Session 10

Fuzzing

Security of Information Systems (SIS)

Computer Science and Engineering Department

December 4, 2024

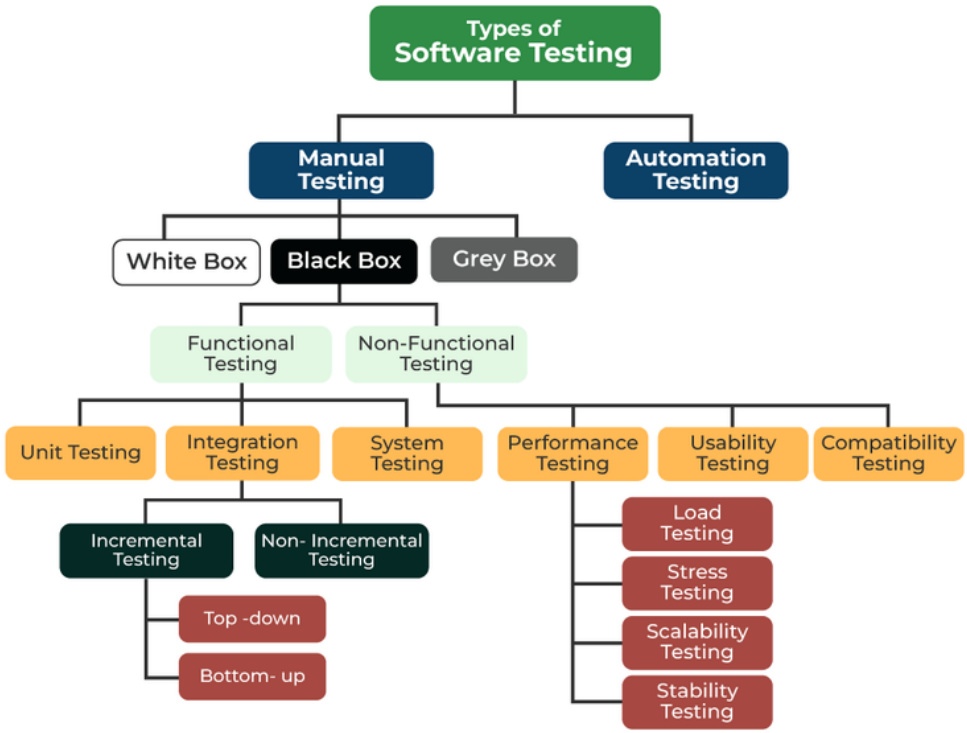
Papers

- kAFL: Hardware-Assisted Feedback Fuzzing for OS Kernels
- Dissecting American Fuzzy Lop: A FuzzBench Evaluation

Software Testing

- validating implementation
- “oracles”: specification, contracts
- find bugs, vulnerabilities

Types of Testing



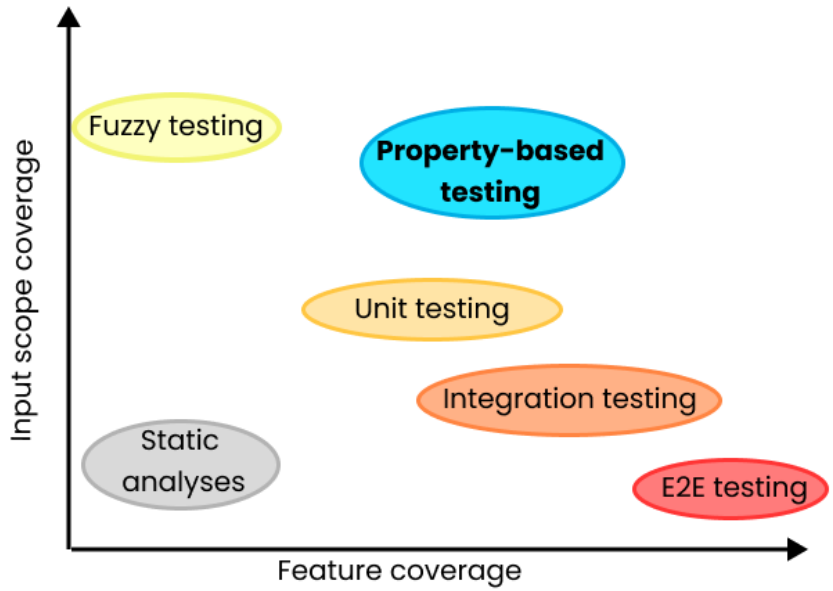
Blackbox / Interface Testing

- do not care / know the internals
- validate input according to specification
- send input, validate output
- similar approach used by fuzzing

Property-based Testing

- define property for functions
- generate inputs
- use inputs and "shrink" to find issues

Property-based Testing vs. Fuzzing



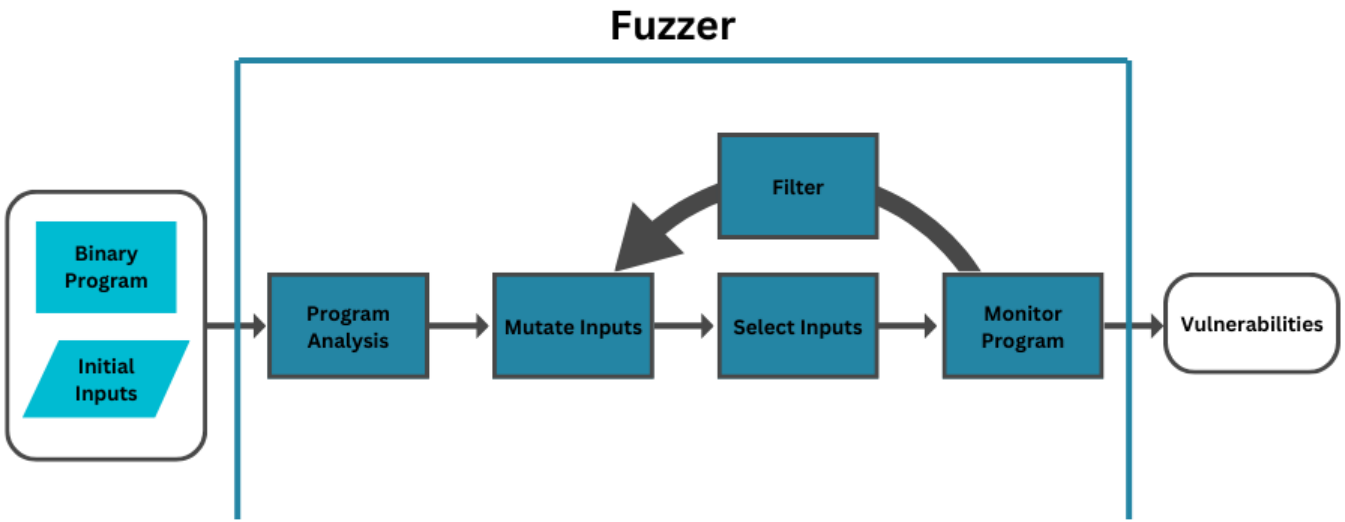
Fuzzing (as testing)

- send quasi-random inputs
- collect results and failures
- do automatically
- have feedback loop based on results / behavior
- `https://github.com/google/fuzzing/blob/master/docs/glossary.md`

Pre and Post Fuzzing

- prepare initial set of inputs (if any)
- do "harnesses" (see later)
- analyze output (crashes)
- fix program

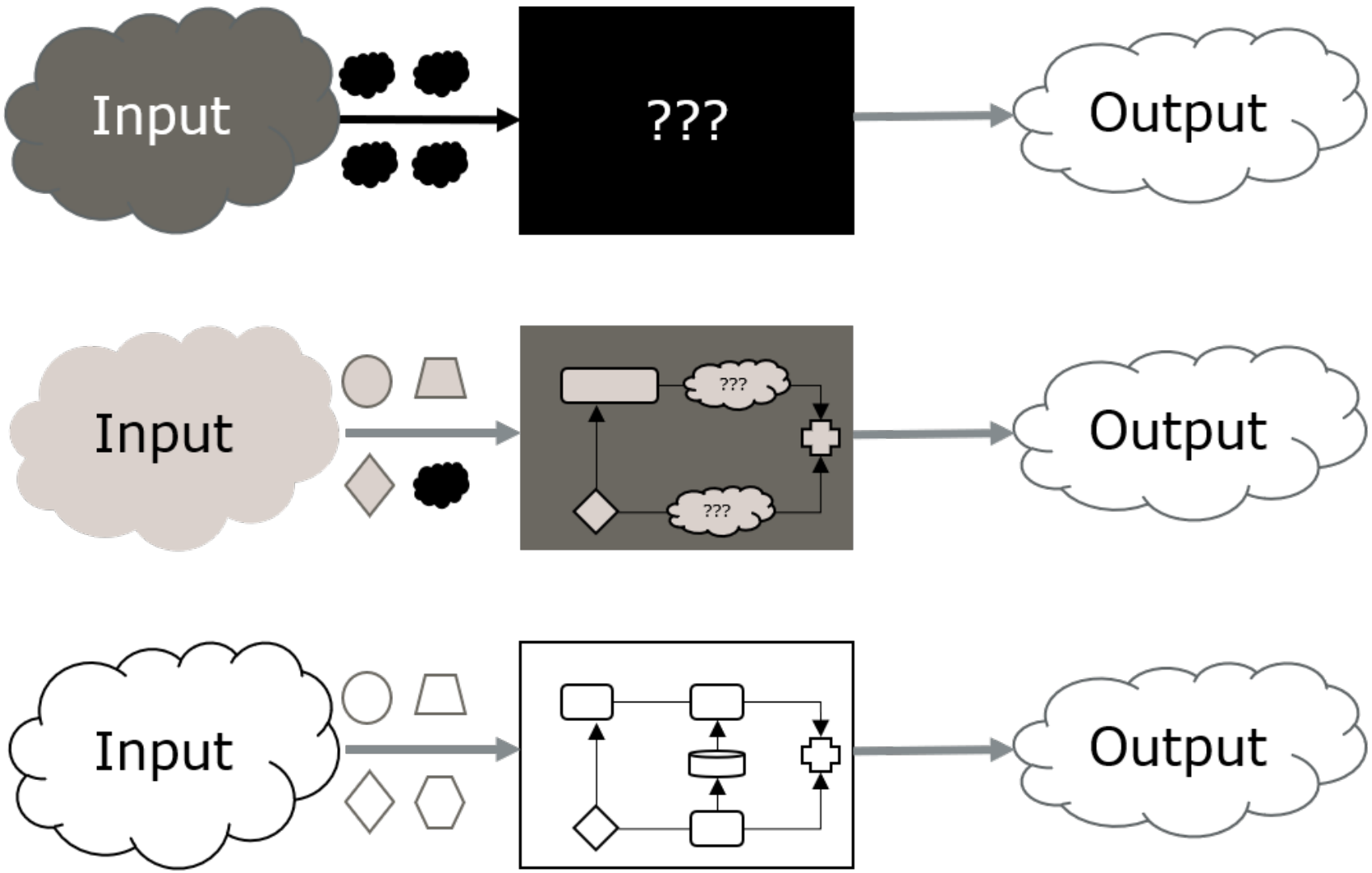
Fuzzer Architecture



Dumb vs Smart Fuzzing

- dumb: pure random data
- smart: know data format
- dump: simpler, but random
- smart: more complex

Blackbox vs Whitebox Fuzzing



Fuzzer Harness

- a testing harness
- an entry in the program executable
- update the code to provide input

Mutation vs Generational Fuzzing

Mutation vs Generation

Mutation-based	Super easy to setup and automate +	Little to no protocol knowledge required +	Limited by initial corpus −	May fail for protocols with checksums, or other complexity −
Generation-based	Writing generator is labor intesive for complext protocols −	have to have spec of protocol (frequently not a problem for common ones http, snmp, etc...) −	Completeness +	Can deal with complex checksums and dependencies +

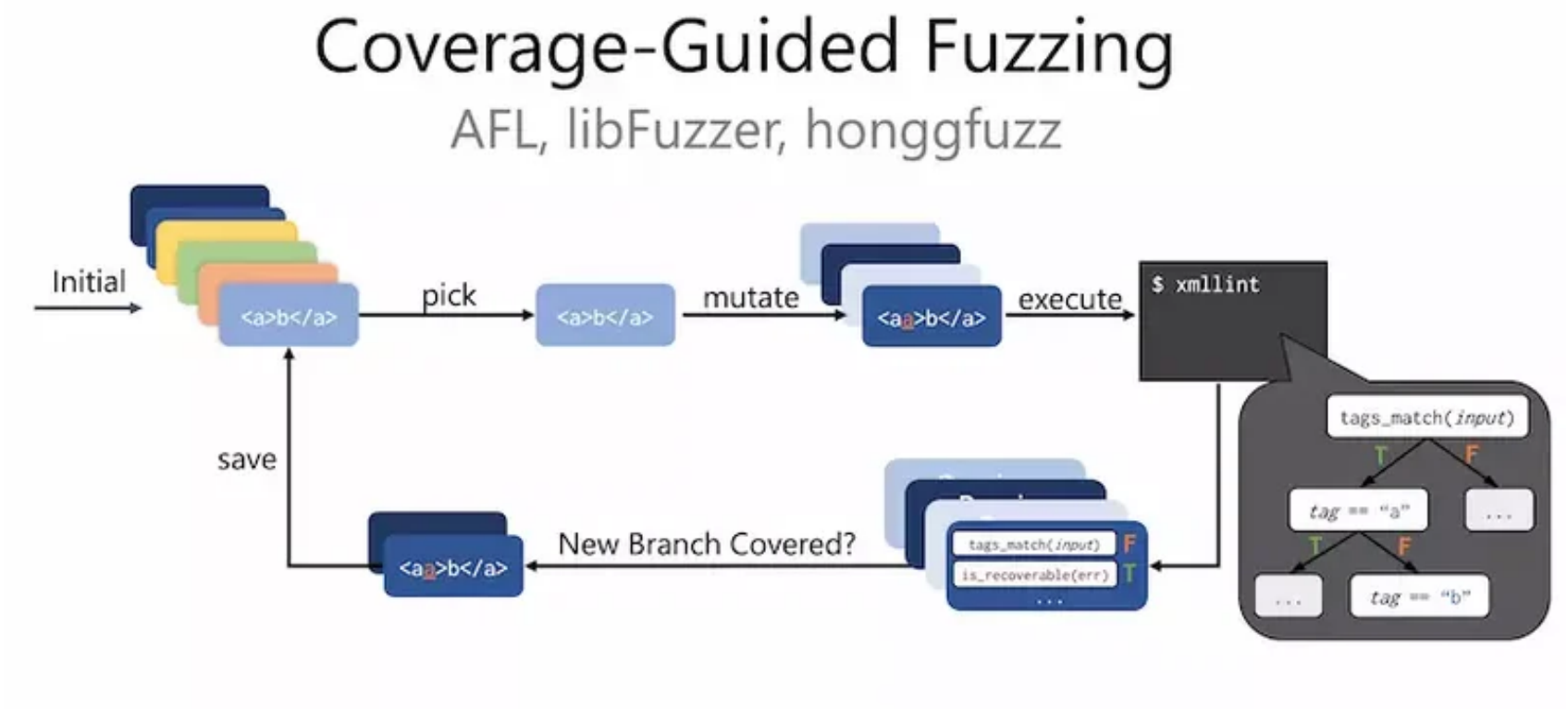
Coming up: What is Fuzzing?

11

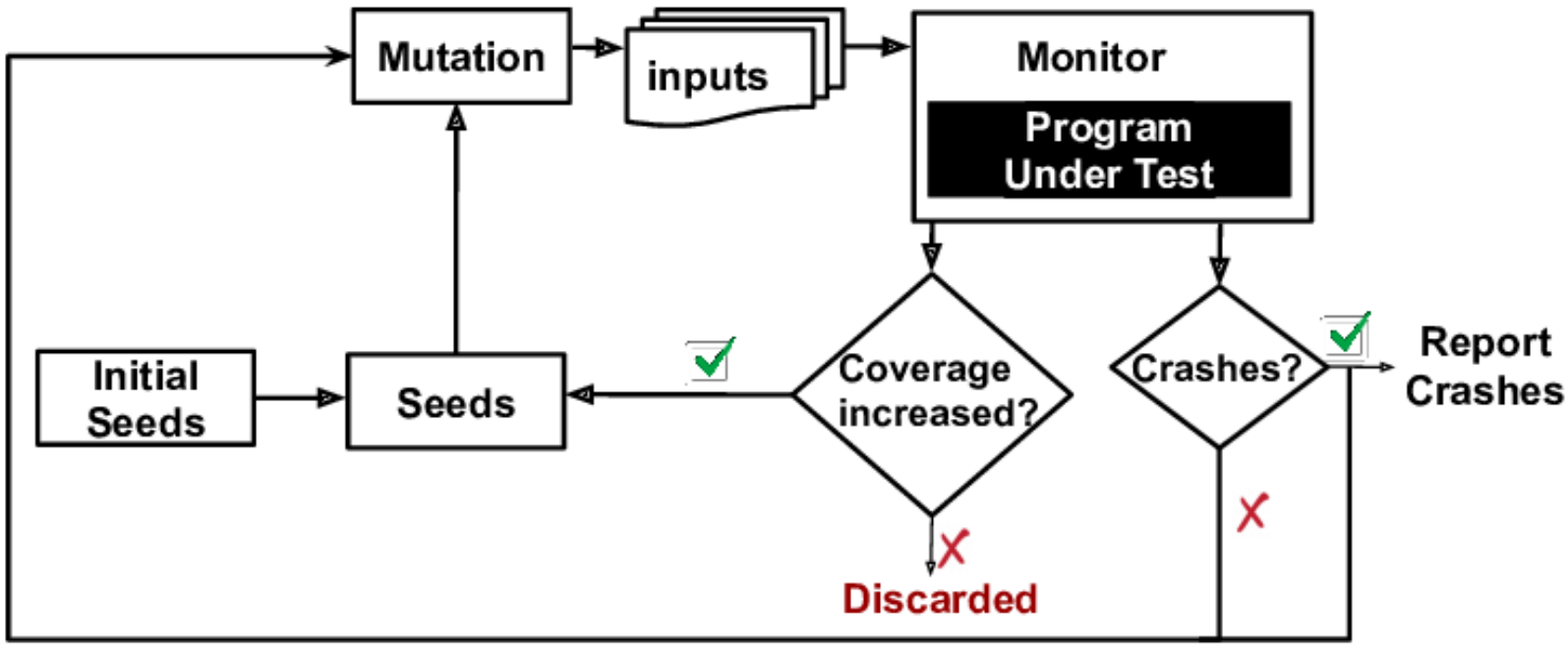
Payload Fuzzing vs API Fuzzing

- payload: create input, send input to program, capture output
- program has input functions: standard input (generally), files, socket
- API fuzzing: library, input is sent via calls (structures)
- do API call, receive output
- analyze crashes

Coverage-based Fuzzing



Coverage-based Fuzzing Process



GCOV, KCOV

- GNU Coverage tool: instruments source code, at build-time
- KCOV: coverage for the Linux kernel, exposed via debugfs
- both enabled at build time
- also bcov, lcov

Runtime Instrumentation Coverage

- can also work on binary code
- intrument programs at runtime
- slower than build time instrumentation
- no need to rebuild

Using Sanitizers

- for fuzzing builds
- force crashes to appear
- without sanitizers, some crashes may not appear
- ASan, KASAN

Fork Server

- start new instances per input
- quick start of new instances
- working as a "harness" to quickly get an application to serve a new input
- can parallelize runs

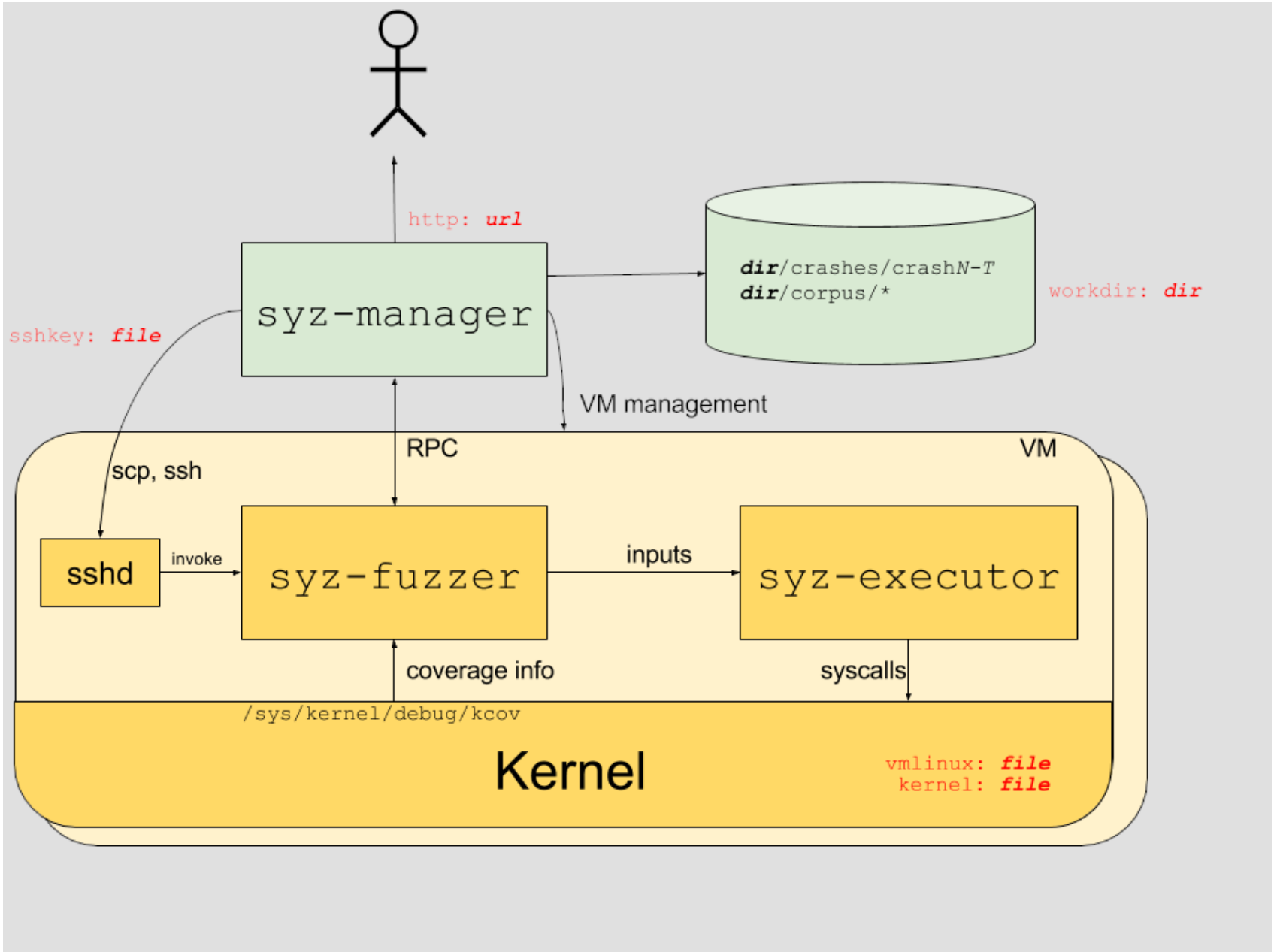
API Fuzzing

- instead of a payload, construct arguments (structures)
- it's similar to a set of payloads
- make call to function with arguments, instead of sending payload
- receive function return value, instead of looking at output
- still aim to capture crashes

OS Fuzzing

- fuzz the OS syscall interface
- run inside the virtual machine
- have some sort of manager interface to capture crashes and restart virtual machines
- how to capture coverage?

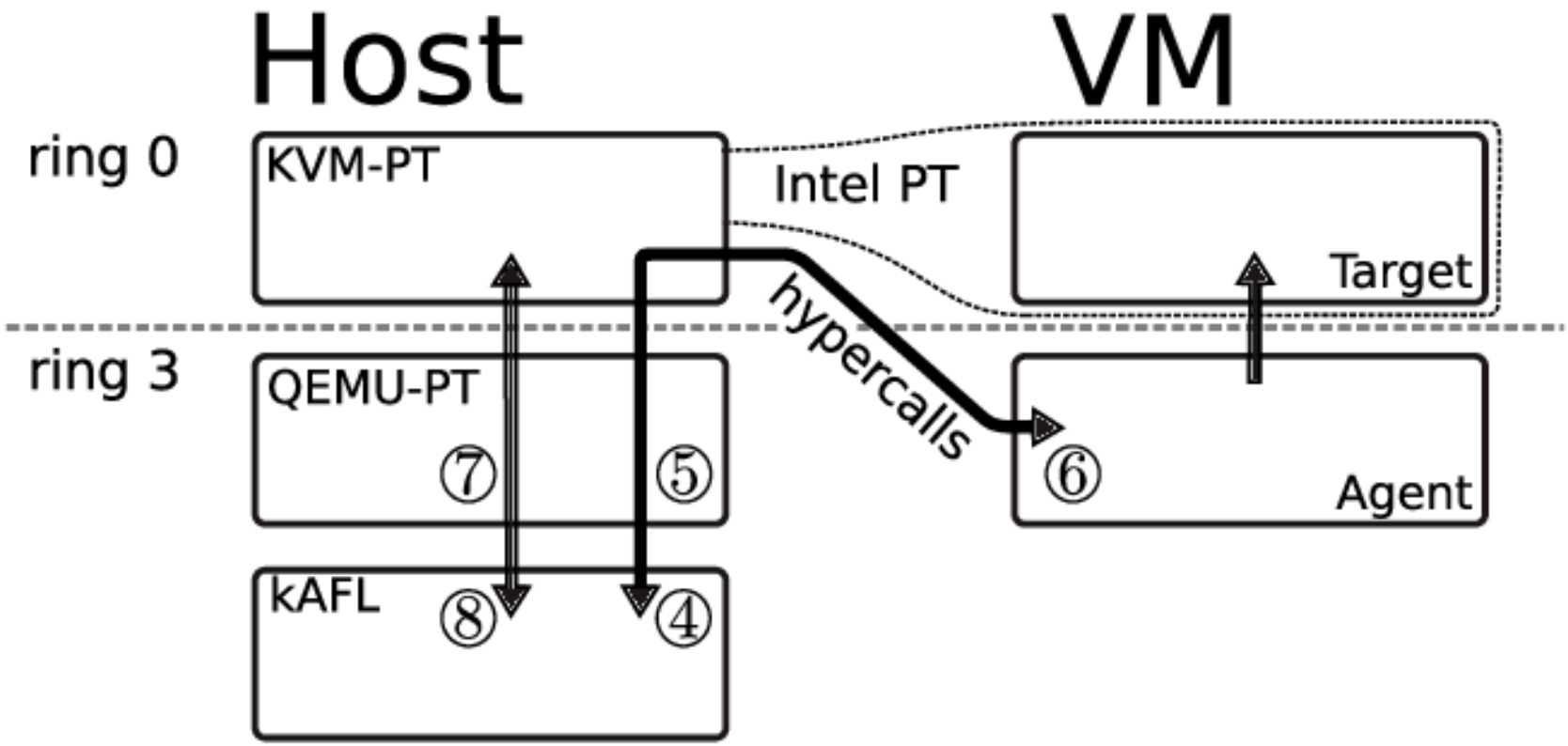
Syzkaller


http: url
syz-manager
dir/crashes/crashN-T
dir/corpus/*
workdir: dir
sshkey: file
VM management
RPC
VM
scp, ssh
sshd
invoke
syz-fuzzer
inputs
syz-executor
coverage info
syscalls
/sys/kernel/debug/kcov
Kernel
vmlinux: file
kernel: file

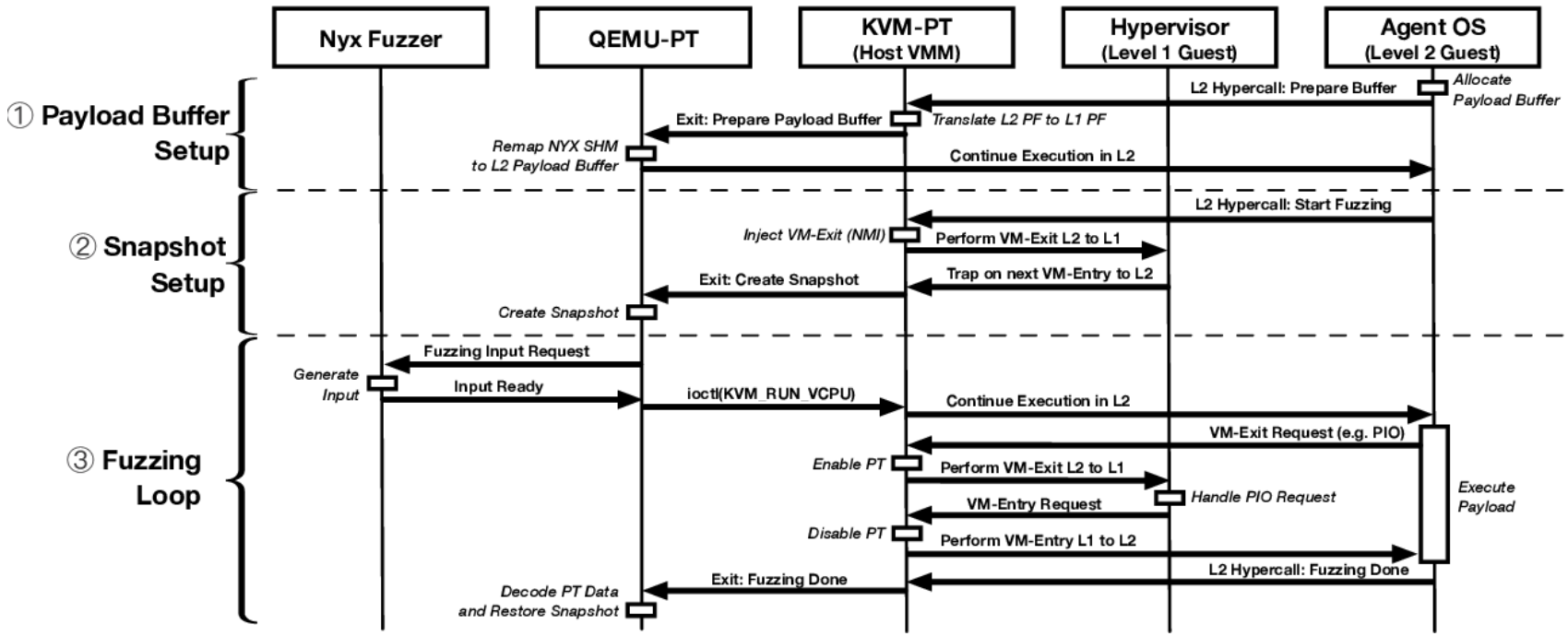
kAFL / Nyx

- KVM-based
- use Intel PT (processor trace for coverage)
- no build-level coverage part required

kAFL


Host
VM
ring 0
KVM-PT
Intel PT
Target
ring 3
QEMU-PT
hypercalls
⑦
⑤
⑥
Agent
kAFL
⑧
④

Nyx


Nyx Fuzzer
QEMU-PT
KVM-PT
(Host VMM)
Hypervisor
(Level 1 Guest)
Agent OS
(Level 2 Guest)
① Payload Buffer Setup
② Snapshot Setup
③ Fuzzing Loop
L2 Hypercall: Prepare Buffer
Allocate Payload Buffer
Exit: Prepare Payload Buffer
Translate L2 PF to L1 PF
Remap NYX SHM to L2 Payload Buffer
Continue Execution in L2
L2 Hypercall: Start Fuzzing
Inject VM-Exit (NMI)
Perform VM-Exit L2 to L1
Exit: Create Snapshot
Trap on next VM-Entry to L2
Create Snapshot
Fuzzing Input Request
Generate Input
Input Ready
ioctl(KVM_RUN_VCPU)
Continue Execution in L2
VM-Exit Request (e.g. PIO)
Enable PT
Perform VM-Exit L2 to L1
Handle PIO Request
VM-Entry Request
Execute Payload
Disable PT
Perform VM-Entry L1 to L2
L2 Hypercall: Fuzzing Done
Exit: Fuzzing Done
Decode PT Data and Restore Snapshot

k/FX

- `https://github.com/intel/kernel-fuzzer-for-xen-project`
- kernel fuzzer
- uses Xen, VMI, AFI

Modern Fuzzers

- AFL
- honggfuzz
- libfuzzer
- OSS Fuzz: `https://github.com/google/oss-fuzz`, `https://google.github.io/oss-fuzz/`
- Google Fuzzbench: `https://github.com/google/fuzzbench`, `https://google.github.io/fuzzbench/`

Cyber-Reasoning Systems (CRS)

- fully autonomous
- analyze a system, detect vulnerabilities, create exploits, self healing
- combines fuzzing, symbolic execution and other techniques
- DARPA CyberGrand Challenge 2016: `https://www.darpa.mil/program/cyber-grand-challenge`

Keywords

- TODO

- TODO

Resources

- TODO

References

- TODO