Building security predicates for some types of vulnerabilities*

A.N. Fedotov, V.V. Kaushan, S.S. Gaissaryan , Sh.F. Kurmangaleev

{fedotoff, korpse, ssg, kursh}@ispas.ru

*supported by RFBR grant № 17-01-00600 A

Motivation

Static and dynamic analysis (fuzzing) are used in industrial software development.

Vulnerabilities leading to arbitrary code are most dangerous.

Problems in exploitability estimation of program bugs:

- Industrial fuzzers could produce lots of crashes:
  - Miller C. et al. Crash analysis with BitBlaze //at BlackHat USA. – 2010.
  - Godefroid P., Levin M. Y., Molnar D. SAGE: whitebox fuzzing for security testing //Queue. – 2012. – T. 10. – №. 1. – C. 20.
  - Bounimova E., Godefroid P., Molnar D. Billions and billions of constraints: Whitebox fuzz testing in production //Proceedings of the 2013 International Conference on Software Engineering. – IEEE Press, 2013. – C. 122-131.

- Exploitation hardening mechanisms in modern OS and compilers.

The goal is to assess the quality of the protective mechanisms being developed

Approaches 1/2

Crash analysis is based on an estimation of the program state (values of registers and memory cells, signal numbers) at crash point.

Pros:

- fast and simple;
- high accuracy in the determination of unexploited crashes

(null pointer dereference, division by zero, safe functions and canaries).

Cons:

- lack of Exploit.

Tools: !exploitable (Microsoft), gdb exploitable plugin (cert), CrashFilter.

Approaches 2/2

Automatic exploit generation.

Pros:

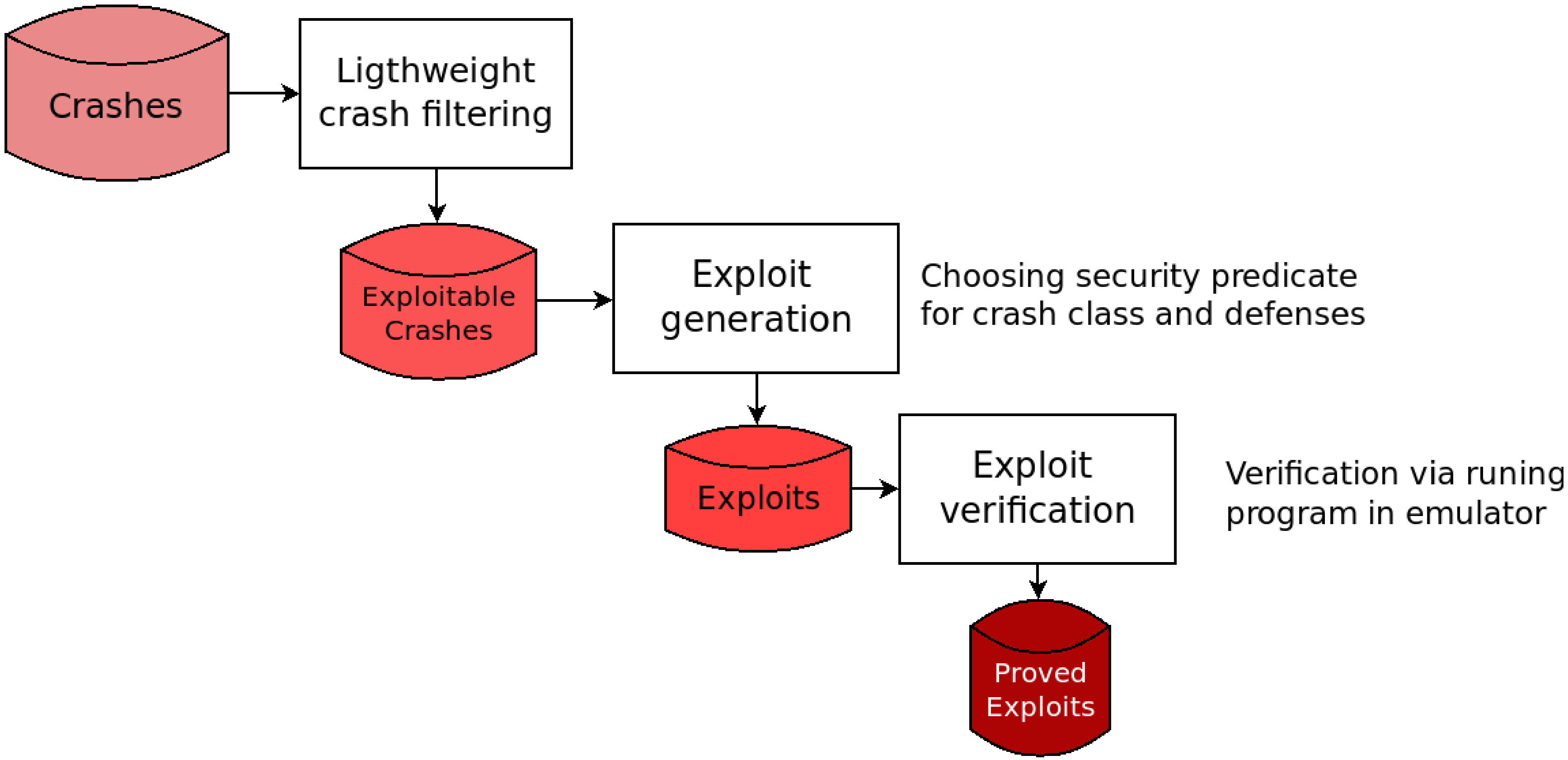
- Exploit.
- No false positive (in case of exploit verification).

Cons:

- Symbolic execution produce the large overhead.

Tools: AEG (MAYHEM), CRAX, REX.

Method for exploitability estimation of program bugs



Crash filtering

The goal is to filter non-exploitable crashes (null pointer dereference, division by zero, safe function and canaries etc.)

Total crash classes : 17

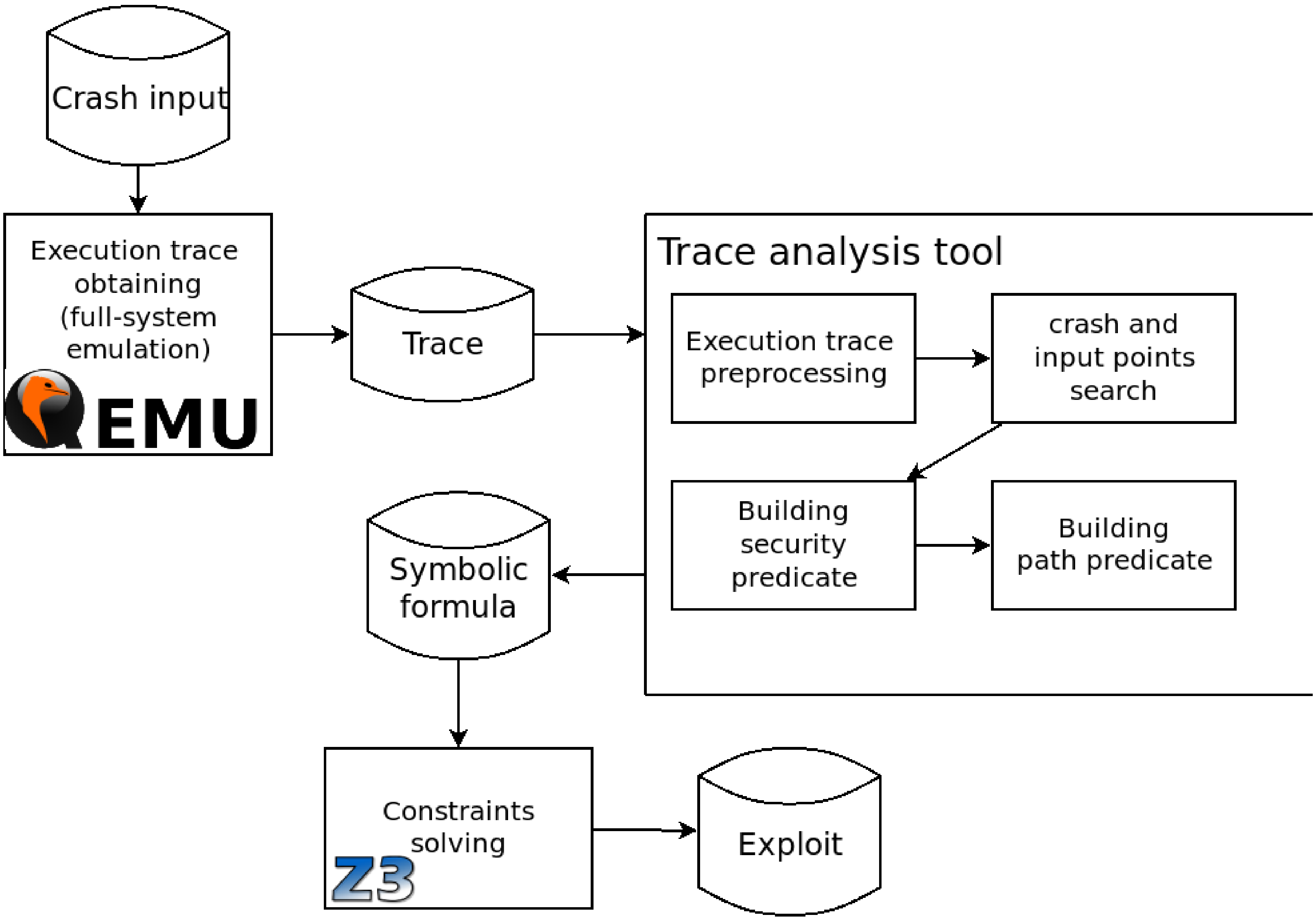
Exploitable *classes*: 4

- Memory access violation on program counter;
- Memory access violation on control flow transfer instruction (CALL/JUMP);
- Memory access violation on return instruction (RET);
- Memory access violation on store instruction (CWE-123);

Non-exploitable classes: 13

Method is based on *DynamoRIO.*

Automatic exploit generation



Input points search

All types of input (network, command line arguments, environment variables, files, stdin) represented via files

Comand line: ./vuln $(cat seed.bin)

Trace

Process cat

handle = open("seed.bin")

buf = read(handle)

close(handle)

Source:buf

Process vuln

main (int argc, int **argv)

Sink:argv[1]

Crash search in trace

Search for violation of normal program execution

- A violation of execution is an interrupt
- Consideration of interruptions from which there was no return in the trace
- Consideration of control flow transferring or writing to memory instructions
- Check if instruction operands are tainted

Building path and security predicates

Building path predicate is based on:

- taint analysis
- translation to intermediate representation (Pivot)
- interpretation of Pivot-code
- building symbolic formulas from Pivot instructions.

Security predicate goals:

1. Describe location of payload in memory.
2. Describe control flow transfer of control to payload.

Building security predicate depends on crash class and defense mechanisms (DEP, ASLR) to bypass.

Crash classification in execution trace

Crash classification in execution trace clarifies crash filtering classification for exploitable crashes because of taint analysis:

1. Memory access violation on return instruction. Stack pointer tainted. DEP bypass.
2. Memory access violation on return instruction. Return address value is tainted. DEP & ASLR bypass.
3. Memory access violation on control flow transfer instruction. Operand memory address is tainted. Example: **CALL DWORD:PTR[EAX].** DEP bypass.
4. Memory access violation on control flow transfer instruction. Target address is tainted. Example: **CALL EAX.** DEP & ASLR bypass.
5. Memory access violation on store instruction. Source operand and destination address are tainted (CWE-123). DEP & ASLR bypass.

ИСП РАН

Defenses bypass

- ASLR bypass – trampolines (CALL/JUMP REG)
- DEP / DEP & ASLR bypass – ROP
  - GOT-slot attack for CWE-123 (5th crash) in Linux programs

For some crash classes (except 1st and 2nd) needed special gadgets: gadget trampolines.

Shift stack*:* shifts stack pointer on constant value.

add esp, 42; ret.

Arithmetic stack*:* shifts stack pointer on register value.

add esp, eax; ret.

Stack pivot*: moves register value to stack pointer*

mov esp,eax; ret.

Exploitability estimation of bugs gained from fuzzing Debian 6.0.10

Crash class groups	Crash class	Crash count
Exploitable	Memory access violation on program counter	13
Not exploitable	Heap error	23
Not exploitable	Memory access violation	238

Total crashes: 274. All 13 exploitable crashes are exploited. 5 crashes are exploited with DEP enabled. 1 crash is exploited with DEP and ASLR enabled.

Exploit generation for crashes gained from public sources

Windows 32-bit XP

AudioCoder (DEP), VuPlayer (DEP), Pcman, 3proxy, CoolPlayer.

Linux 32-bit

Torque-server (DEP & ASLR), nullhttpd и etc.

Linux 64-bit

Mkfs.jfs, faad,dvips

Exploitability estimation for test cases from DARPA CGC 2016

Programs were ported for Linux. Manual crash input search.

Exploitable crashes found:

- **Bloomy_Sunday** (Verification exploit fail)
- **Charter** (Memory access violation on store instruction with untainted source operand)
- ***Movie_Rental_Service*** (Exploited use after free)
- **Multi_User_Calendar** (Exploited stack buffer overflow)
- **Palindrome** (Exploited stack buffer overflow)
- **PKK_Steganography** (Exploited stack buffer overflow)
- **Sample_Shipgame** (Exploited stack buffer overflow)
- **ValveChecks** (Exploited stack buffer overflow)

ИСП РАН

Thank you!

Questions?