Dynamic Taint Analysis

Dynamic Taint Analysis for Automatic Detection, Analysis and Signature Generation of Exploits on Commodity Software

James Newsome and Dawn Song

Appeared in NDSS'06

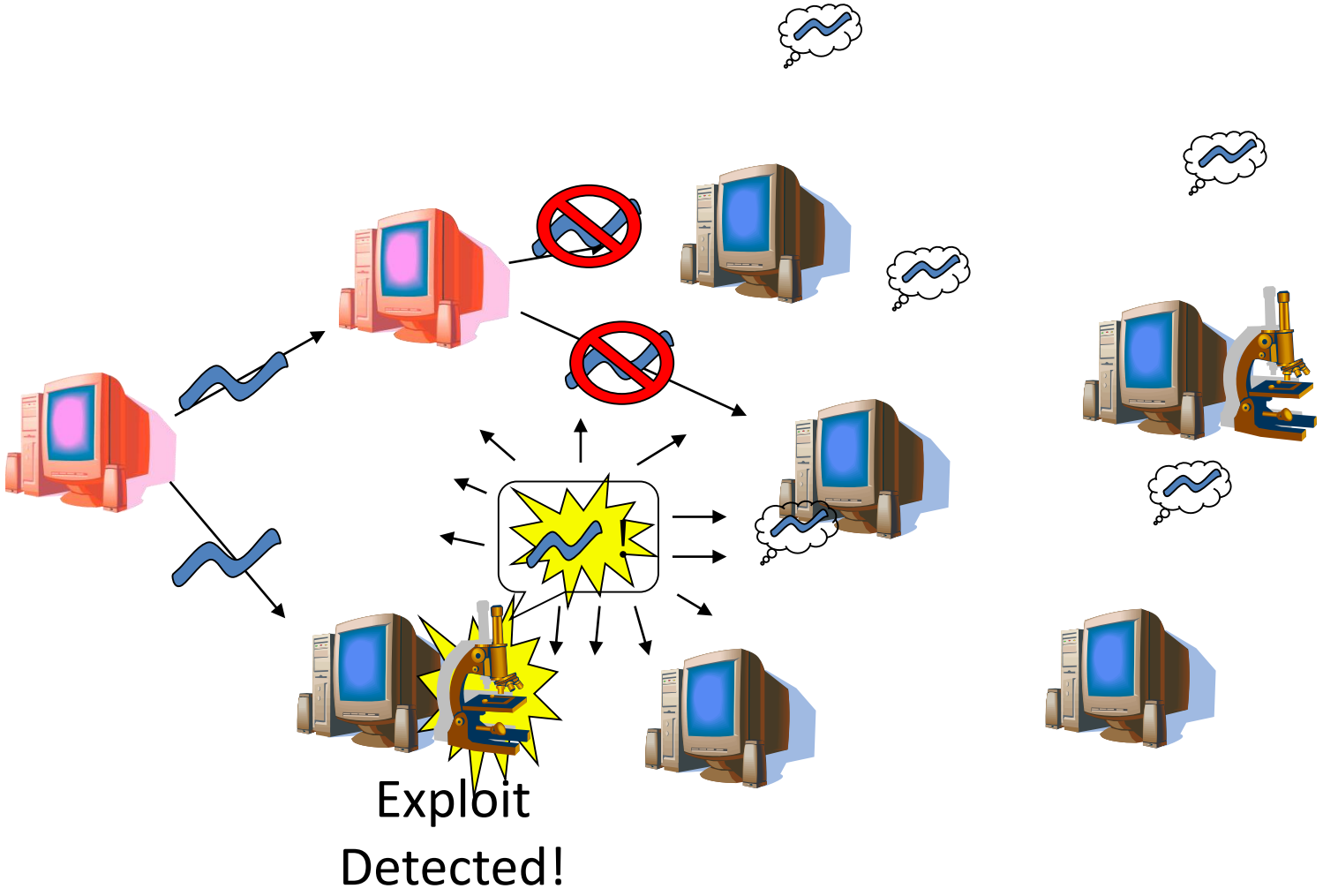
All Rights Reserved. Copyright 2010 by CSRG-Yin Lab.

Problem: Internet Worms

- Propagate by exploiting vulnerable software
- No human interaction needed to spread
- Able to rapidly infect vulnerable hosts
  - Slammer scanned 90% of Internet in 10 minutes
- Need **automatic** defense against new worms

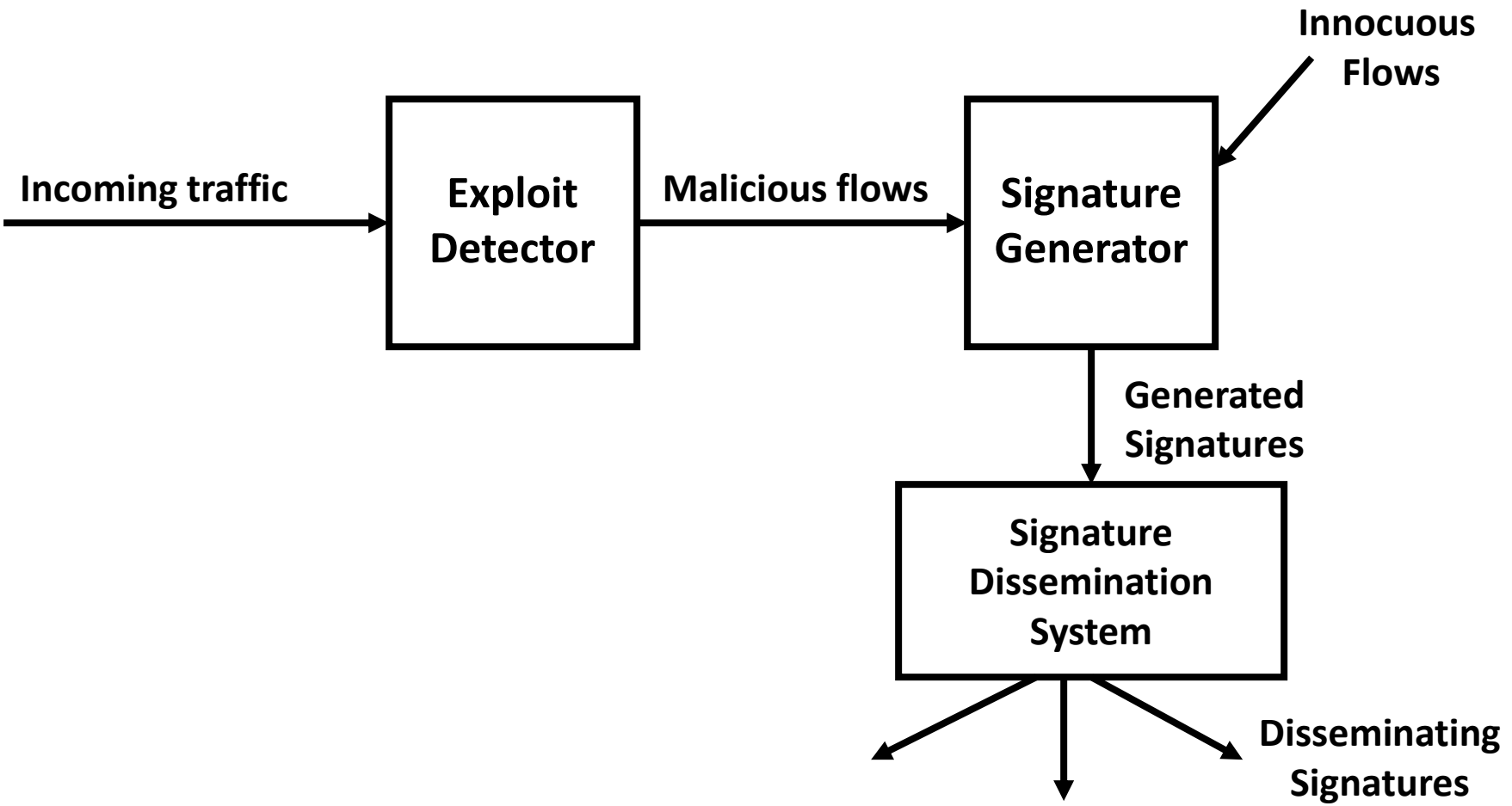
All Rights Reserved. Copyright 2010 by CSRG-Yin Lab.

Automatic Worm Defense



All Rights Reserved.
Copyright 2010 by CSRG-Yin Lab.

Architecture



All Rights Reserved.
Copyright 2010 by CSRG-Yin Lab.

Common Traits of Software Exploits

- Most known exploits are *overwrite attacks*
- Attacker's data overwrites sensitive data
- Common overwrite vulnerabilities:
  - Buffer overflows
  - Format string
  - Double-free
- Common overwrite targets:
  - Return address
  - Function pointer

All Rights Reserved.
Copyright 2010 by CSRG-Yin Lab.

Approach: Dynamic Taint Analysis

- Hard to tell if data is sensitive when it is *written*
  - Binary has no type information
- Easy to tell it is sensitive when it is *used*
- Approach: *Dynamic Taint Analysis*:
  - Keep track of *tainted* data from untrusted sources
  - Detect when tainted data is used in a sensitive way
    - *e.g.*, as return address or function pointer

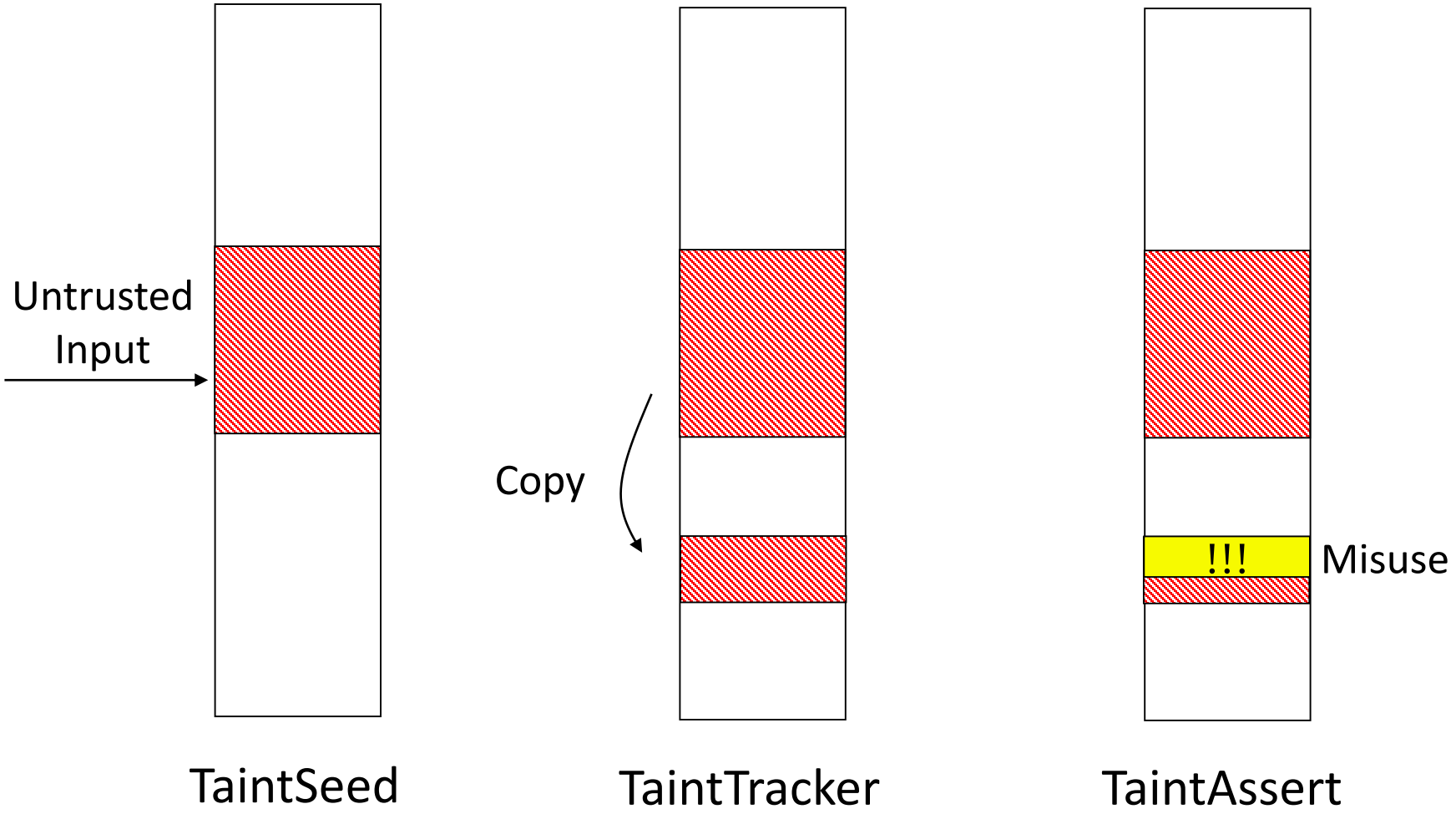
All Rights Reserved. Copyright 2010 by CSRG-Yin Lab.

Design & Implementation: TaintCheck

- Use Valgrind to monitor execution
  - Instrument program binary at run-time
  - No source code required
- Track a taint value for each location:
  - Each byte of tainted memory
  - Each register

All Rights Reserved. Copyright 2010 by CSRG-Yin Lab.

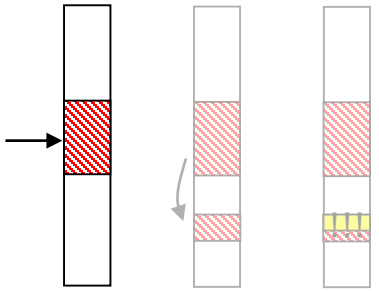
TaintCheck Components



All Rights Reserved.
Copyright 2010 by CSRG-Yin Lab.

TaintSeed

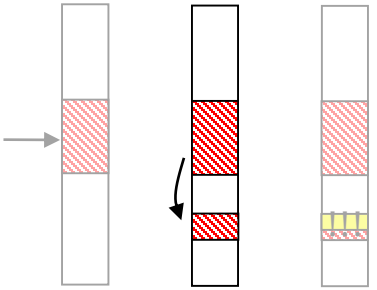
- Monitors input via system calls
- Marks data from untrusted inputs as tainted
  - Network sockets (default)
  - Standard input
  - File input
    - (except files owned by root, such as system libraries)



All Rights Reserved.
Copyright 2010 by CSRG-Yin Lab.

TaintTracker

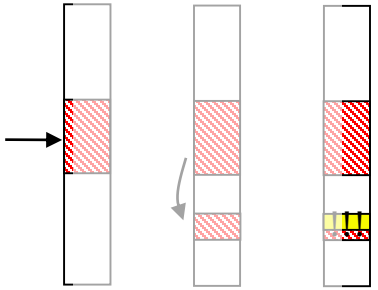
- Propagates taint
- Data movement instructions:
  - *e.g.*, move, load, store, etc.
  - Destination tainted iff source is tainted
  - Taint data loaded via tainted index
    - *e.g.,* unicode = translation_table[tainted_ascii]
- Arithmetic instructions:
  - *e.g.,* add, xor, mult, etc.
  - Destination tainted iff *any* operand is tainted
- Untaint result of constant functions
    - xor eax, eax



All Rights Reserved.
Copyright 2010 by CSRG-Yin Lab.

TaintAssert

- Detects when tainted data is misused
  - Destination address for control flow (default)
  - Format string (default)
  - Argument to particular system calls (e.g., `execve`)
- Invoke Exploit Analyzer when exploit detected



All Rights Reserved.
Copyright 2010 by CSRG-Yin Lab.

Coverage: Attack Classes Detected

	Format String	Stack Overflow	Heap Overflow	Heap Corruption (Double Free)
Return Address	✓	✓	N/A	✓
Function Pointer	✓	✓	✓	✓
Fn Ptr Offset (GOT)	✓	✓	✓	✓
Jump Address	✓	✓	✓	✓

All Rights Reserved.
Copyright 2010 by CSRG-Yin Lab.

Experimental Results: Detects Many Attacks

Vulnerable Program	Overwrite Method	Overwrite Target	Detected
ATPhttpd	Buffer overflow	Return address	✓
Synthetic	Buffer overflow	Function pointer	✓
Synthetic	Buffer overflow	Format string	✓
cfingerd	syslog format string	GOT entry	✓
wu-ftpd	vsnprintf format string	Return address	✓

Others including slapper, SQL Slammer

All Rights Reserved. Copyright 2010 by CSRG-Yin Lab.

Comparison to Previous Mechanisms

- Used Wilander testbed [NDSS03]
  - 20 exploit tests
  - Overwrite Targets: return address, base pointer, function pointer, longjmp buffer
  - Overwrite Techniques: overflow to target, overflow to pointer to target
  - Evaluate previous run-time detection mechanisms

All Rights Reserved. Copyright 2010 by CSRG-Yin Lab.

Comparison Results

Mechanisms	Attacks Prevented or Halted
StackGuard	15%
Stack Shield	30%
ProPolice	50%
Libsafe & Libverify	20%
TaintCheck	100%

All Rights Reserved.
Copyright 2010 by CSRG-Yin Lab.

Other Applications

- Information leakage detection/analysis
- Malware analysis
- Fuzzing
- A base for symbolic execution/concolic testing
- …

Pointer Tainting

- mov eax, [ebx + 4]

  When ebx is tainted, shall eax be tainted?

- Often used for table lookup, e.g.,
  - Convert from ascii to Unicode
  - Convert a date from one format to another

- It may cause taint explosion

Over tainting & Under tainting

- xor eax, eax
- sub eax, eax

- Taint granularity is important (bit, byte, word, etc.)
  - Coarser granularity may cause over tainting

Examples of bit-level tainting rules

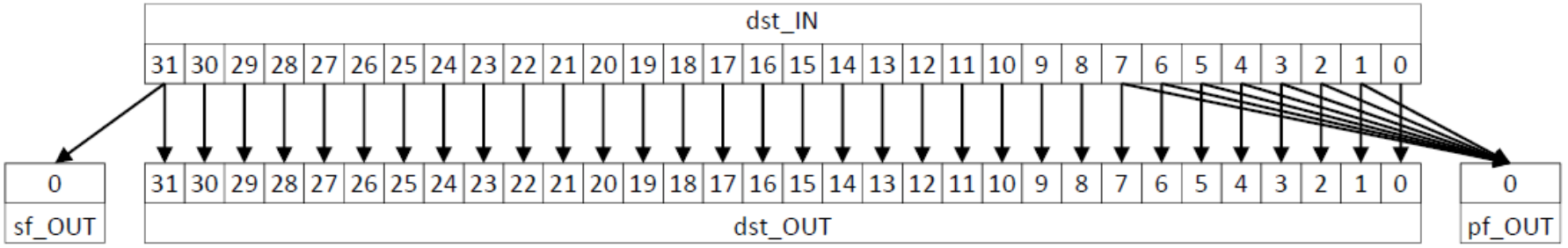


Figure 2: Information flows of *dst* in the `or` instruction

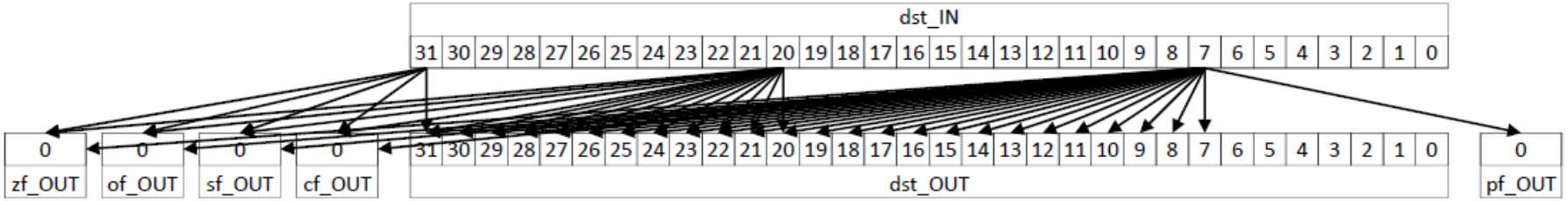


Figure 3: Information flow of bits 7, 20 and 31 of `dst` in `sbb`

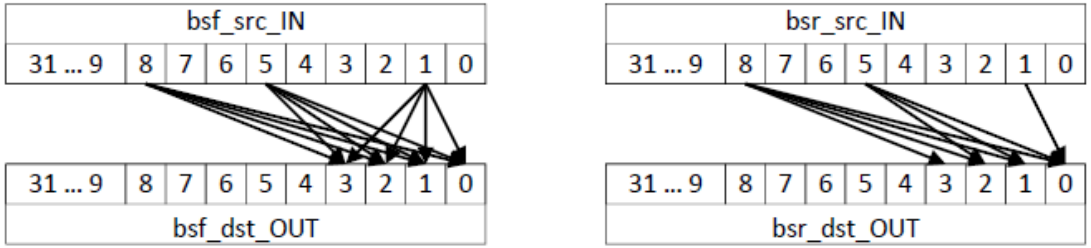


Figure 4: Comparison between *bsf* and *bsr*

Rules for x86 Instructions

						Previously Published Taint Trackers								
Instruction	**Inputs**	**Outputs**	**# Cases**	**Runtime**	**Flow Type**	**DroidScope[16]**	**Cat1**	**libdft[9]**	**Minemu[10]**	**Cat2**	**TEMU[15]**	**Cat3**	**Memcheck[32]**	**SPITA**
adc *dst, src*	dst,src,cf	dsr,src,zf,of,sf,af,cf,pf	4550	1m19s	U	A	A	I	A	A	S	S	U	S
add *dst, src*	dst,src	dst,src,zf,of,sf,af,cf,pf	4480	1m13s	U	A	A	I	A	A	A	A	S	S
and *dst, src*	dst,src	dst,src,zf,sf,pf	4288	1m05s	I	A	I	I	A	I	I	S	S	S
dec *dst*	dst	dst,zf,of,sf,af,pf	1184	20s	U	A	A	I	A	A	A	A	U	S
div *rm32*	edx,eax,rm32	edx,eax,rm32	9216	95m48s	D	A	A	I	N	A	A	A	A	D
idiv *rm32*	edx,eax,rm32	edx,eax,rm32	9216	307m04	A	A	A	I	N	A	A	A	A	A
imul1 *rm32*	eax,rm32	edx,eax,rm32,of,cf	6272	289m51s	U	A	A	I	N	A	A	A	U	U
imul2 *dst, rm32*	dst,rm32	dst,rm32,of,cf	4224	52m37s	U	A	A	I	N	A	A	A	U	U
imul3 *dst, rm32, imm32*	rm32,imm32	dst,rm32,imm32,of,cf	6272	53m56s	U	A	A	I	N	A	A	A	U	U
inc *dst*	dst	dst,zf,of,sf,af,pf	1184	19s	U	A	A	I	A	A	A	A	U	S
mul *rm32*	eax,rm32	edx,eax,rm32,of,cf	6272	16m02s	U	A	A	I	N	A	A	A	U	U
not *dst*	dst	dst	1024	15s	I	A	I	I	A	I	I	I	I	I
or *dst, src*	dst,src	dst,src,zf,sf,pf	4288	1m05s	I	A	I	I	A	I	I	S	S	S
rcl *dst, imm8*	dst,imm8,cf	dst,imm8,of,cf	1722	42s	A	A	A	N	A	A	A	A	A	S
rcr *dst, imm8*	dst,imm8,cf	dst,imm8,of,cf	1722	42s	A	A	A	N	A	A	A	A	A	S
rol *dst, imm8*	dst,imm8	dst,imm8,of,cf	1680	41s	A	A	A	N	A	A	A	A	S	S
ror *dst, imm8*	dst,imm8	dst,imm8,of,cf	1680	41s	A	A	A	N	A	A	A	A	S	S
sal *dst, imm8*	dst,imm8	dst,imm8,zf,of,sf,af,cf,pf	1840	35s	U	A	A	N	A	A	S	S	S	S
sar *dst, imm8*	dst,imm8	dst,imm8,zf,of,sf,af,cf,pf	1840	34s	D	A	A	N	A	A	S	S	S	S
sbb *dst, src*	dst,src,cf	dst,src,zf,of,sf,af,cf,pf	4550	1m21s	U	A	A	I*	A*	A	A	A*	A	S
shr *dst, imm8*	dst,imm8	dst,imm8,zf,of,sf,af,cf,pf	1840	35s	D	A	A	N	A	A	S	S	S	S
sub *dst, src*	dst,src	dst,src,zf,of,sf,af,cf,pf	4480	1m17s	U	A	A	I*	A*	A*	A*	A*	S	S
xor *dst, src*	dst,src	dsr,src,zf,sf,pf	4288	1m05s	I	A	I	I*	A*	A*	A*	A*	I	I
bsf *dst, src*	src	dst,src,zf	2080	31s	A	N	A	I	N	A	A	A	A	S
bsr *dst, src*	src	dst,src,zf	2080	31s	S	N	A	I	N	A	A	A	A	S
cmpxchg *rm32, r32*	eax,rm32,r32	eax,rm32,r32,zf,of,sf,af,cf,pf	9792	2m39s	S	N	E	E	N	E	E	E	E	S
TOTAL			102064	13h52m48s										

How to verify tainting rules

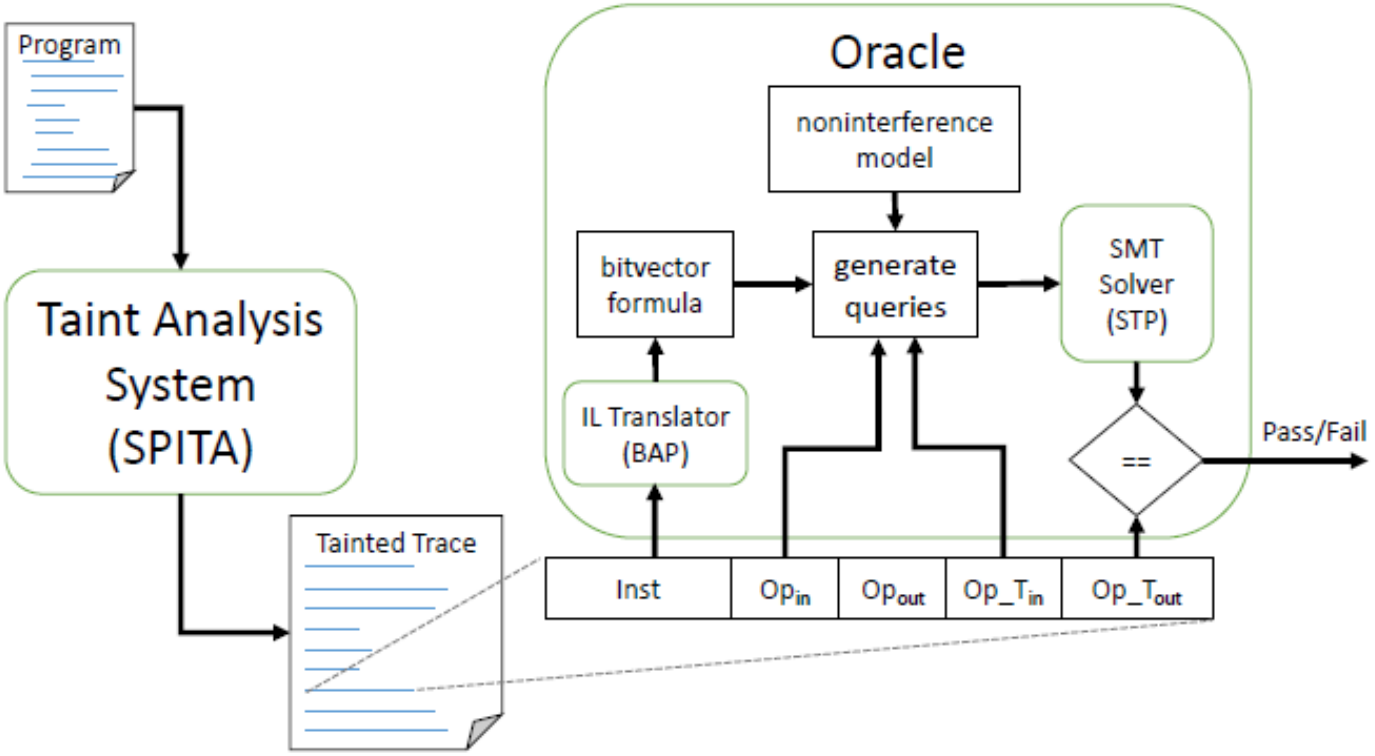


Figure 8: Per-Trace Verification Overview

```
 1. // Query for bit [31] of R_EBX:u32
 2. R_EBX_C:u32 = 0x46018902:u32
 3. R_EBX_T:u32 = 0x56718e20:u32
 4. //Concretization of flags
 5. goal:bool = false
 6. R_EBX:u32 = (R_EBX_O1:u32 & R_EBX_T:u32)
              | (R_EBX_C:u32 & ~R_EBX_T:u32)
 7. R_EBX:u32 = 0:u32 // sets R_EBX to 0
 8. //BAP IR for calculating the flags for xor ebx, ebx
 9. goal1:u32 = R_EBX:u32 & 0x80000000:u32
10. R_EBX:u32 = (R_EBX_O2:u32 & R_EBX_T:u32)
              | (R_EBX_C:u32 & ~R_EBX_T:u32)
11. //Same BAP IR for emulating xor
12. goal2:u32 = R_EBX:u32 & 0x80000000:u32
13. goal:bool = goal1:u32 <> goal2:u32
```

Figure 9: Query to determine whether bit 31 of EBX should be tainted

Table IV: **Comparing SPITA with TEMU on tainted shell commands.** "n / m" indicates that "n" bytes are tainted, and "m" tainted EIPs are observed.

Command	SPITA	TEMU
`dir`	207 / 0	639 / 0
`cd`	146 / 0	616 / 0
`cipher c:`	929 / 0	3617 / 0
`echo hello`	660 / 0	3808 / 0
`find "jone" a.txt`	967 / 0	5684 / 0
`findstr /s /i jone ./*`	945 / 0	1333 / 0
`ls`	350 / 3	34923 / 0
`cd`	306 / 3	301 / 0
`cat ./readme`	545 / 31	26619 / 0
`echo hello`	744 / 9	704 / 0
`ln -s a.txt nbench`	1122 / 35	24707 / 0
`mkdir test`	551 / 9	23766 / 0