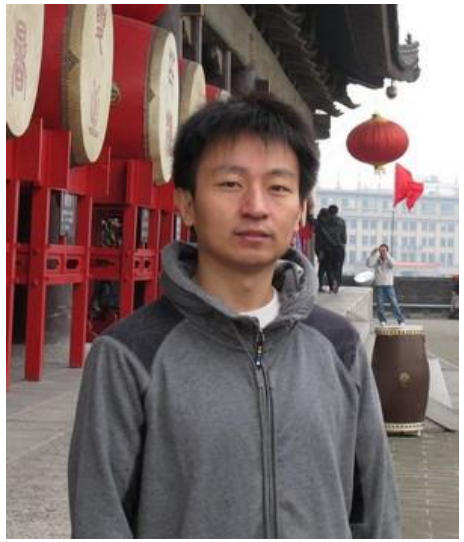
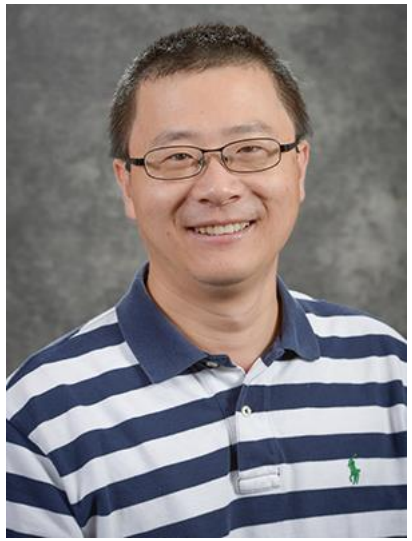
TaintScope: A Checksum-Aware Directed Fuzzing Tool for Automatic Software Vulnerability Detection

单击输入您的封面副标题

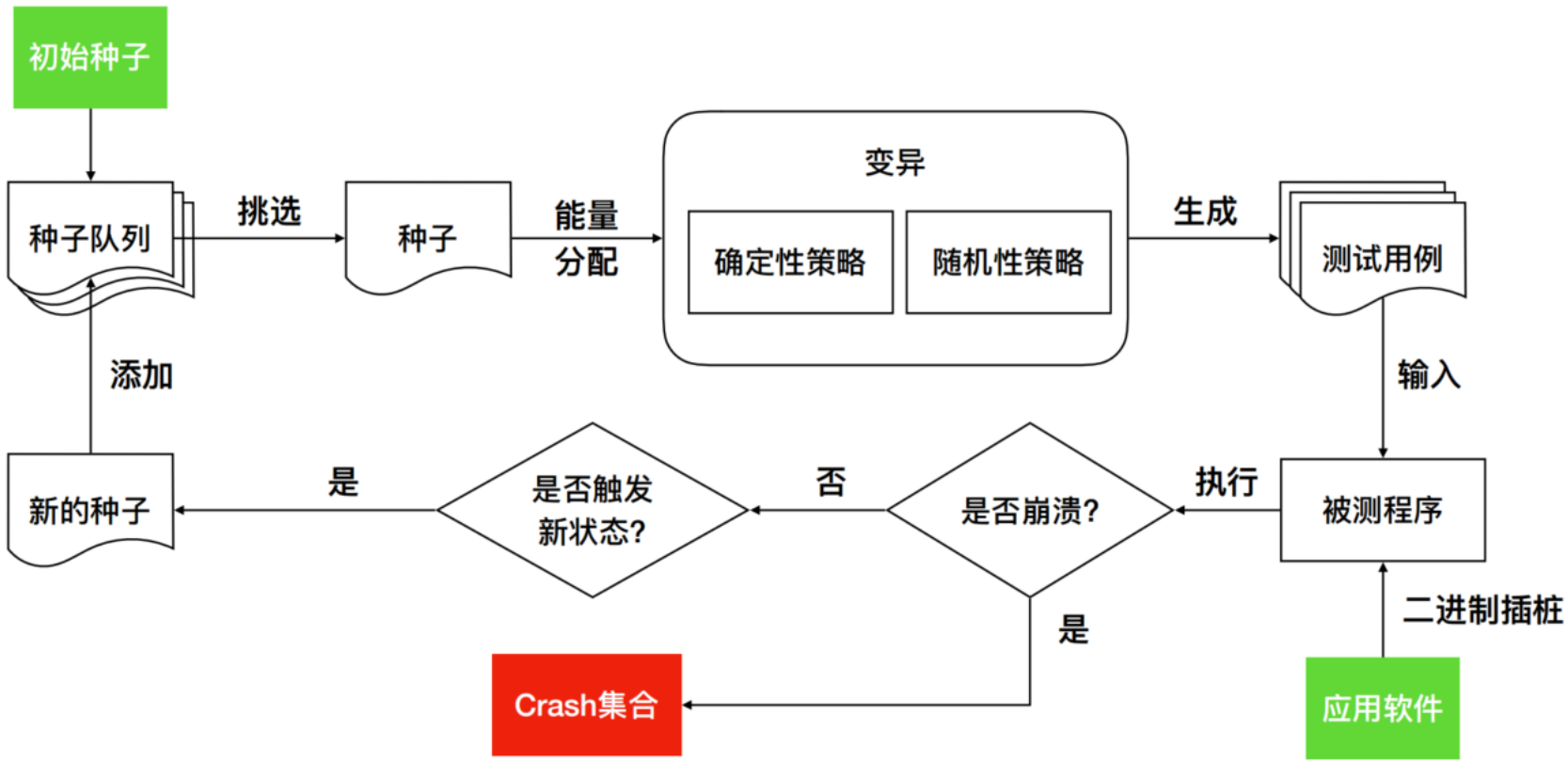
Arthors

Tielei Wang[1,2], Tao Wei[1,2], Guofei Gu[3], Wei Zou[1,2*]
[1]Key Laboratory of Network and Software Security Assurance (Peking University),
Ministry of Education, Beijing 100871, China
[2]Institute of Computer Science and Technology, Peking University
[3] Department of Computer Science & Engineering, Texas A&M University
{wangtielei, weitao, zouwei}@icst.pku.edu.cn, guofei@cse.tamu.edu

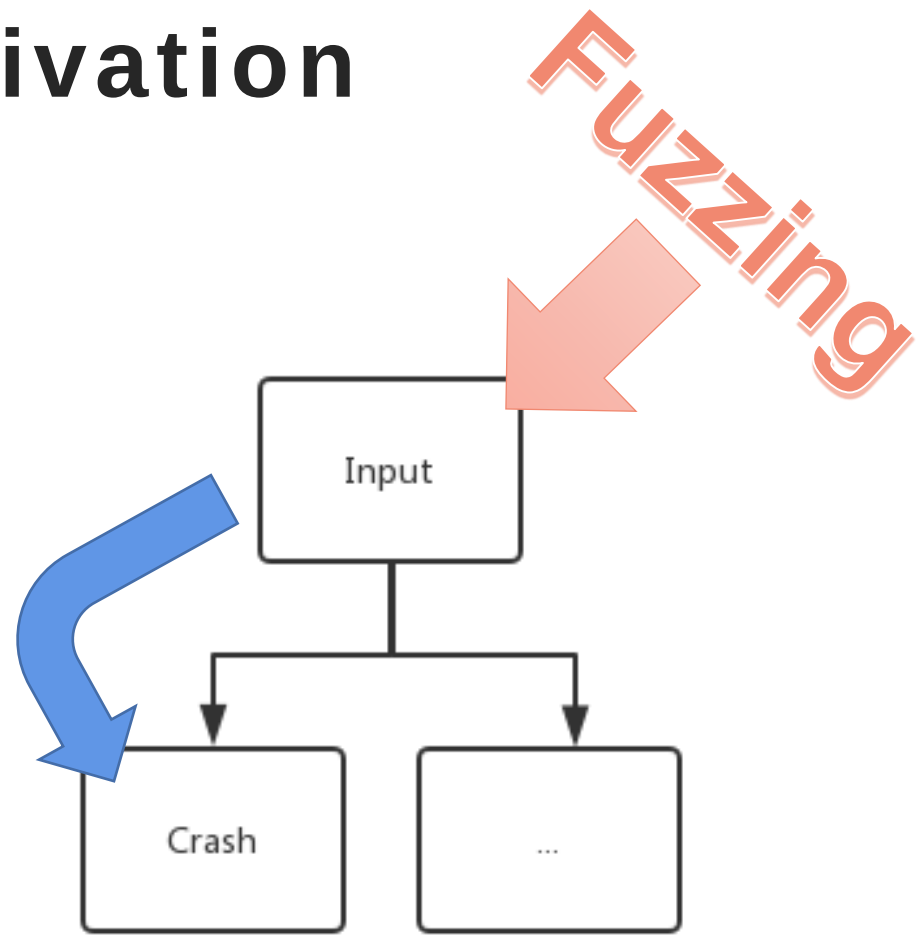




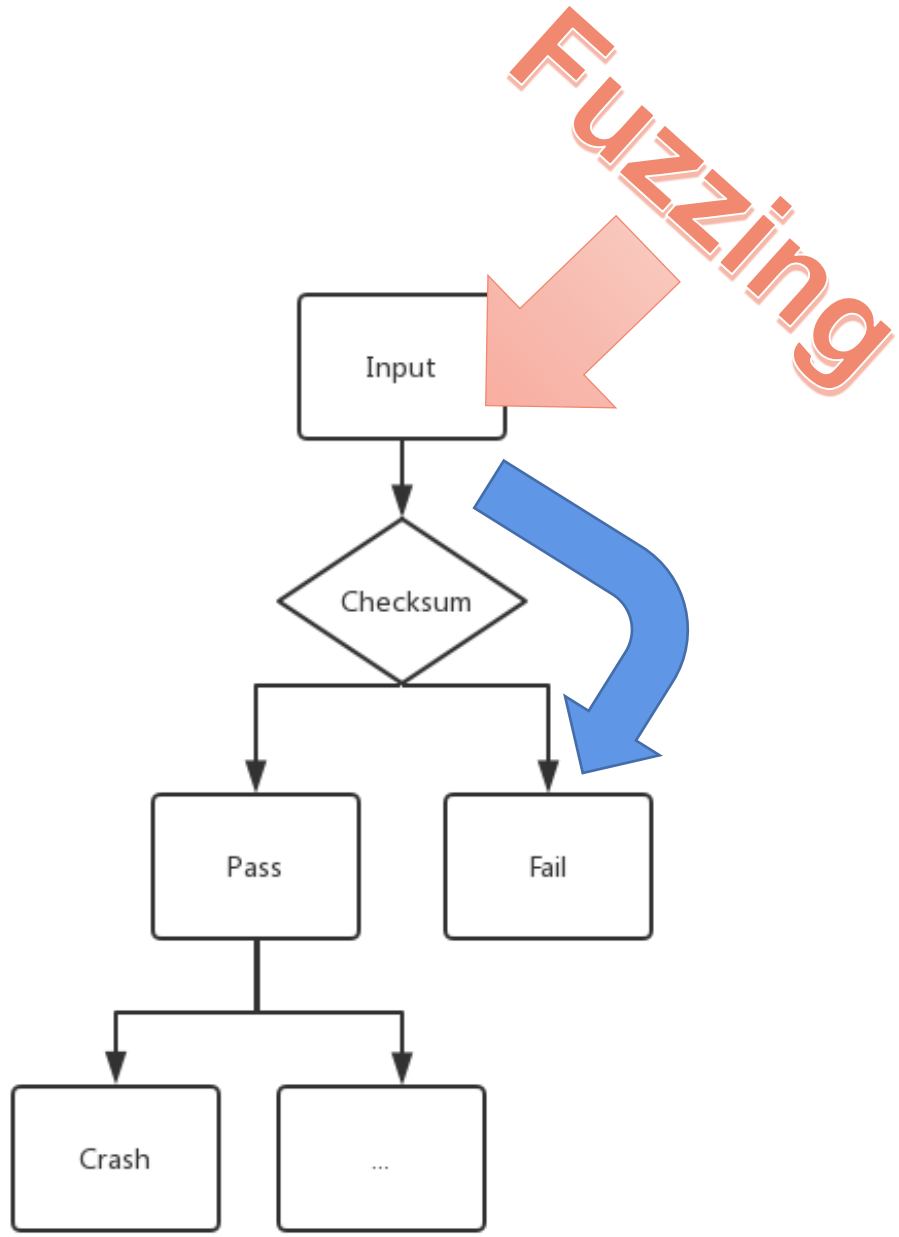
- 以AFL为代表的CGF主要流程



Motivation



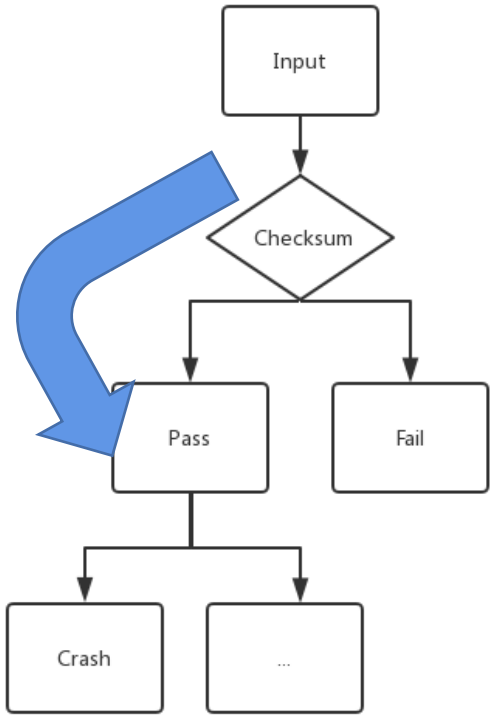
Ideal



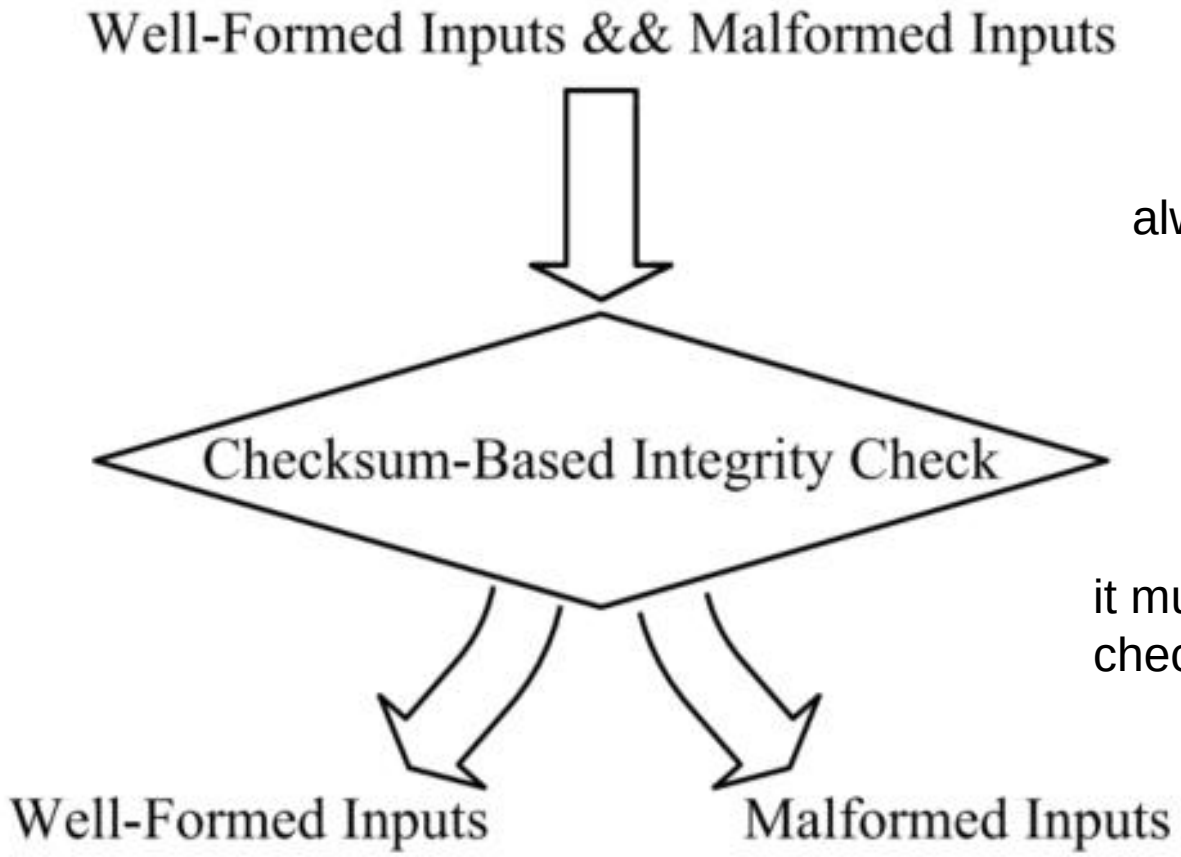
Real World

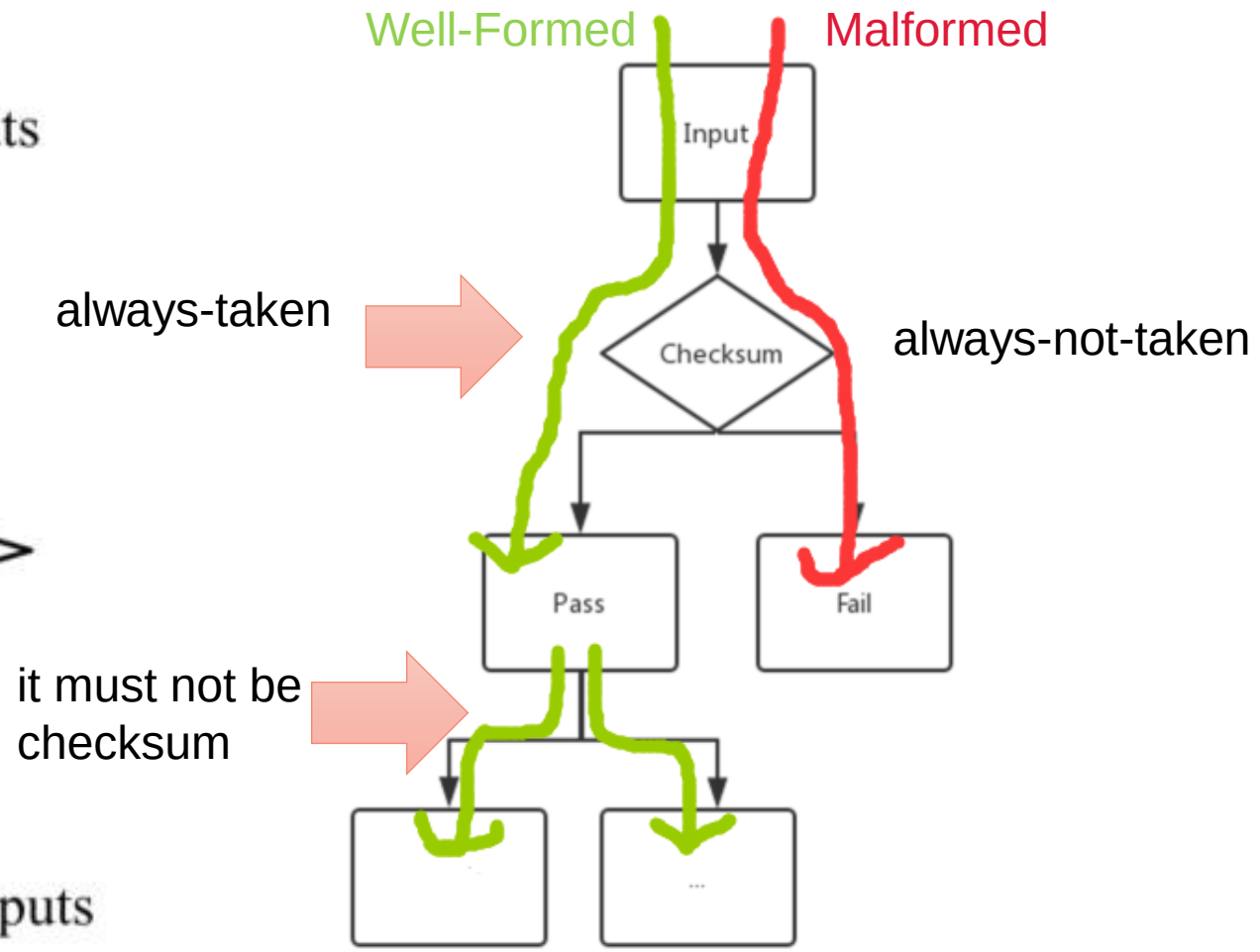
Motivation

- 1. White Box Fuzzing (e.g., Fuzzing Based on Symbolic Execution) cannot solve the constraints of checksum
- 2. Generation-based Fuzzing is unable to reverse engineer the checksum algorithms.
- So, how to bypass the checksum?



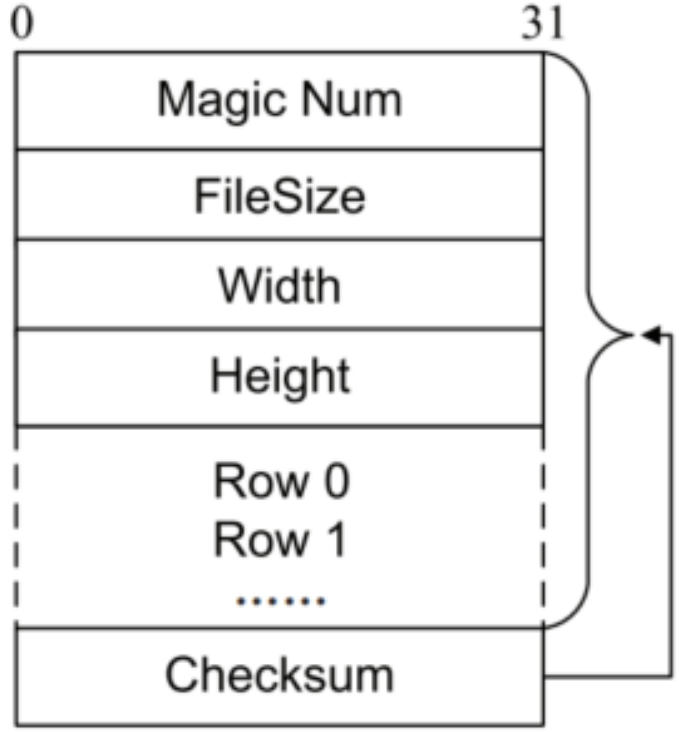
High-Level Intuition



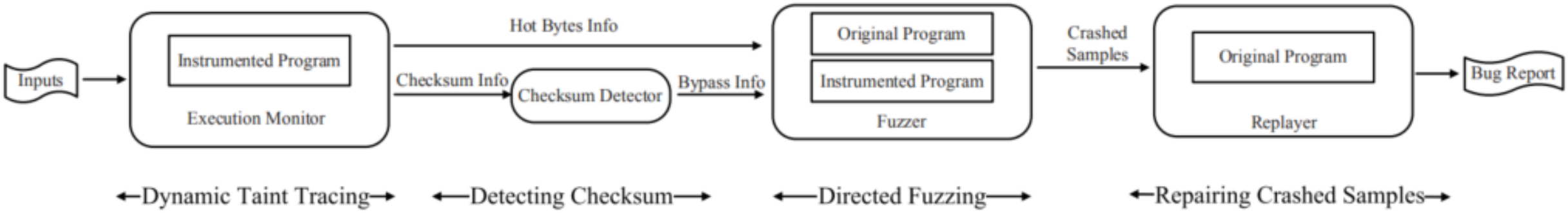


A Motivating Example

```
1  void decode_image(FILE* fd){
2    ...
3    int length              = get_length(fd);
4    int recomputed_chksum = checksum(fd, length);
5    int chksum_in_file     = get_checksum(fd);
     //line 6 is used to check the integrity of inputs
6    if(chksum_in_file != recomputed_chksum)
7      error();
8    int Width     = get_width(input_file);
9    int Height    = get_height(input_file);
10   int size = Width*Height*sizeof(int); //integer overflow
11   int* p    = malloc(size);
12   ...
13   for(i=0; i<Height;i++){// read ith row to p
14     read_row(p + Width*i, i, fd); //heap overflow
```



TaintScope Overview



A. Fine-grained Dynamic Taint Tracing

- Analysts can specify a filename or an IP address as a taint source.
- Data-flow dependency only: data movement instructions (e.g., MOV, PUSH), arithmetic instructions (e.g., SUB, ADD), and logic instructions (e.g., AND, XOR). Also eflag is also taken into consideration.

```
1  void decode_image(FILE* fd){
2   ...
3   int length             = get_length(fd);
4   int recomputed_chksum = checksum(fd, length);
5   int chksum_in_file     = get_checksum(fd);
    //line 6 is used to check the integrity of inputs
6   if(chksum_in_file != recomputed_chksum)
7     error();
8   int Width     = get_width(input_file);
9   int Height    = get_height(input_file);
10  int size = Width*Height*sizeof(int); //integer overflow
11  int* p    = malloc(size);
12  ...
13  for(i=0; i<Height;i++){// read ith row to p
14    read_row(p + Width*i, i, fd); //heap overflow
```

B. Identification of Hot Bytes

- By default, the execution monitor checks which input bytes can pollute the arguments of memory allocation functions (e.g., malloc, realloc) and string functions (e.g., strcpy, strcat).

```
1  void decode_image(FILE* fd){
2   ...
3   int length             = get_length(fd);
4   int recomputed_chksum = checksum(fd, length);
5   int chksum_in_file     = get_checksum(fd);
    //line 6 is used to check the integrity of inputs
6   if(chksum_in_file != recomputed_chksum)
7     error();
8   int Width     = get_width(input_file);
9   int Height    = get_height(input_file);
10  int size = Width*Height*sizeof(int); //integer overflow
11  int* p    = malloc(size);
12  ...
13  for(i=0; i<Height;i++){// read ith row to p
14    read_row(p + Width*i, i, fd); //heap overflow
```

C. Locating Checksum Check Points and Checksum Fields

- There must be a special branch point (e.g., a conditional jump instruction such as JZ, JE) in the program, where all well-formed inputs follow the same branch whereas malformed inputs follow the other one.

- Identifying Potential Checksum Check Points. (all conditional jump instructions)
- Refinement Procedure. ($(P1 \cap P'0) \cup (P0 \cap P'1)$)
- Checksum Field Identification. (back to source)

P: Set of Assertion
P1: Well-formed Input Always Taken
P0: Well-formed Input Always not Taken
P‘1: Malformed Input Always Taken
P’0: Malformed Input Always not Taken

D. Directed and Checksum-aware Fuzzing

- Checksum-aware Fuzzing. To force the conditional jump instructions always (not) taken, the condition code flags (e.g., OF, CF, ZF) in the eflags register are set to proper values before the execution of the branch instructions.

```
1  void decode_image(FILE* fd){
2    ...
3    int length             = get_length(fd);
4    int recomputed_chksum = checksum(fd, length);
5    int chksum_in_file     = get_checksum(fd);
     //line 6 is used to check the integrity of inputs
6    if(chksum_in_file != recomputed_chksum)
7      error();
8    int Width     = get_width(input_file);
9    int Height    = get_height(input_file);
10   int size = Width*Height*sizeof(int); //integer overflow
11   int* p    = malloc(size);
12   ...
13   for(i=0; i<Height;i++){// read ith row to p
14     read_row(p + Width*i, i, fd); //heap overflow
```

E. Repairing Test Cases Using Combined Concrete and Symbolic Execution

- The test case generated cannot pass the checksum!
- So it need to be repaired.

```
 1  void decode_image(FILE* fd){
 2   ...
 3   int length             = get_length(fd);
 4   int recomputed_chksum = checksum(fd, length);
 5   int chksum_in_file     = get_checksum(fd);
     //line 6 is used to check the integrity of inputs
 6   if(chksum_in_file != recomputed_chksum)
 7     error();
 8   int Width    = get_width(input_file);
 9   int Height   = get_height(input_file);
10   int size = Width*Height*sizeof(int); //integer overflow
11   int* p   = malloc(size);
12   ...
13   for(i=0; i<Height;i++){// read ith row to p
14     read_row(p + Width*i, i, fd); //heap overflow
```

Evaluation

- .

Category	Application	Version	OS	Category	Application	Version	OS
Image Viewer	Google Picasa	3.1.0	Windows	Media Player	MPlayer	SVN-28979	Linux
	Adobe Acrobat	9.1.3	Windows		Gstreamer	0.10.15	Linux
	ImageMagick	6.5.2-7	Linux		Winamp	5.552	Windows
	Microsoft Paint	5.1	Windows	Other	libtiff	3.8.2	Linux
Web Browser	Amaya	11.1	Windows		XEmacs	21.4.22	Linux
	Dillo	2.1.1	Linux		wxWidgets	2.8.10	Linux

Table I
AN INCOMPLETE LIST OF APPLICATIONS USED IN OUR EXPERIMENT

Executable	Package	Input Format	Input Size (Bytes)	# Hot Bytes	# X86 Instrs	Run Time
Display	ImageMagick	TIFF	5778	18	191,759,211	2m53s
			2,020	18	82,640,260	1m30s
		PNG	5,149	9	19,051,746	1m54s
			1,250	29	47,246,043	1m8s
		JPEG	6,617	11	48,983,897	1m13s
			6,934	9	48,823,905	1m11s
PicasaPhotoViewer.exe	Google Picasa	GIF	3,190	14	304,993,501	1m25s
			6,529	43	536,938,567	2m57s
		PNG	2,730	18	712,021,776	5m16s
			1,362	16	660,183,239	4m8s
		BMP	3,174	8	310,909,256	1m21s
			7,462	19	468,273,580	2m35s
Acrobat.exe	Adobe Acrobat	BMP	1,440	6	658,370,048	4m25s
			3,678	6	663,923,080	5m2s
		PNG	770	21	297,492,758	3m8s
			1,250	12	354,685,431	4m31s
		JPEG	1,012	13	328,365,912	4m14s
			2,356	4	356,136,453	4m36s

Table II
HOT BYTES IDENTIFICATION RESULTS

Evaluation

Executable	Package (Version)	File Format	Checksum Algorithm	$\lvert\mathcal{A}\rvert$	$\lvert(\mathcal{P}_1 \cap \mathcal{P}'_0) \cup (\mathcal{P}_0 \cap \mathcal{P}'_1)\rvert$	Detected?
PicasaPhotoViewer	Google Picasa (3.1)	PNG	CRC32	830	1	√
Acrobat	Adobe Acrobat (9.1.3)			5,805	1	√
Snort	snort (2.8.4.1)	PCAP	TCP/IP checksum	2	2	√
tcpdump	tcpdump (4.0.0)			5	2	√
sigtool	clamav (0.95.2)	CVD	MD5	2	1	√
vcdiff	open-vcdiff (0.6)	VCDIFF	Adler32	1	1	√
Tar	GNU Tar (1.22)	Tar Archive	Tar checksum	9	1	√
objcopy	GNU binutils (2.17)	Intel HEX	Intel HEX checksum	62	1	√

Table III
CHECKSUM IDENTIFICATION RESULTS

Evaluation

- 单击此

Package	Vuln-Type	# Vulns	Checksum-aware?	Advisory	Severity Rating
Microsoft Paint	Memory Corruption	1	N	CVE-2010-0028	Moderate
Google Picasa	Infinite loop	1	N	pending	N/A
	Integer Overflow	1		SA38435	Moderate
Adobe Acrobat	Infinite loop	1	N	CVE-2009-2995	Extremely critical
	Memory Corruption	1	N	CVE-2009-2989	Extremely critical
ImageMagick	Integer Overflow	1	N	CVE-2009-1882	Moderate
CamlImage	Integer Overflow	3	**Y**	CVE-2009-2660	Moderate
LibTIFF	Integer Overflow	2	N	CVE-2009-2347	Moderate
wxWidgets	Buffer Overflow	2	N	CVE-2009-2369	Moderate
	Double Free	1	**Y**		
IrfanView	Integer Overflow	1	N	CVE-2009-2118	High
GStreamer	Integer Overflow	1	**Y**	CVE-2009-1932	Moderate
Dillo	Integer Overflow	1	**Y**	CVE-2009-2294	High
XEmacs	Integer Overflow	3	**Y**	CVE-2009-2688	Moderate
	Null Dereference	1	N	N/A	N/A
MPlayer	Null Dereference	2	N	N/A	N/A
PDFlib-lite	Integer Overflow	1	**Y**	SA35180	Moderate
Amaya	Integer Overflow	2	**Y**	SA34531	High
Winamp	Buffer Overflow	1	N	SA35126	High
Total		27			

Table V
VULNERABILITIES DETECTED BY TAINTSCOPE