Study on Kernel Vulnerability Discovery

Chao Zhang
Tsinghua University

About Me

- Experience
  - **Tsinghua University**, Assoc. Prof., 2016/11-present
  - **UC Berkeley**, Postdoc, 2013/9-2016/9, **Advisor: Dawn Song**
  - **Peking University**, Ph.D., 2008/9-2013/7, **Advisors: 邹维，韦韬**
  - **Peking University**, B.S., 2004/9-2008/7, Math
- Honors
  - Thousand Youth Talents Plan（青年千人）
  - Young Elite Scientists Sponsorship Program by CAST
  - Young Talent Development Program by CCF
- Awards
  - **DARPA CGC**, Captain of Team CodeJitsu, Defense #1 in 2015, Attack #2 in 2016
  - **Microsoft BlueHat** Prize Contest 2012, Special Recognition Award
  - **DEFCON CTF**, 2015 (#5), 2016 (#2), 2017 (#5)
  - GeekPwn, 2017/5/12

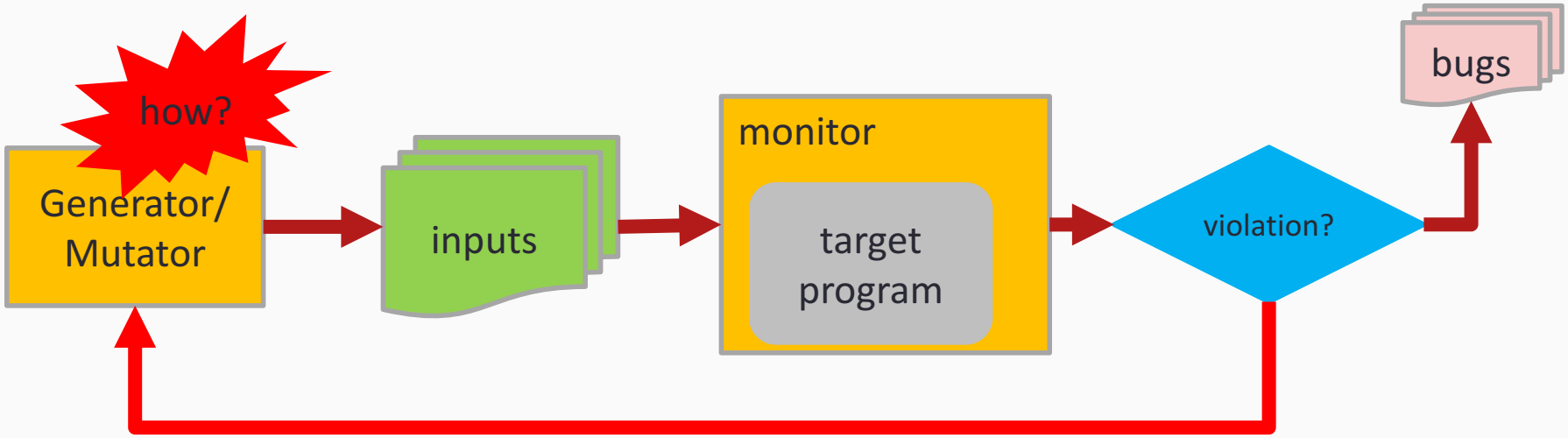
Consequences of Vulnerabilities


Ooops, your files have been encrypted!
POLICE

Vulnerability Discovery

- Code Review *(10%?)*
- Static Analysis
- Dynamic Analysis
  - online
  - offline
- Taint Analysis
- Symbolic Execution
- Fuzzing *(80%?)*
  - mutation, generation
  - blackbox, greybox, whitebox
  - smart, dumb

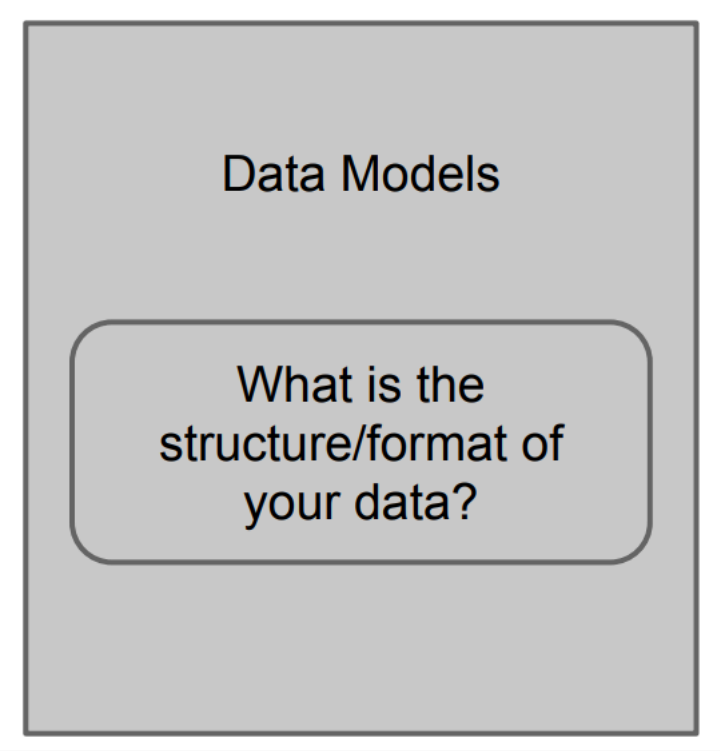
Basics of Fuzzing

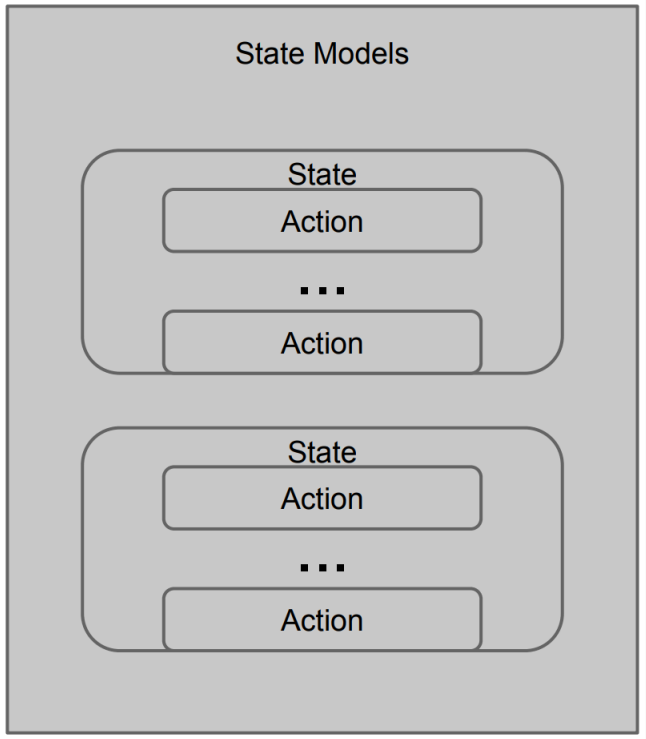


- generation-based
  - generate inputs from templates (e.g., grammar, specification)
- mutation-based
  - mutate inputs from seed inputs

Basics of Fuzzing: PEACH

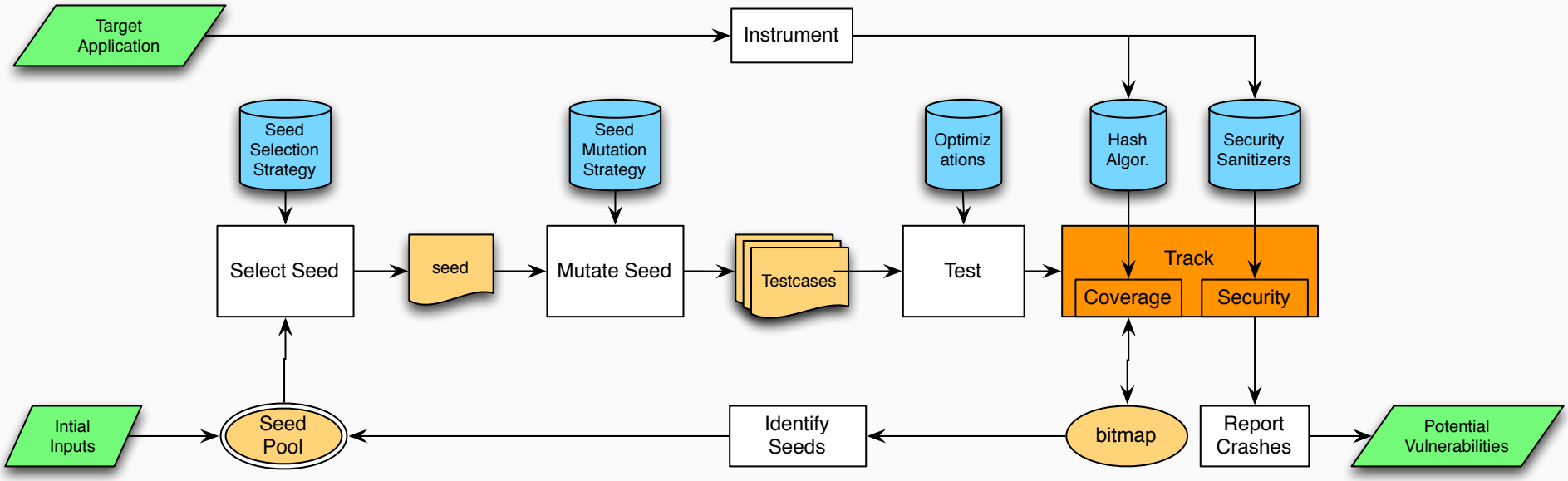
- generation-based and mutation-based fuzzing
- relies on user-supplied data and state models





Basics of Fuzzing: AFL

A very popular open source mutation-based code coverage guided fuzzer.

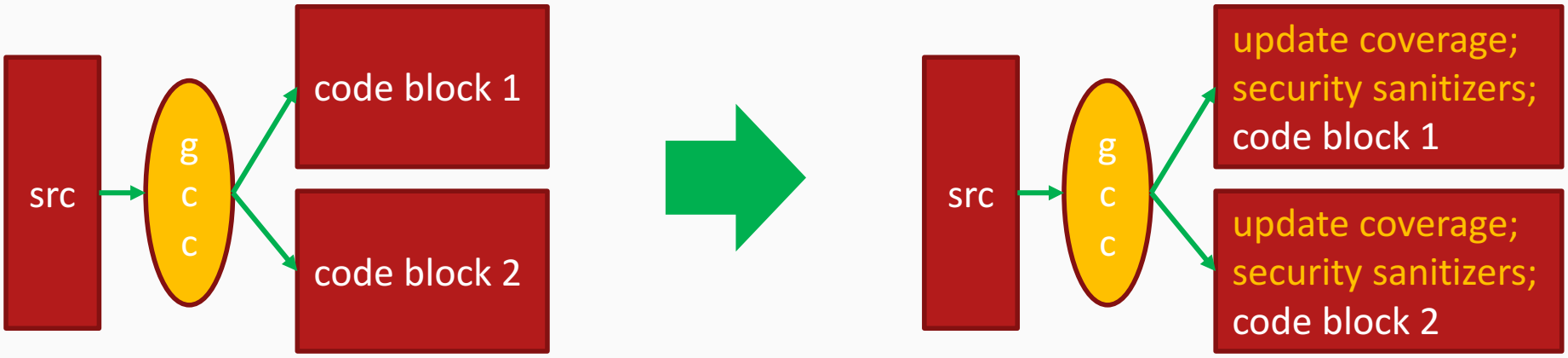


- scalable, few knowledge is required
- evolving, code coverage guided
- fast, throughput is high
  testing, tracking
- sensitive, catch security violations

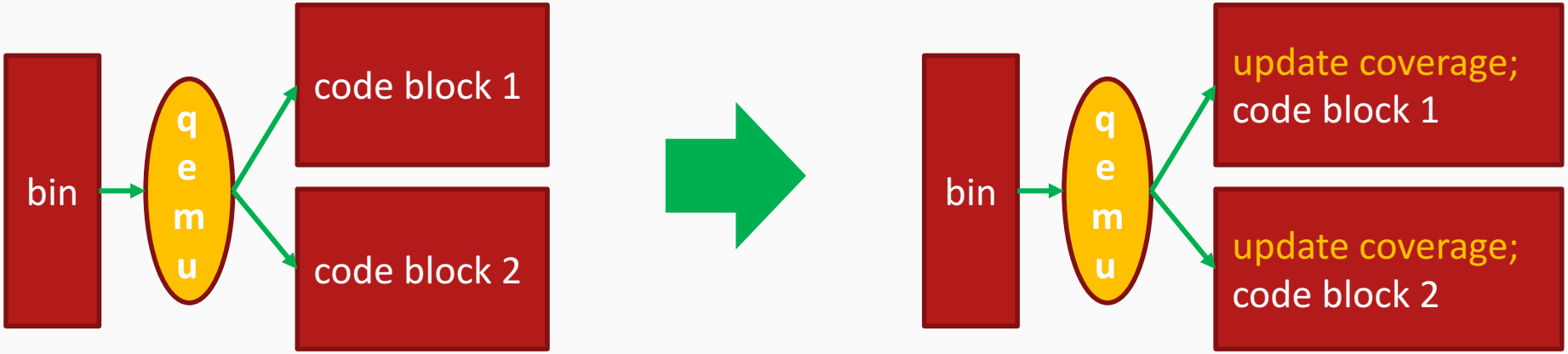
- mutation-based
- keep only GOOD seeds contributing to cov
- fork, forkserver, persistent, parallel
  - *gcc_mode, qemu_mode*
- AddressSanitizer, ThreadSanitizer...

AFL Instrumentation

- gcc_mode, llvm_mode (src required)



- qemu_mode (binary only)



Kernel Fuzzing

--- syscall fuzzing

Categories

- Knowledge based
- Coverage guided

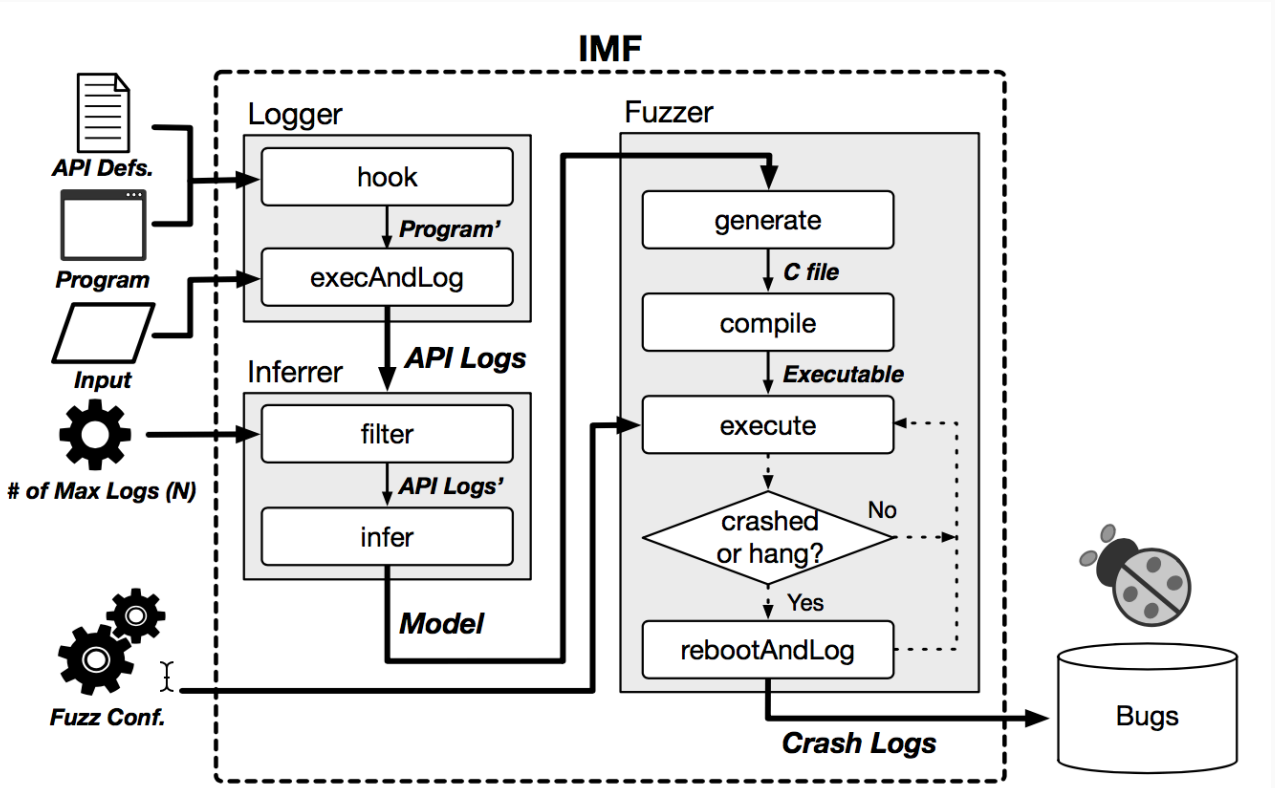
Trinity

- https://github.com/kernelslacker/trinity

- Idea: feed syscall with arguments of correct type
  - certain data type
  - certain enumeration values
  - certain range of values

IMF: Inferred Model-based Fuzzer (CCS'07)

- Learn from normal testing, to get templates
  - order dependency of syscalls
  - value dependency of syscalls
- Generate testcases based on templates

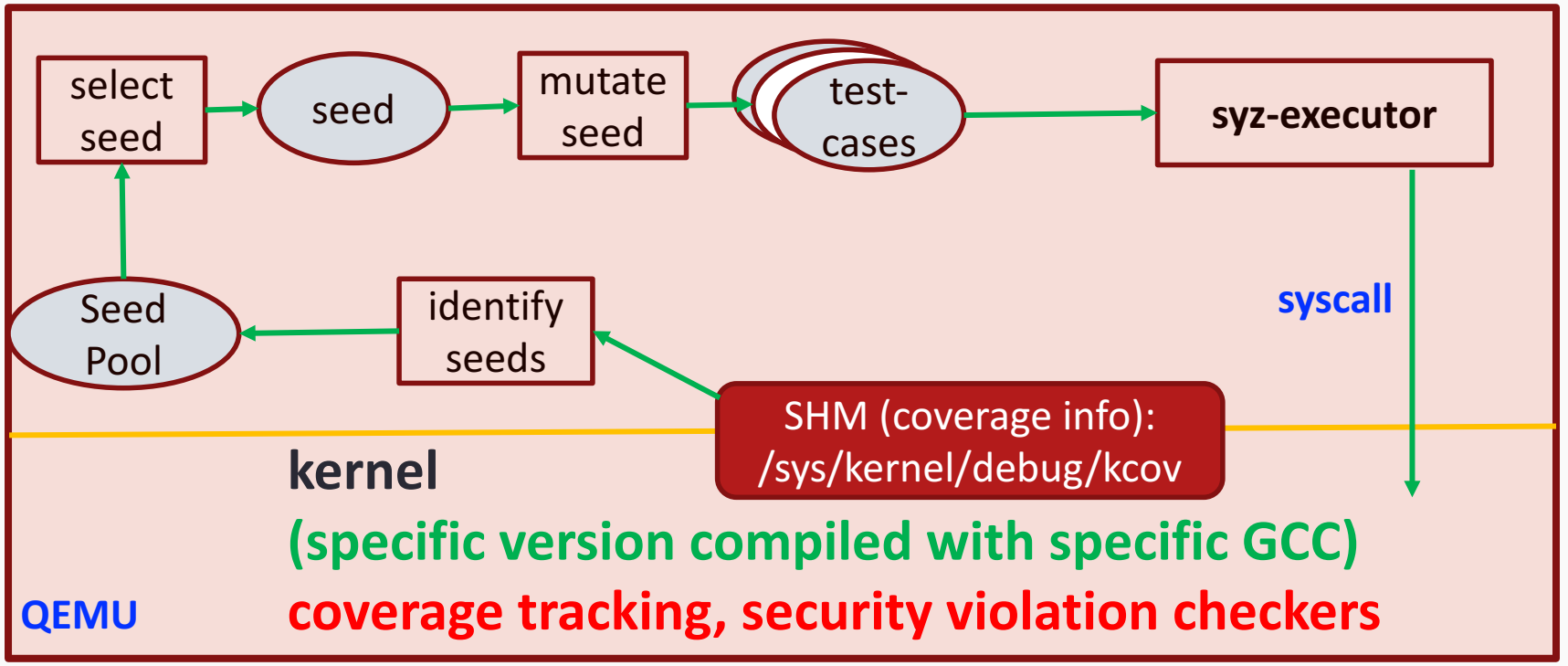


Categories

- Knowledge based
- Coverage guided

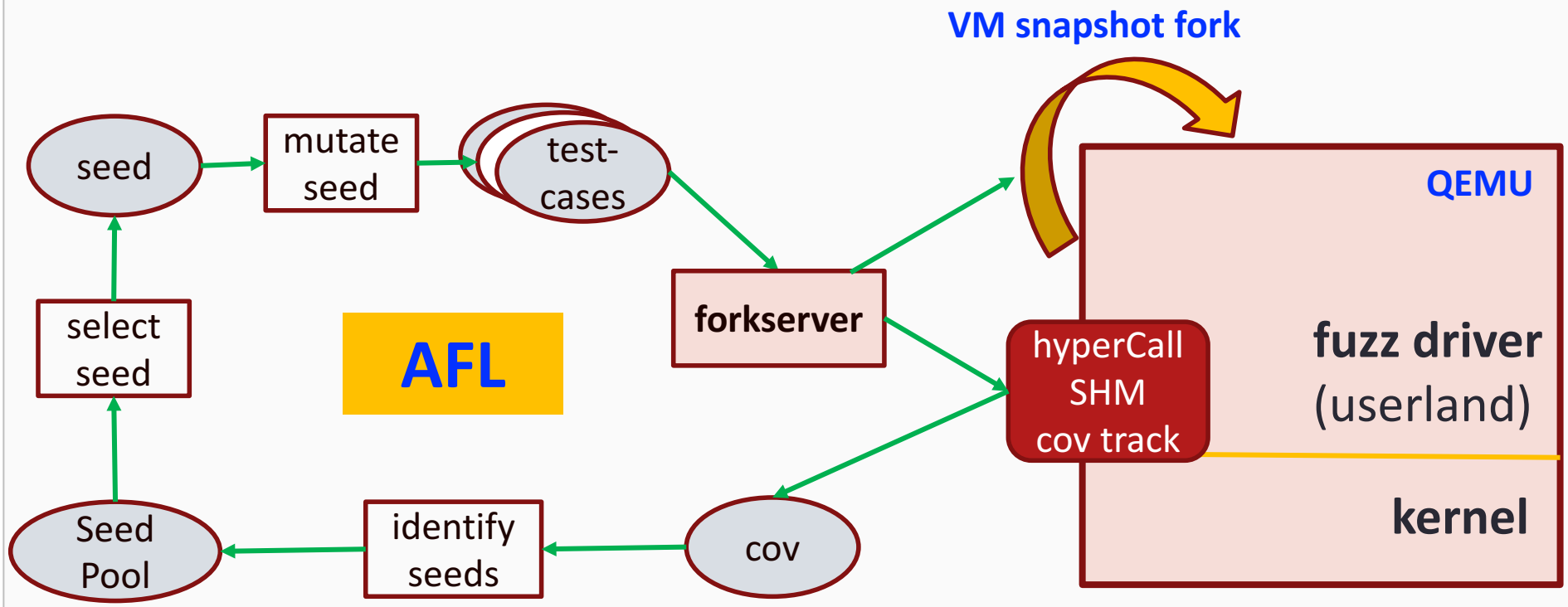
syzkaller (AFL gcc_mode)

- Instrument the kernel (via compilation) with
  - code coverage tracking
  - security violation checking
- Multiple VM could be parallelized



TriforceAFL (AFL qemu_mode)

- A modified version of AFL that supports kernel fuzzing with QEMU full-system emulation.



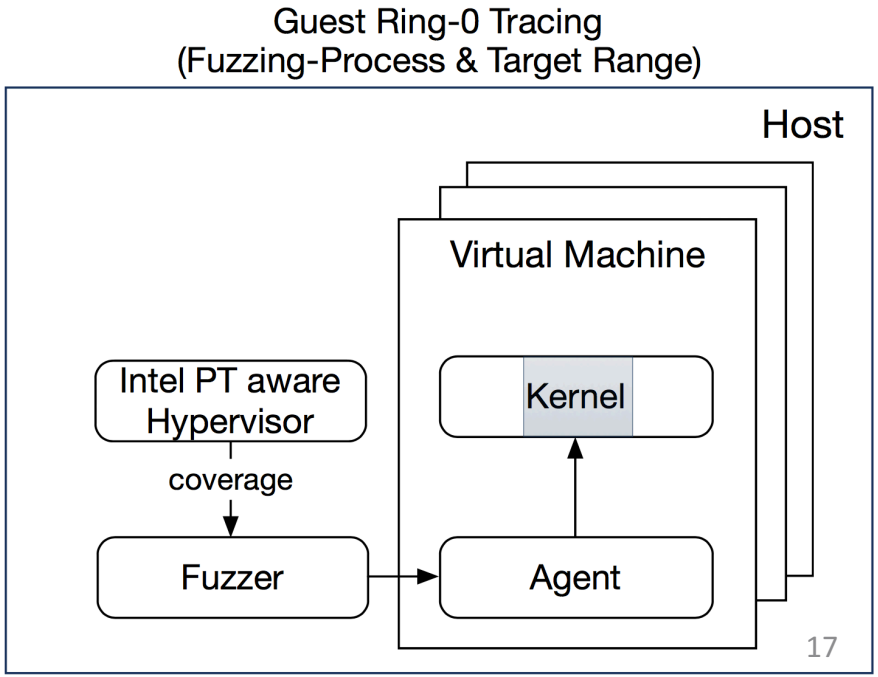
kAFL: Hardware-Assisted (USENIX Sec'07)

- Motivation

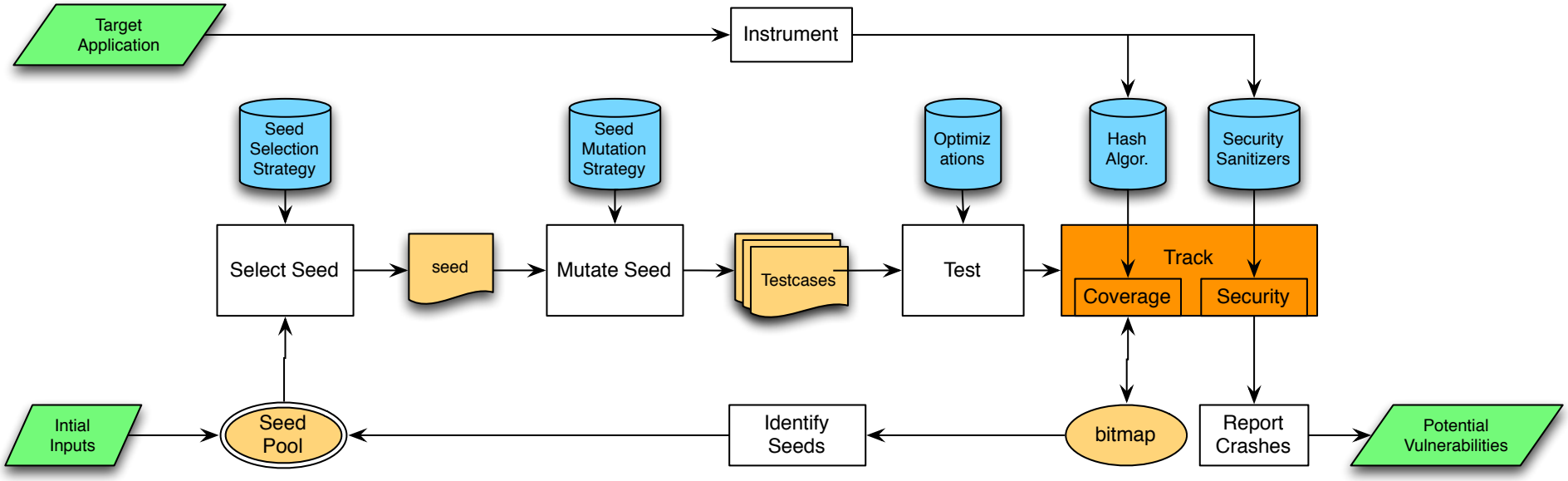
	Fast	Crash Tolerant	OS Independent	Binary Only
TriforceAFL (Jesse Hertz & Tim Newsham, NCC Group)	✗	✓	~	✓
Syzkaller (Dmitry Vyukov)	✓	✓	✗	✗
AFL Filesystem Fuzzer (Vegard Nossum & Quentin Casanovas, Oracle)	✓	~	✗	✗
PT Kernel Fuzzer (Richard Johnson, Talos)	✓	✗	✗	✓

- Solution
  - track coverage via Intel PT
  - track only kernel code
    - vCPU, supervisor
    - process, IP range
  - QEMU+KVM

40X faster than Triforce



Improve fuzzing


Target Application
Instrument
Seed Selection Strategy
Seed Mutation Strategy
Optimiz ations
Hash Algor.
Security Sanitizers
Select Seed
seed
Mutate Seed
Testcases
Test
Track
Coverage
Security
Intial Inputs
Seed Pool
Identify Seeds
bitmap
Report Crashes
Potential Vulnerabilities

Questions of Cov-guided Fuzzing

- How to get initial inputs?
- How to select seed from the pool?
- How to generate new testcases?
  - How to mutate seeds? Location and value.
- How to test target application?
  - efficiency, input sources, coverage, ...
- How to track the testing?
  - code coverage, security violation, ...?

How to get initial inputs?

- Why is it important?
  - cpu time
  - complex data structure
  - hard-to-reach code
  - reusable between fuzzings
- Solutions
  - standard benchmarks
  - crawling from the Internet
  - existing PoC samples
- Extra step
  - distill the corpus

How to select seed from the pool?

- Why is it important?
  - prioritize seeds which are more helpful,
    - e.g., cover more code, more likely to trigger vulnerabilities
  - save computing resources
  - faster to identify hidden vulnerabilities
- Solutions
  - AFLFast (CCS'16): seeds exercising less-frequent paths or picked fewer
  - Vuzzer (NDSS'17): seeds exercising deeper paths
  - QTEP (FSE'17): seeds covering more faulty code
  - AFLgo (CCS'17): seeds closer to target vulnerable paths
  - SlowFuzz (CCS'17): seeds consuming more resources

How to generate new testcases?

- Why is it important?
  - explore more code in a shorter time
  - target potential vulnerable locations
- Solutions
  - Vuzzer (NDSS’17):
    - where to mutate: bytes related to branches
    - what value to use: tokens used in the code.
  - Skyfire (Oakland’17):
    - learn Probabilistic Context-Sensitive Grammar from crawled inputs
  - Learn&Fuzz (Microsoft, 2017/2):
    - learn RNN from valid inputs
  - Neural Fuzzing (Microsoft, 2017/11)
    - predicate which bytes to mutate via LSTM
  - GAN Fuzzing (2017/11)

How to efficiently test application?

- Why is it important?
  - test more in a unit time
  - very important
- Efficiency:
  - fork + execve
  - forkserver
  - persistent mode
  - parallel mode
  - Intel PT
  - ...
- Input sources:
  - stdio
  - file
  - network
  - GUI
  - managed code
  - ...

How to track the testing?

- Why is it important?
  - Code coverage: leading to thorough program states exploring
  - Security violations: capturing bugs that have no explicit results

- Code coverage:
  - AFL bitmap
  - SanitizerCoverage
  - code instrumentation
    - source code
    - binary code, qemu_mode

- Security violations:
  - AddressSanitizer
  - UBSan
  - MemorySanitizer
  - ThreadSanitizer
  - DataFlowsanitizer
  - LeakSanitizer
  - ...

Fuzzing in real world

- Dumb enough, easy to use, but effective!
  - VERY popular in industry
- Key to find more vulnerabilities
  - domain knowledge
  - write your own mutation algorithm for your target application

纯干货：微软漏洞中国第一人黄正——如何用正确姿势挖掘浏览器漏洞（附完整 PPT） | 硬创公开课

Conclusions

- Fuzzing is the most popular vulnerability discovery solution.
- AFL is one of the most popular fuzzers, studied by academia and industry researchers.
  - scalable, fast, evolving, sensitive
- Kernel fuzzing attracts more and more attentions.
- We could improve fuzzers in many ways.

? & #