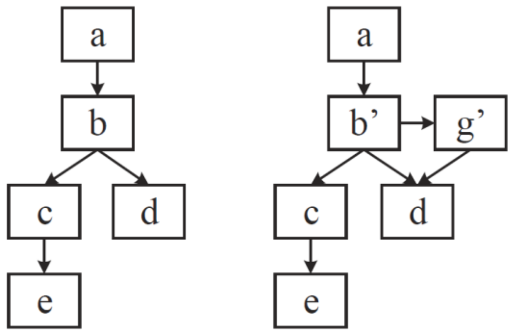
(b) Pre- and Post-State before and after Executing the Code Segment

- Identify the overall semantic change

  Sem Diff1: a->b'->c vs a->b->c

  Sem Diff2: a->b'->d vs a->b->d

  Sem Diff3: a->b'->g'->d vs a->b->d



(a) Original Function (b) Patched Function

Evaluation: Accuracy of SPAIN

- We manually identified all the security and non-security patches of all the 20 versions of OpenSSL 1.0.1 (446,747 LOC) by analyzing its commits on GitHub (63 CVEs)

TABLE I: Accuracy and Performance on OpenSSL

Ver.	CVE	Sec. Pat.		Non-Sec. Pat.		T.P.	F.P.	T. (s)
		G.T.	Iden.	G.T.	Iden.			
0–a	1	7	4	11	0	0.57	0	11
a–b	0	0	0	5	0	–	0	13
b–c	1	3	2	0	0	0.67	–	7
c–d	3	19	8	27	0	0.42	0	60
d–e	0	0	0	6	1	–	0.17	8
e–f	3	12	9	17	10	0.75	0.59	14
f–g	2	10	8	6	1	0.8	0.17	11
g–h	8	29	18	4	1	0.62	0.25	20
h– i	9	37	24	23	8	0.65	0.35	54
i–j	4	23	11	5	2	0.48	0.4	35
j–k	7	25	19	15	5	0.76	0.33	39
k–l	0	0	0	1	0	–	0	7
l–m	8	34	25	7	3	0.74	0.43	95
m–n	5	60	49	38	5	0.82	0.13	76
n–o	1	0	0	6	0	0	0	9
o–p	2	2	1	1	0	0.5	0	9
p–q	2	39	31	40	11	0.79	0.28	48
q–r	1	10	9	3	0	0.9	0	18
r–s	6	13	11	2	0	0.85	0	36
Sum.	63	323	229	217	47	0.71	0.22	–

Evaluation: Scalability of SPAIN

- Linux Kernel is an open source software with around 18,963,973 LOC and developed since 2002

- Adobe PDF Reader is a closed source software. We use two of its libraries, 3difr.x3d and AXSLE.dll, which have around 1,293 and 4,874 functions respectively

Version	Total Func.	Sec. Patched	T. (s)
Linux: 3.16.2 – 3.16.3	249341	1221	807
difr.x3d: 11.0.08 – 11.0.09	1293	12	21
difr.x3d: 11.0.13 – 11.0.14	1293	13	23
difr.x3d: 11.0.15 – 11.0.16	1293	11	20
AXSLE.dll: 11.0.15 – 11.0.17	4875	27	84

Evaluation: Patch & Vulnerability Patterns

Vul. type	Concrete Vulnerability	Vulnerability Pattern	Concrete Patch	Patch Pattern
Double Free	call BN_to_ASN1_INTEGER test eax, eax mov esi, eax ... mov [esp + 4Ch + var_4C], esi call ASN1_STRING_clear_free ... mov [esp + 4Ch + var_4C], esi call ASN1_STRING_clear_free	call ⟨untrusted_func⟩ ⟨sanitycheck⟩ : ⟨return_value⟩ ... mov ⟨func_param⟩, ⟨return_value⟩ call ⟨free⟩ ... mov ⟨func_param⟩, ⟨return_value⟩ call ⟨free⟩	call BN_to_ASN1_INTEGER test eax, eax mov esi, eax ... mov [esp + 4Ch + var_4C], esi call ASN1_STRING_clear_free ...	call ⟨untrusted_func⟩ ⟨sanity check⟩ : ⟨return_value⟩ ... mov ⟨func_param⟩, ⟨return_value⟩ call ⟨free⟩ ...

Integer Underflow Overflow	mov eax, [esp + 4Ch + arg_4]
	...
	mov ecx, eax
	sub ecx, [esp + 4Ch + var_2C]
	add [esp + 4Ch + var_24], ecx
	...
	mov esi, [esp + 4Ch + _24]
	mov ecx, [esp + 4Ch + arg_8]
	mov [ecx], esi
		mov ⟨TNT_IN⟩, ⟨untrusted_src⟩
		...
		⟨arith_op⟩ ⟨TNT_result⟩, ⟨TNT_IN⟩
		...
		mov ⟨sec_SSTV_sink⟩, ⟨TNT_result⟩
			mov eax, [esp + 4Ch + arg_4]
			...
			mov edi, [esp + 4Ch + var_2C]
			cmp eax, edi
			jl error
			mov ecx, eax
			sub ecx, [esp + 4Ch + var_2C]
			add [esp + 4Ch + var_24], ecx
			...
			mov esi, [esp + 4Ch + _24]
			mov ecx, [esp + 4Ch + arg_8]
			mov [ecx], esi
				mov ⟨TNT_IN⟩, ⟨untrusted_src⟩
				⟨sanity check⟩ : ⟨TNT_IN⟩
				...
				⟨arith_op⟩ ⟨TNT_result⟩, ⟨TNT_IN⟩
				...
				mov ⟨sec_SSTV_sink⟩, ⟨TNT_result⟩

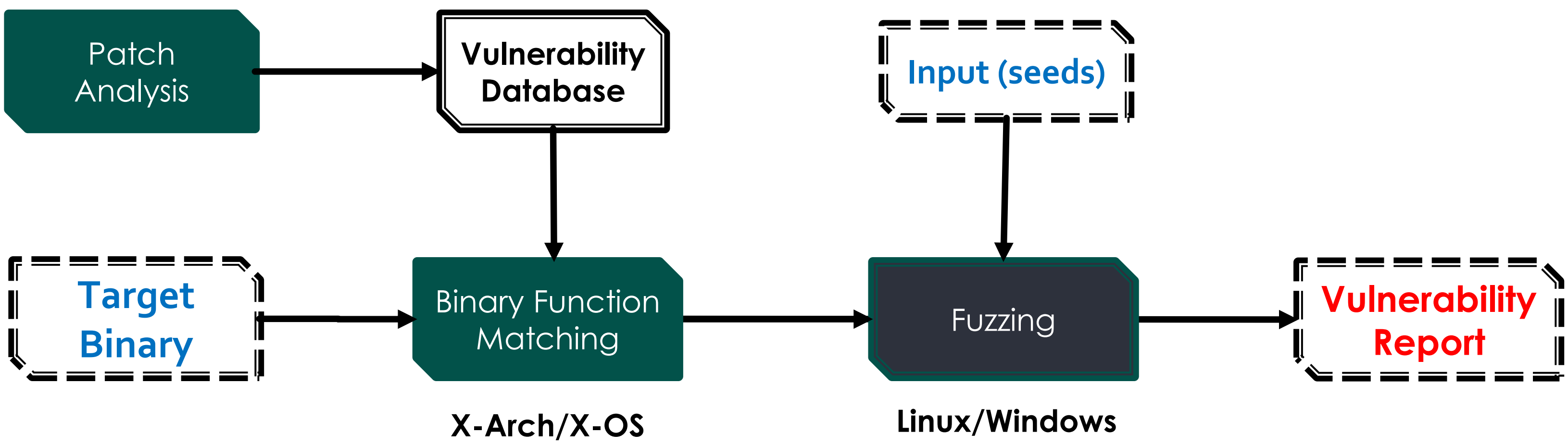
Use After Free	mov ebp, [esp + 0ECh + arg_0]
	...
	mov [esp + 0ECh + dest], ebp
	call ssl3_release_read
		mov ⟨TNT_IN⟩, ⟨untrusted_src⟩
		...
		mov ⟨func_param⟩, ⟨TNT_pointer⟩
		call ⟨free⟩
			mov ebp, [esp + 0ECh + arg_0]
			...
			mov eax, [ebp + 58h]
			mov eax, [eax + 0F8h]
			test eax, eax
			jnz ⟨do not release⟩
			...
			mov [esp + 0ECh + dest], ebp
			call ssl3_release_read
				mov ⟨TNT_IN⟩, ⟨untrusted_src⟩
				⟨sanity check⟩ : ⟨TNT_pointer⟩
				...
				mov ⟨func_param⟩, ⟨TNT_pointer⟩
				call ⟨free⟩

NULL Pointer Dereference	callBN_new mov edi, eax ... cmp eax, [edi + 8]	call⟨untrusted_func⟩ ... ⟨mem_deref⟩ : ⟨return_value⟩	call BN_new test eax, eax mov edi, eax jz error ... cmp eax, [edi + 8]	call⟨untrusted_func⟩ ⟨sanity check⟩ : ⟨return_value⟩ ... ⟨mem_deref⟩ : ⟨return_value⟩

Buffer Overflow	mov ebx, [esp + 3Ch + arg_8]
	...
	mov [esp + 3Ch + n], ebx
	call eax
		mov ⟨TNT_IN⟩, ⟨untrusted_src⟩
		mov ⟨func_param⟩, ⟨TNT_IN⟩
		...
		call ⟨untrusted_func⟩
			mov ebx, [esp + 3Ch + arg_8]
			cmp ebx, [esp + 3Ch + arg_4]
			jl error
			...
			mov [esp + 3Ch + n], ebx
			call eax
				mov ⟨TNT_IN⟩, ⟨untrusted_src⟩
				⟨sanity check⟩ : ⟨TNT_IN⟩
				...
				mov ⟨func_param⟩, ⟨TNT_IN⟩
				call ⟨untrusted_func⟩

- Details can be found at: https://sites.google.com/site/binaryanalysisicse2017/claim/patterns

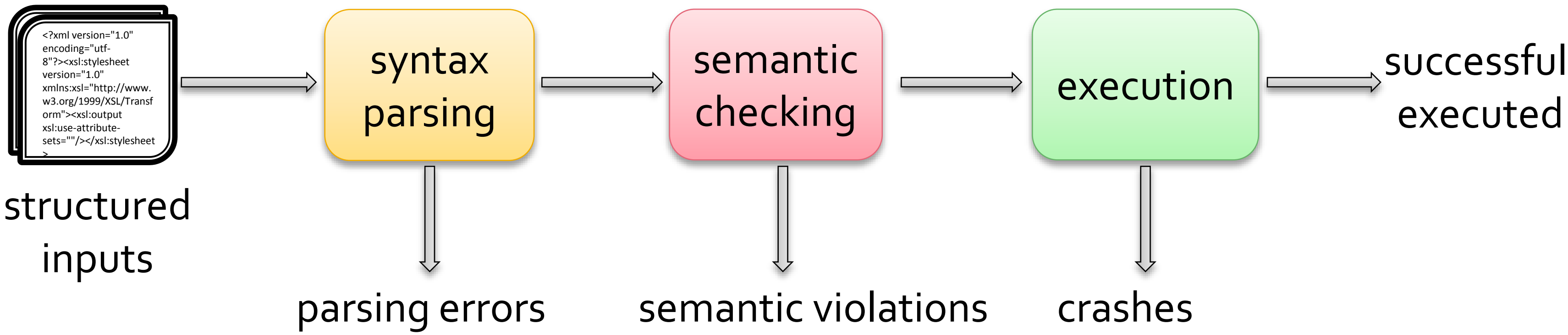
Binary Vulnerability Analysis



Skyfire: Data-Driven Seed Generation for Fuzzing

Junjie Wang, Bihuan Chen, Lei Wei, and Yang Liu
S&P 2017

Fuzzing Target: Stages of processing structured inputs

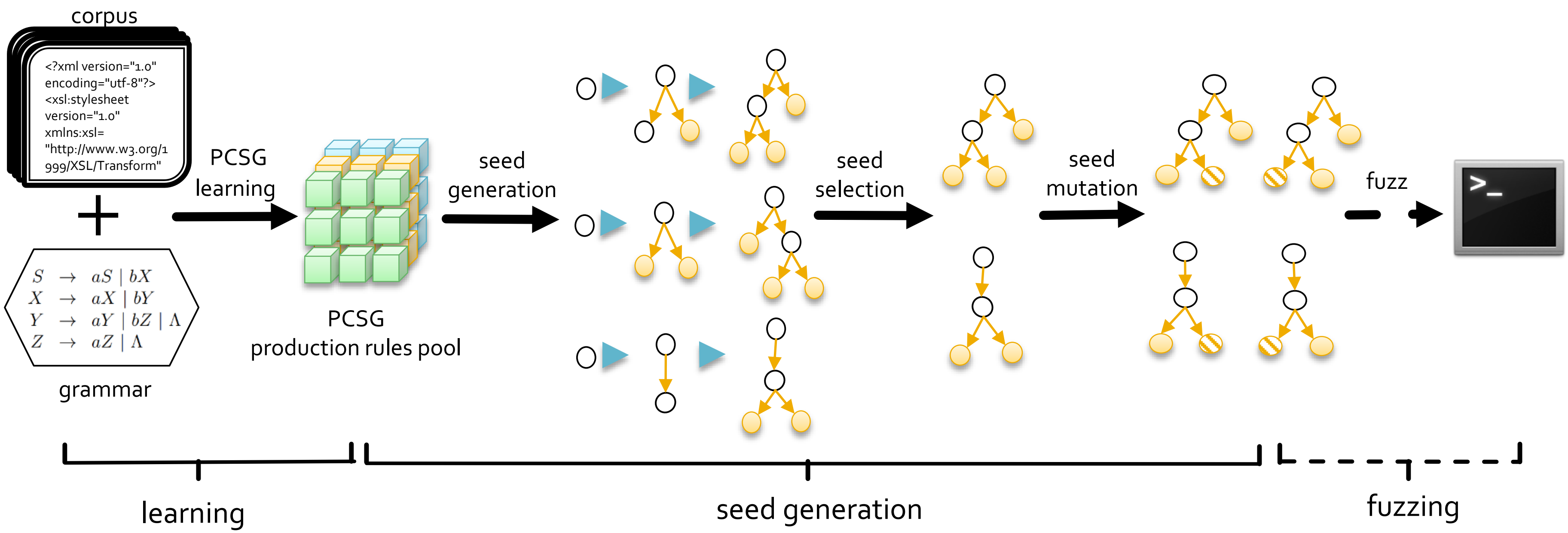


Passing syntax parsing/semantic checking

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

- ❑ Design goal: generates well-distributed test cases
  - ❑ Capture the syntax rules and valid constants
  - ❑ Extend context-free grammar (CFG) to model semantic rules

Skyfire Approach Overview



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 $\alpha \rightarrow \beta_1\beta_2\ldots\beta_n$, $\alpha \in N$, $n \geq 1$, $\beta_i \in (N \cup \Sigma)$ for i = 1…n,

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

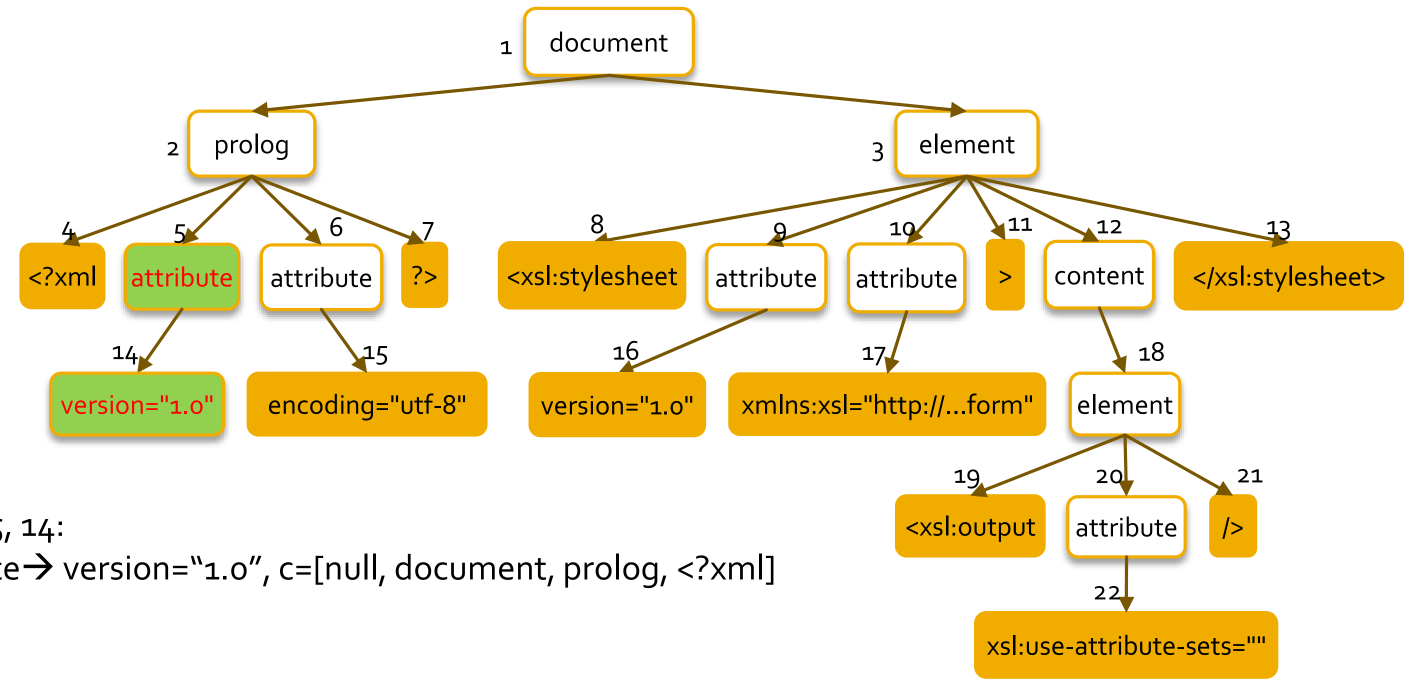
- $[c]\alpha \rightarrow \beta_1\beta_2\ldots\beta n$,

<type of α 's great-grandparent, type of α 's grandparent, type of α 's parent, value of α 's first sibling or type of α 's first sibling 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



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>
```

Experiment setup

- Sablotron
  - Adobe PDF Reader and Acrobat.
- libxslt
  - Chrome browser, Safari browser, and PHP 5
- libxml2
  - Linux, Apple iOS/OS X, and tvOS

Bugs found in XSLT 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.

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

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

We discovered 19 new memory corruption bugs (among which we discovered 16 new vulnerabilities and received 33.5k USD bug bounty rewards) and 32 denial-of-service bugs

On-going Testing Works

- Symbolic Execution (Whitebox)
  - Loop (FSE 16, FSE 17, ASE 17), API summary and SMT
- Good Test Cases (Blackbox)
  - (Systematic) Test Case Generation
  - Model Based Testing:
    - FSE 2017: Guided, Stochastic Model-Based GUI Testing of Android Apps
- Feedback Based Testing (Greybox)
  - Improve the feedback
    - FSE 2017: Steelix: Program-State Based Binary Fuzzing
  - Runtime Seed Prioritization
- Combining Different Ideas
  - Testing Orchestration
    - Static Analysis, Random Testing, Taint + Machine Learning

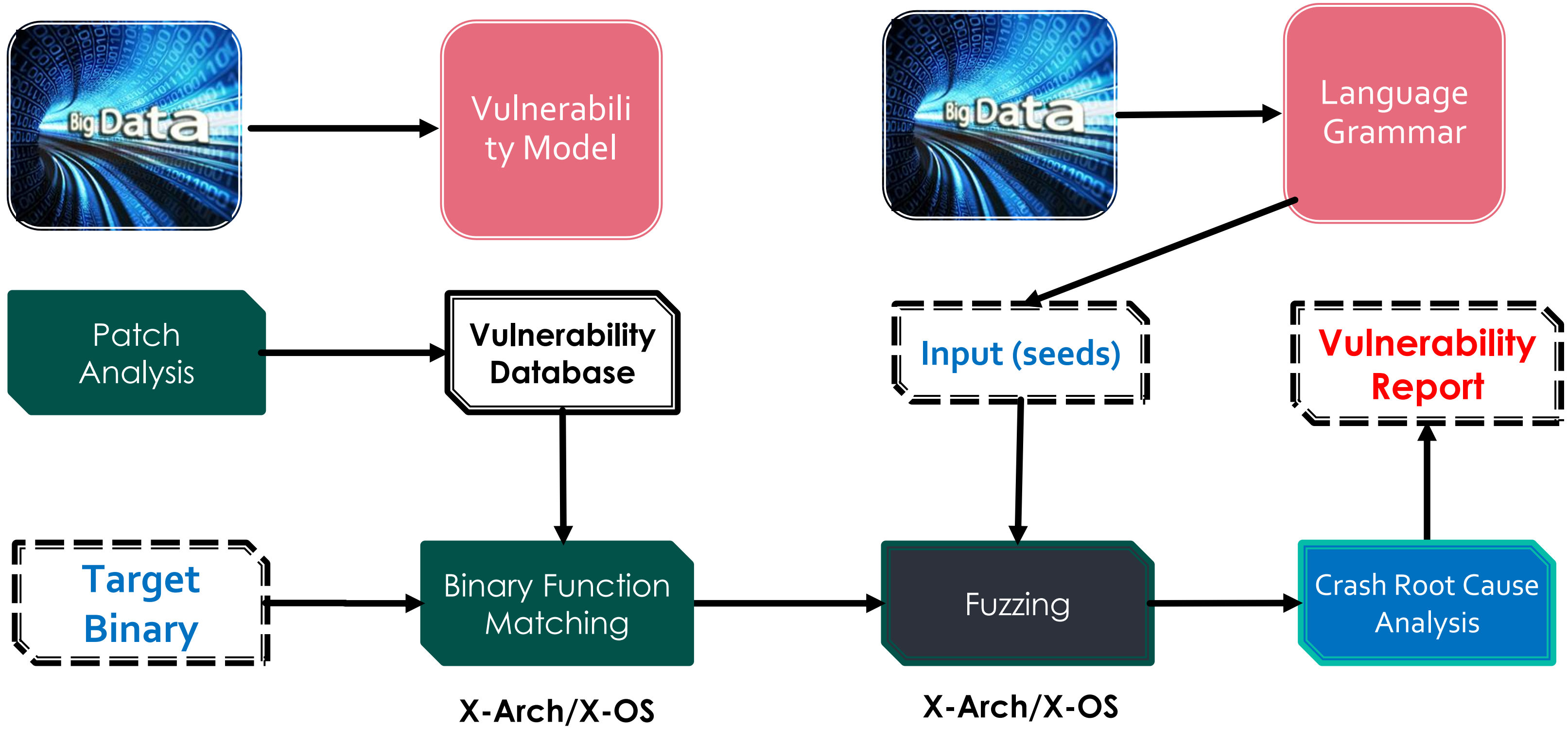
Mobile Performance Testing

Security Protocol Fuzzing

Android OS Fuzzing and Attack Generation

Robot and Automotive Testing

Binary Vulnerability Analysis



- Static Analysis
- Dynamic Analysis
- Machine Learning
- Data Analytics

Summary of the Ideas

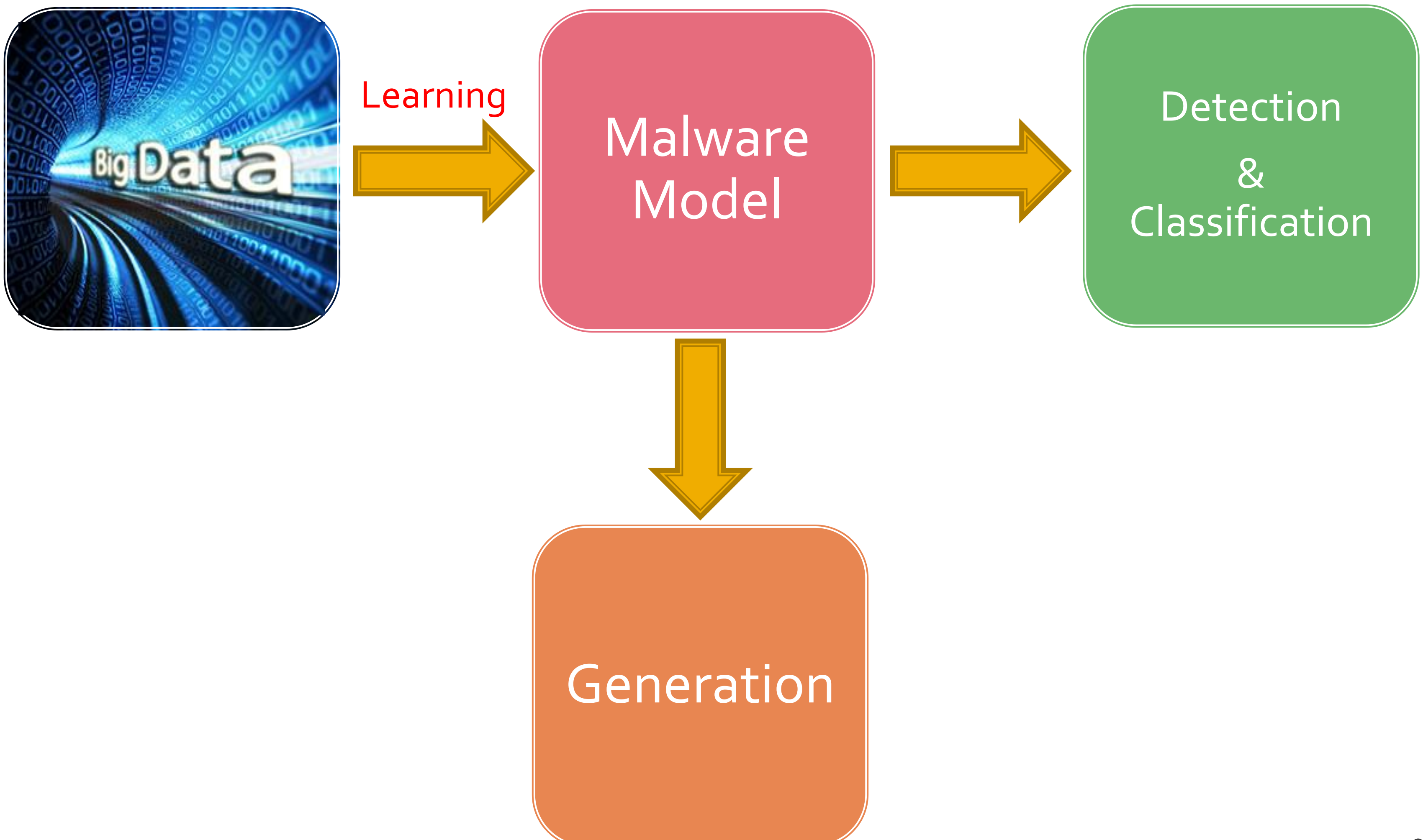
Correctness | Security | Reliability | Performance | Robustness

Analyzing Complex Systems

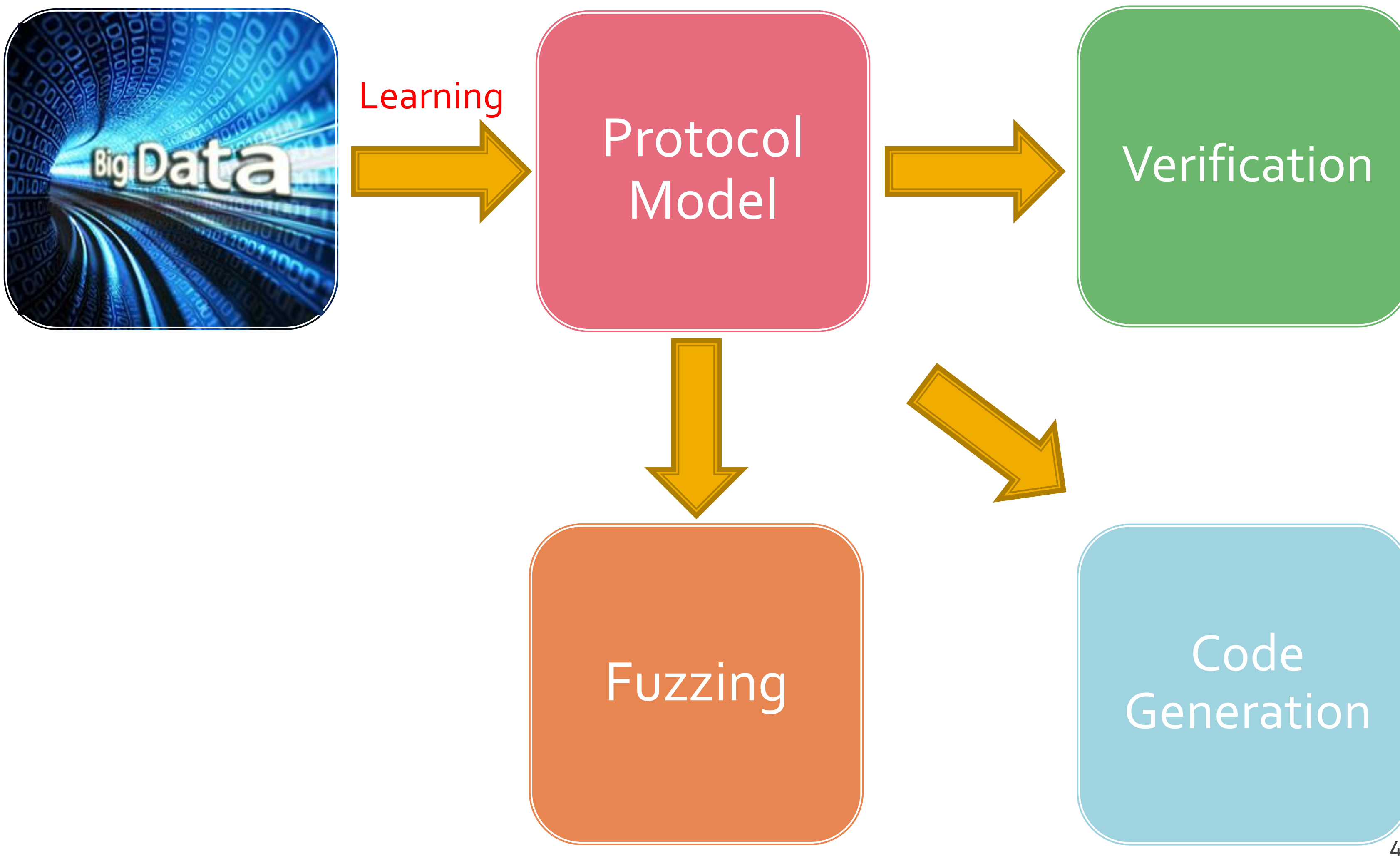
Formal Models and Precise Semantics

Formal Analysis | Program Analysis | Machine Learning | Data Analytics

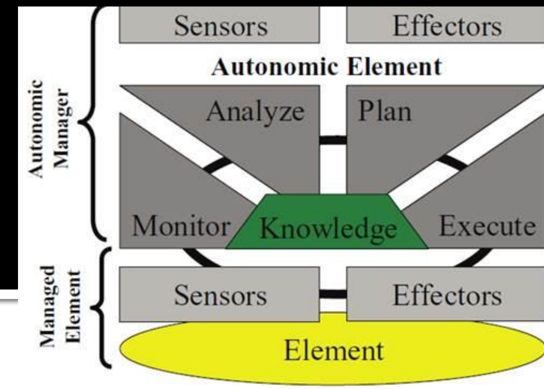
Similar Ideas on Malware Analysis



Similar Ideas on Security Protocol Analysis

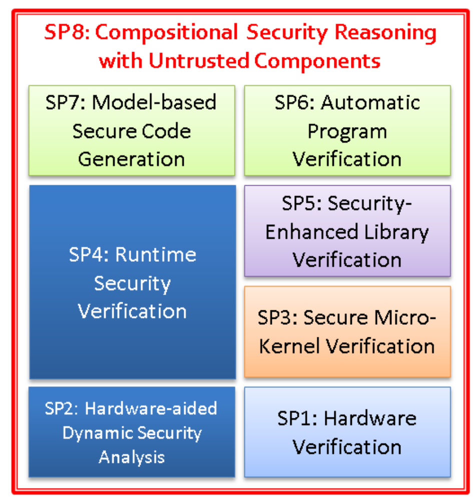


Security Research



- Runtime Adaptive Security (Trust + Resilient)
  - Runtime attack monitoring (Logic and physical model based)
  - Dynamic Adaptation (using based on attack-defence tree and game theory) for ROS
  - Security verification and resilient guarantee
  - Platforms: IT architecture, AV/Drones, IoT/urban computing/smart nation
  - Next generation of security operation center
- Mobile security: malware and vulnerability detection, trend, attribution…

- Binary Analysis
  - Vulnerability learning, matching and fuzzing
  - Patch Analysis and Summary
  - Crash root-cause analysis and debugging
  - Binary repairing and hardening
  - Binary reverse engineering
- Security Verification
- Security using HW and Hardware Security
- Network Security



Thank you

Whatever you learned will be useful somehow somewhere