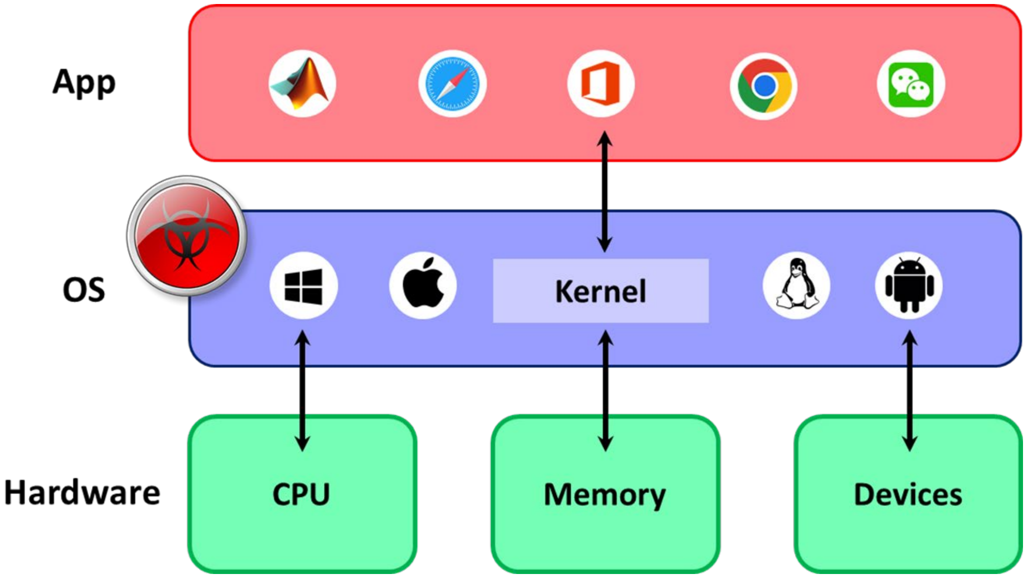
MOCK: Optimizing Kernel Fuzzing Mutation with Context-aware Dependency

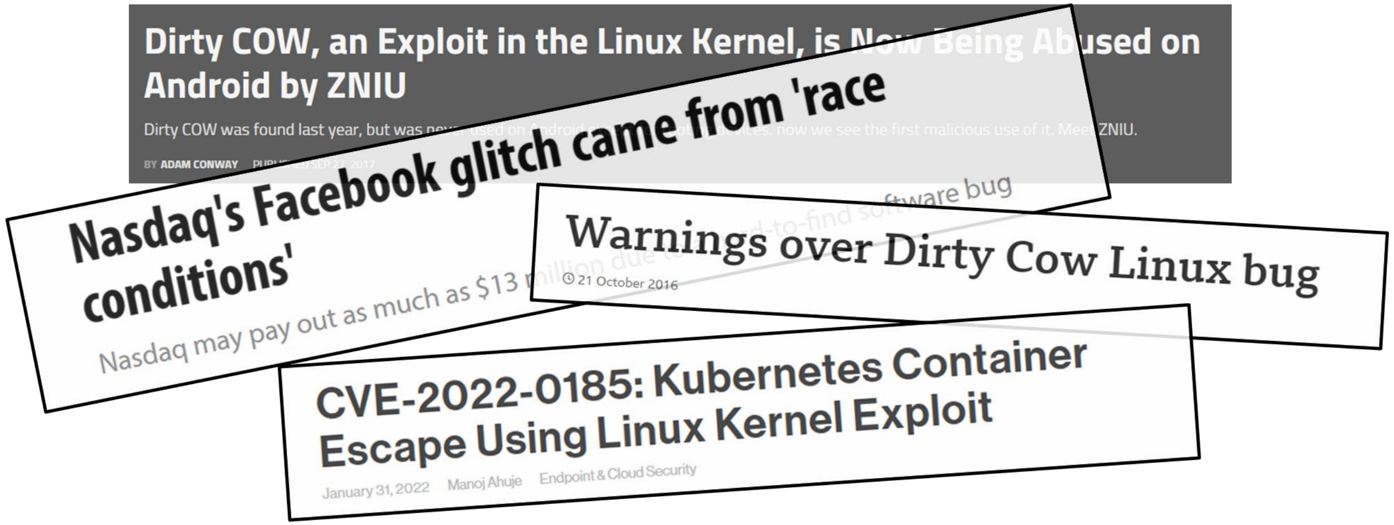
Jiacheng Xu[1], Xuhong Zhang[1], Shouling Ji[1], Yuan Tian[2]
BinBin Zhao[3], Qingying Wang[1], Peng Cheng[1], Jiming Chen[1]

[1]Zhejiang University [2]University of California, Los Angeles [3]Georgia Institute of Technology

OS Kernel

OS kernel is ubiquitous and crucial, but also vulnerable.





Dirty COW, an Exploit in the Linux Kernel, is Now Being Abused on Android by ZNIU

Dirty COW was found last year, but was never used on Android ... now we see the first malicious use of it. Meet ZNIU.

BY ADAM CONWAY PUBLISHED SEP 27 2017

Nasdaq's Facebook glitch came from 'race conditions'

Nasdaq may pay out as much as $13 million due to hard-to-find software bug

Warnings over Dirty Cow Linux bug

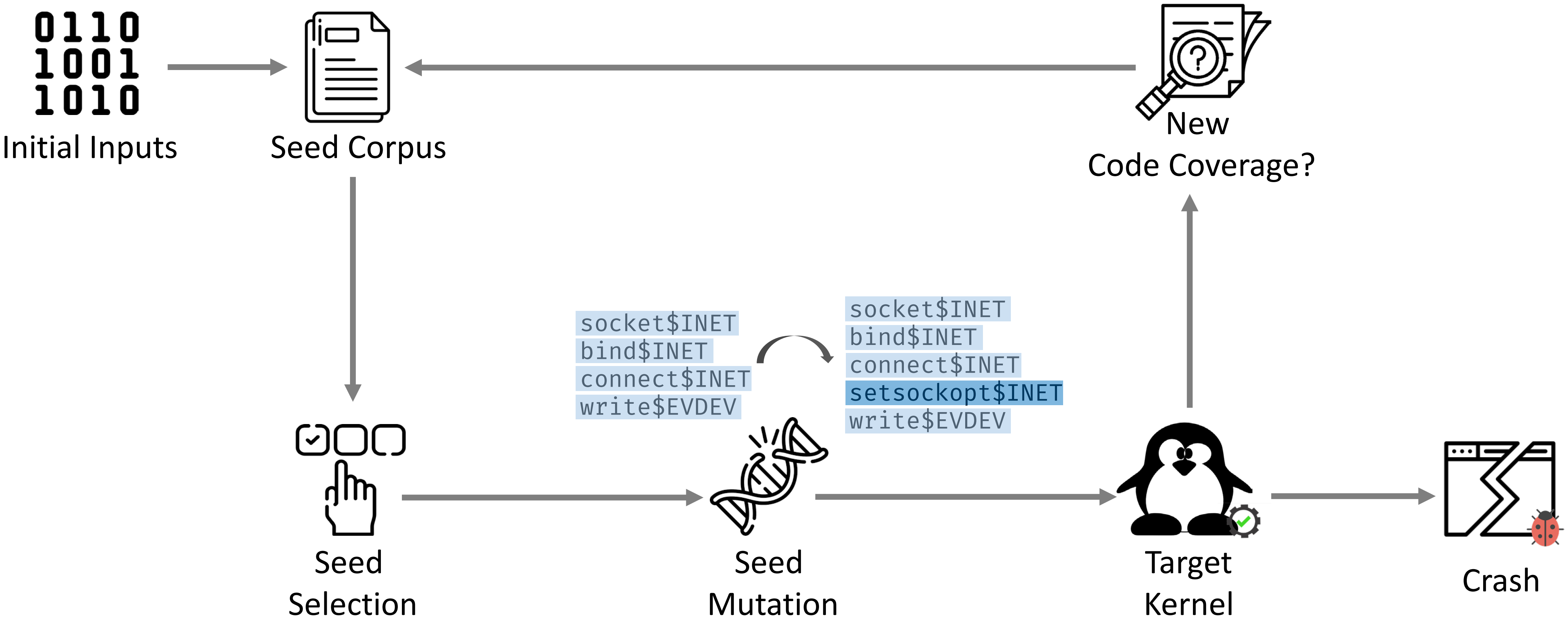
21 October 2016

CVE-2022-0185: Kubernetes Container Escape Using Linux Kernel Exploit

January 31, 2022 Manoj Ahuje Endpoint & Cloud Security

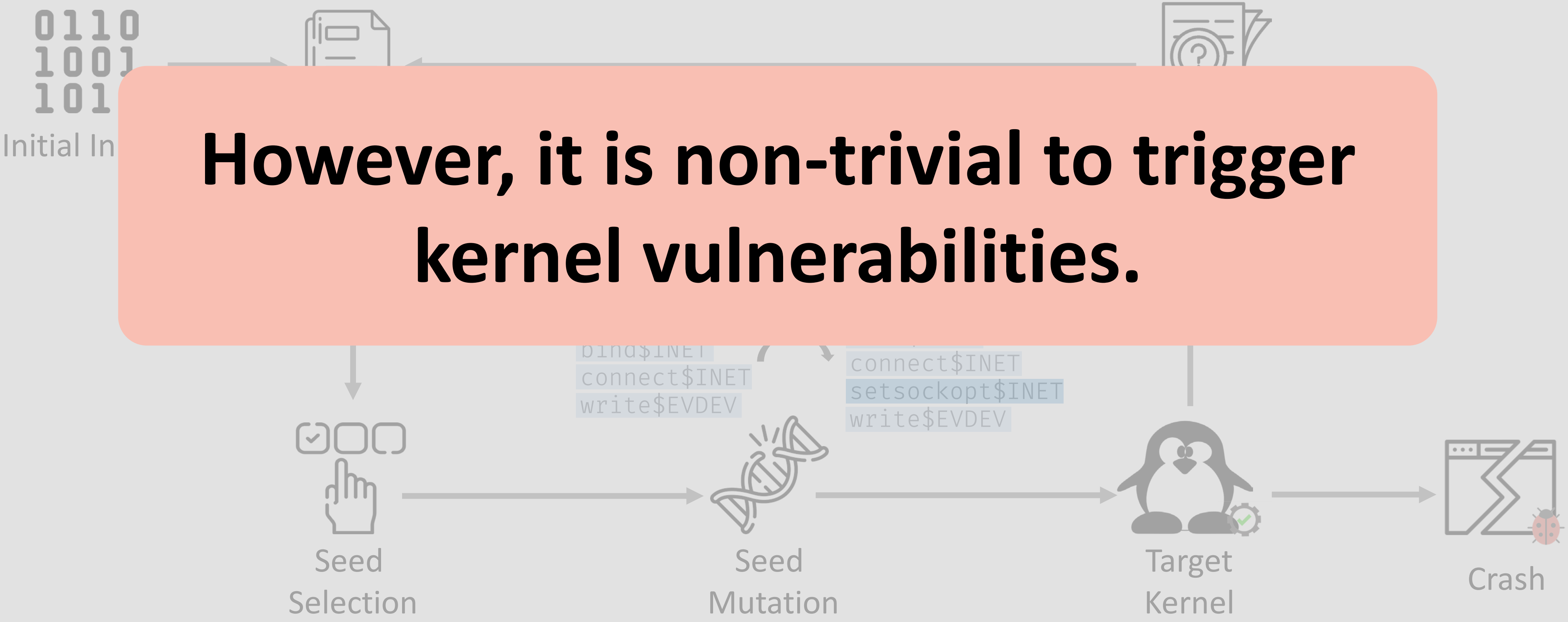
Kernel Fuzzing

Syscall-based greybox fuzzing is a popular technique for finding vulnerabilities.



Kernel Fuzzing

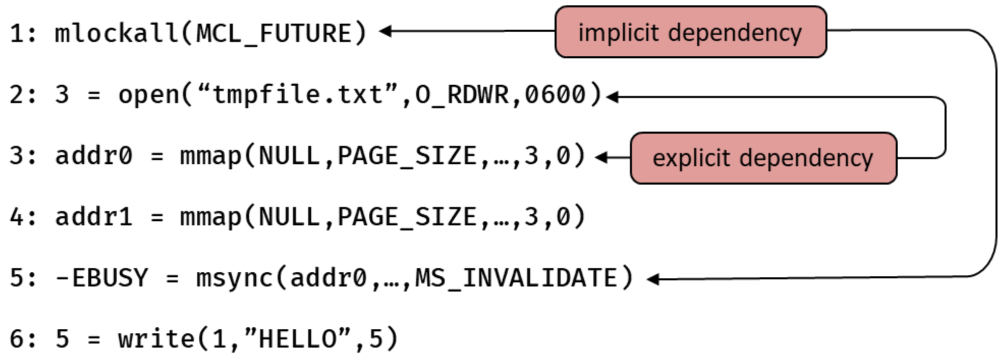
Syscall-based greybox fuzzing is a popular technique for finding vulnerabilities.



However, it is non-trivial to trigger kernel vulnerabilities.

Motivation

Input synthesis is one of bottlenecks.



```
mlockall(MCL_FUTURE)
3 = open("tmpfile.txt",O_RDWR,0600)
addr0 = mmap(NULL,PAGE_SIZE,…,3,0)
addr1 = mmap(NULL,PAGE_SIZE,…,3,0)
-EBUSY = msync(addr0,…,MS_INVALIDATE)
5 = write(1,"HELLO",5)
```

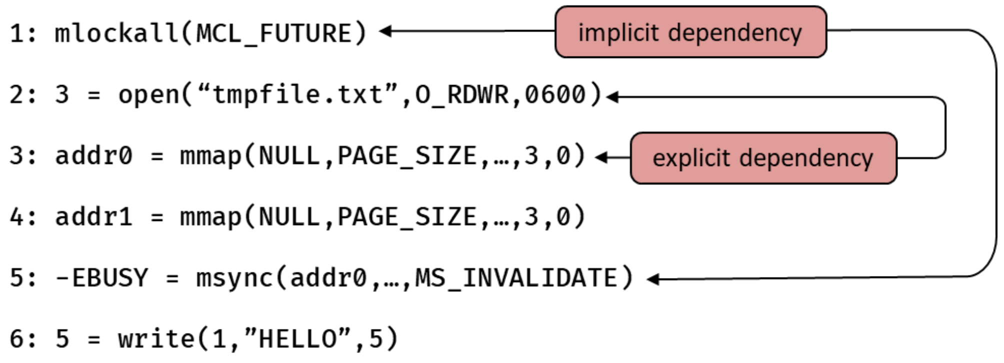
implicit dependency

explicit dependency

- **Explicit/implicit dependency.**
- **Complex bug condition.**
- **Huge search space $\sum_{k=8}^{32}\binom{4000}{k}$.**

Motivation

Input synthesis is one of bottlenecks.



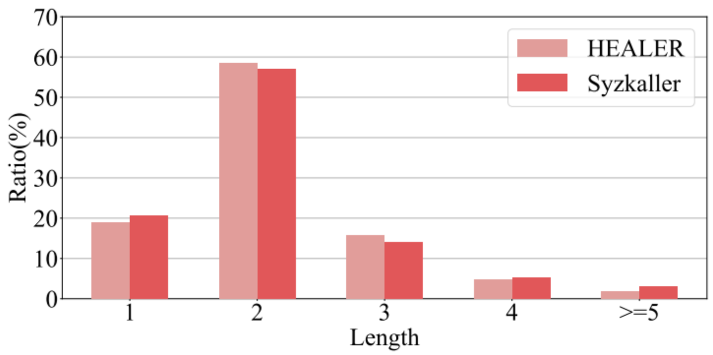
```
mlockall(MCL_FUTURE)
3 = open("tmpfile.txt",O_RDWR,0600)
addr0 = mmap(NULL,PAGE_SIZE,…,3,0)
addr1 = mmap(NULL,PAGE_SIZE,…,3,0)
-EBUSY = msync(addr0,…,MS_INVALIDATE)
5 = write(1,"HELLO",5)
```

implicit dependency

explicit dependency

- **Explicit/implicit dependency.**
- **Complex bug condition.**
- **Huge search space $\sum_{k=8}^{32}\binom{4000}{k}$.**

Existing works have limitations in modeling dependency.

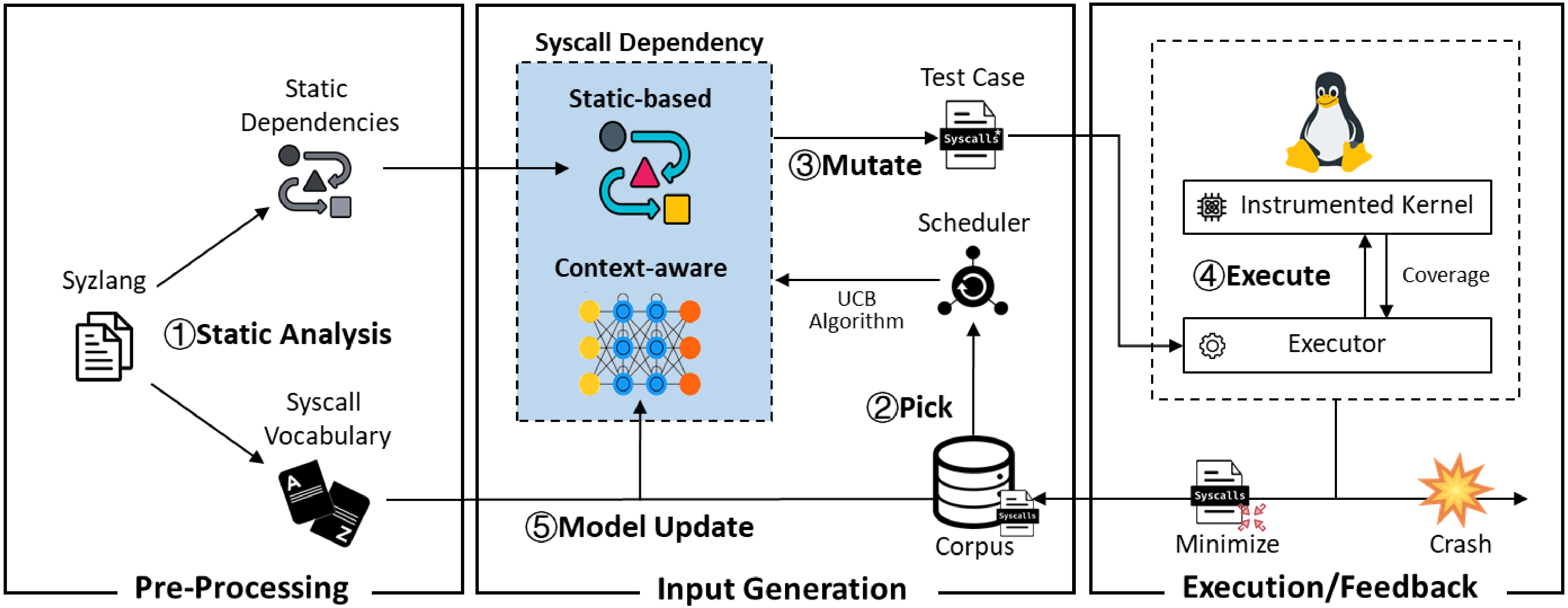


- **24-hour run: 8% of input has 3+ syscalls.**
- **Context-free dependency/mutation.**
- **Underrate the seed corpus.**

Context-aware dependency is desired. But, how to automatically model and utilize context-aware dependency for better fuzzing?

Our Approach

MOCK: a prototype for context-aware kernel fuzzing.



Context-aware Dependency Modeling

¤ **Infer context-aware dependency**

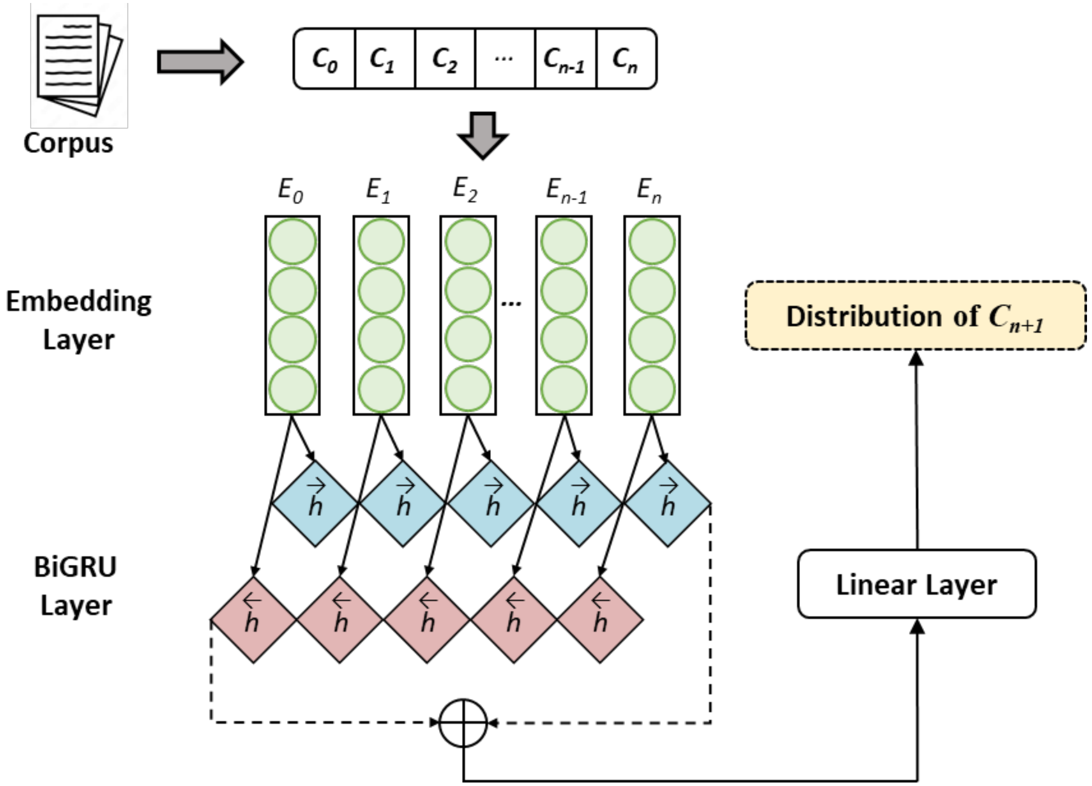
- **a conditional probability under various contexts.**
- **regarded as a NLP problem.**

¤ **Prepare trainingset.**

- **syscall sequences that achieve new code coverage.**
- **sequence minimization[1-2].**

¤ **Employ language model**

- **Bi-LSTM model that detects both front and rear contexts.**



[1] Syzkaller, https://github.com/google/syzkaller
[2] Sun, Hao, et al. "Healer: Relation learning guided kernel fuzzing." SOSP’21.

Context-aware Mutation

¤ Extract context

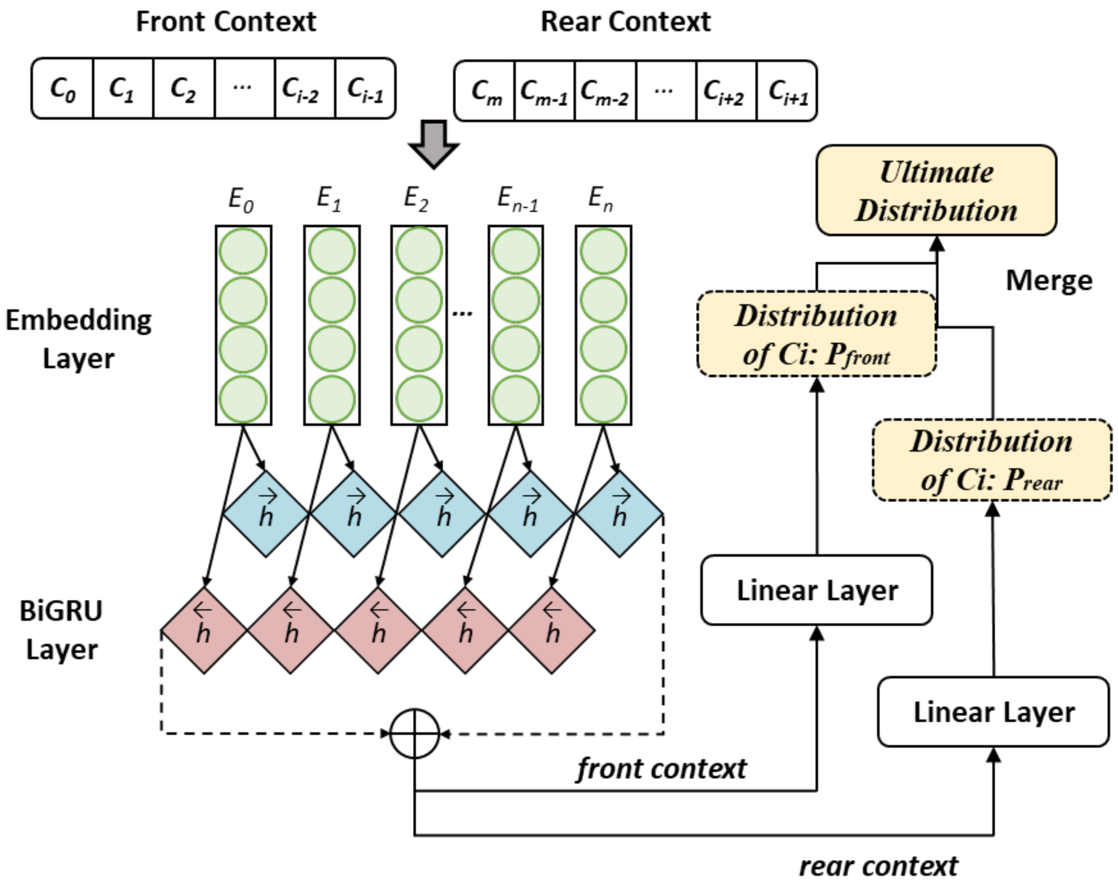
- decide a mutation position.
- extract front and rear contexts.

¤ Candidate suggestion

- feed front and rear context into the model, respectively.
- predict candidate syscalls with two probabilities P_{front} and P_{rear}.

¤ Syscall selection

- merge P_{front} and P_{rear} as the ultimate distribution P_i.
- random choose a candidate by weight P_i.


Front Context
Rear Context
C_0 C_1 C_2 … C_{i-2} C_{i-1}
C_m C_{m-1} C_{m-2} … C_{i+2} C_{i+1}
E_0 E_1 E_2 E_{n-1} E_n
Embedding Layer
BiGRU Layer
$\vec{h}$
$\overleftarrow{h}$
Ultimate Distribution
Merge
Distribution of Ci: P_{front}
Distribution of Ci: P_{rear}
Linear Layer
Linear Layer
front context
rear context

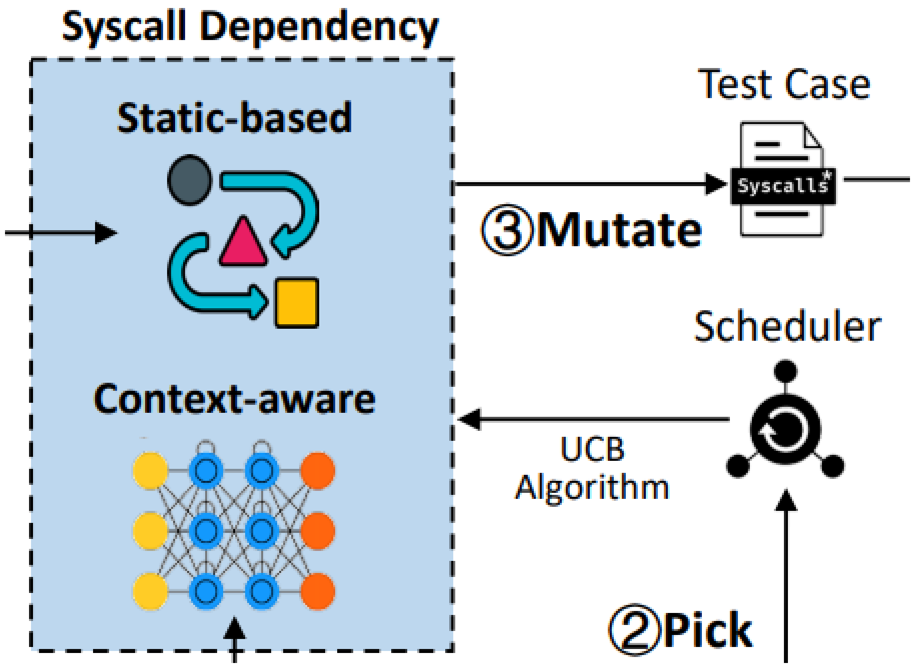
Task Scheduling

¤ Why scheduling

- over-reliance is harmful.
- lack of diversity.
- limited mutation candidates.

¤ Multi-armed bandit

- static dependency & context-aware dependency.
- UCB-1 algorithm.
- coverage-oriented rewards.
- the task that discovers more new code coverage is preferred.



Evaluation

- **RQ1: How does MOCK perform in code coverage?**
- **RQ2: How effective is context-aware dependency compared to context-free dependency?**
- **RQ3: Do various setups (e.g. initial seeds, pre-trained models) reduce warmup time and boost fuzzing performance?**
- **RQ4: How does MOCK perform in vulnerability detection?**
- **RQ5: Can MOCK discover new vulnerabilities in real-world kernels?**
- **RQ6: How is the significance and overhead of key components in MOCK?**

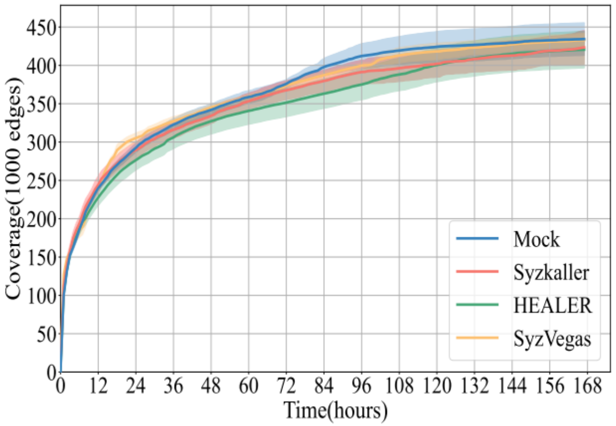
Experiment Setup

- Target kernel: Linux-5.4, 5.10, 5.15.
- The same configuration of kernels and resources.
- Baseline: Syzkaller, HEALER, SyzVegas.
- Fuzzing time budget: 144 hours.
- No initial seeds.

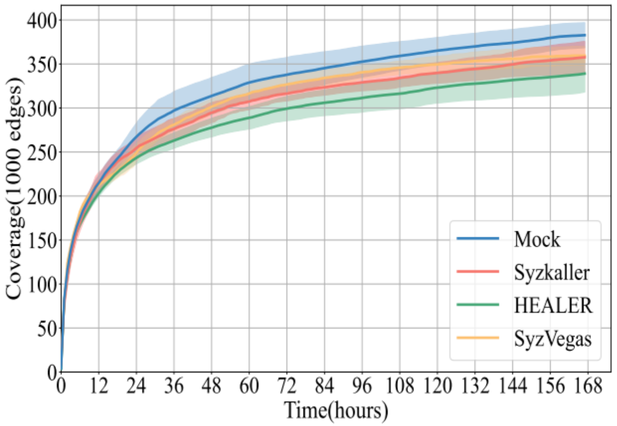
Evaluation (1/6)

¤ Coverage Performance

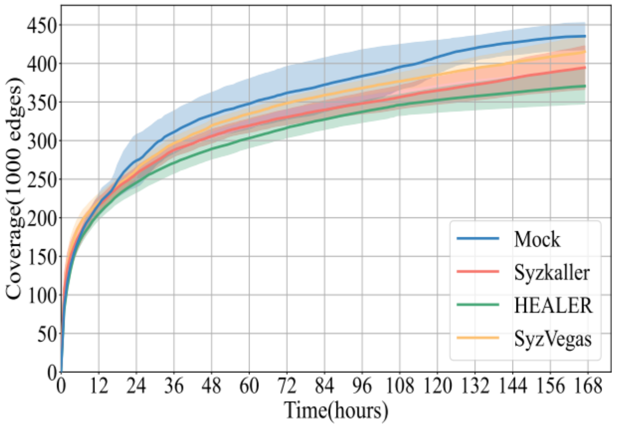
- **code coverage: MOCK achieves a 7% increase compared to SOTAs on average.**
- **speed-up: MOCK achieves a 1.71x acceleration compared to SOTAs on average.**



(a) Linux-5.4



(b) Linux-5.10

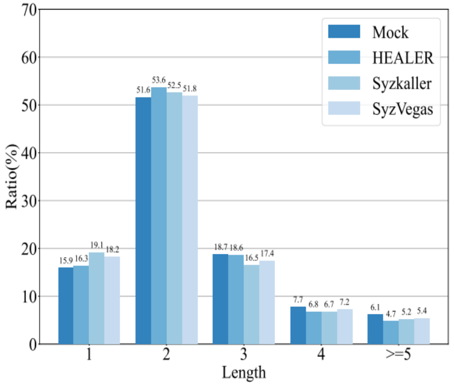


(c) Linux-5.15

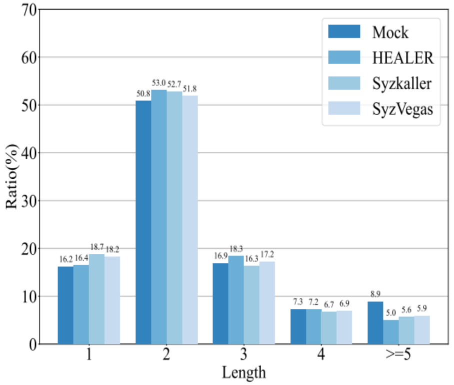
Evaluation (2/6)

¤ Effectiveness of Context-aware Dependency

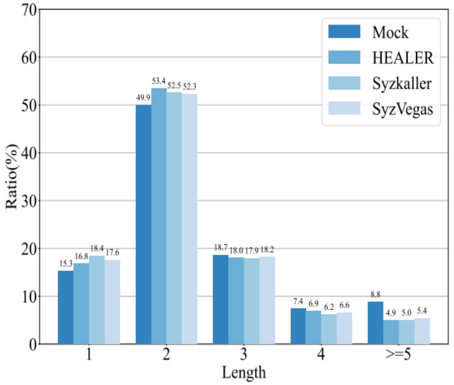
- testcase analysis: MOCK can produce 50% more interrelated syscall sequences.
- contextual mutation analysis: context-aware dependency facilitates more interrelated input synthesis while marginally deficient in a simple context.



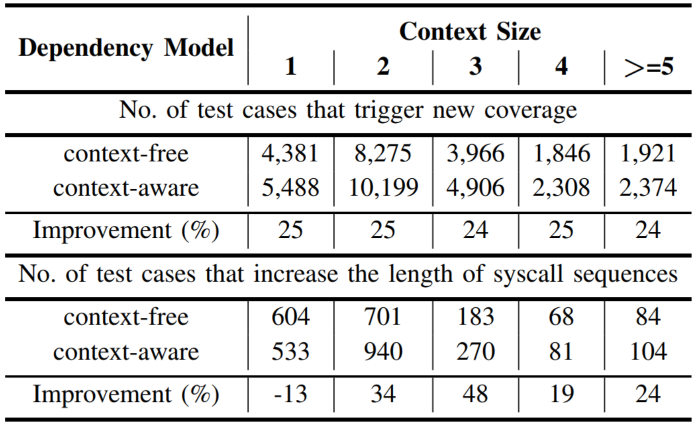
(a) Linux-5.4



(b) Linux-5.10



(c) Linux-5.15

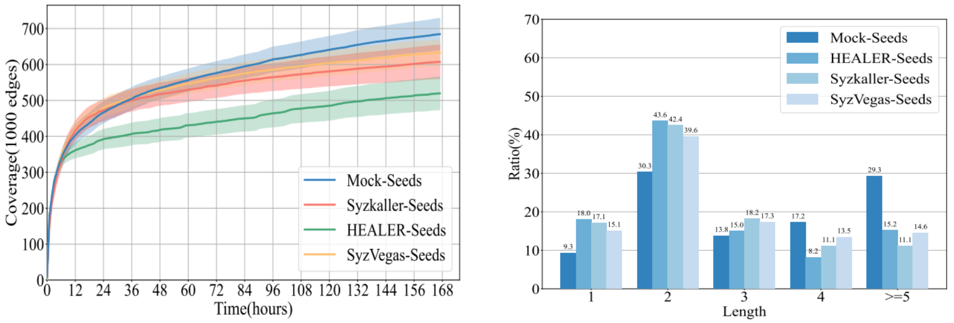


Dependency Model	**Context Size**				
	1	**2**	**3**	**4**	**>=5**
No. of test cases that trigger new coverage					
context-free	4,381	8,275	3,966	1,846	1,921
context-aware	5,488	10,199	4,906	2,308	2,374
Improvement (%)	25	25	24	25	24
No. of test cases that increase the length of syscall sequences					
context-free	604	701	183	68	84
context-aware	533	940	270	81	104
Improvement (%)	-13	34	48	19	24

Evaluation (3/6)

¤ **Various setups**

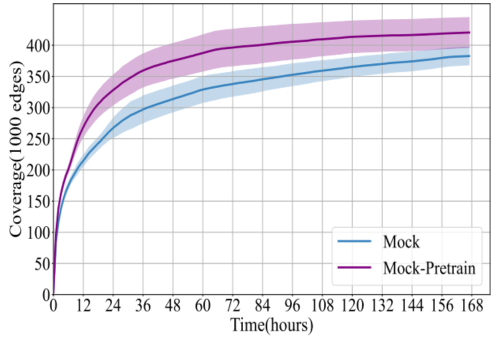
- **fuzzing with initial seeds: more coverage growth (21%), higher speed-up (2.58x) and more interrelated sequences.**



(a) Branch coverage

(b) Length distribution of corpus

- **Pre-trained model**
  - **corpus source: syzbot, previous runs.**
  - **MOCK-Pretrain earns advantages soon after startup.**

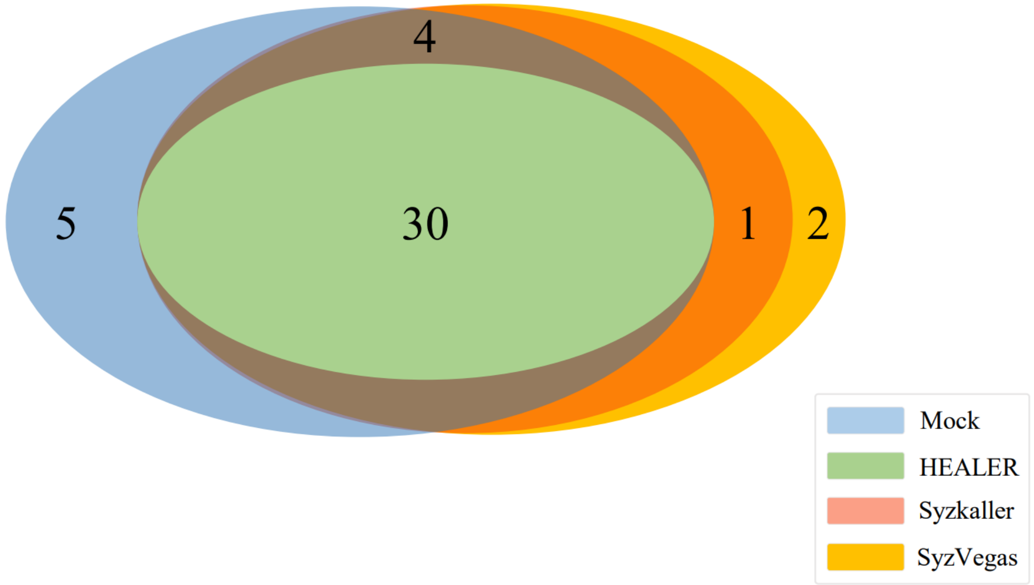


- **both setups reduce the warmup time and boost the fuzzing performance.**

Evaluation (4/6)

¤ Vulnerability Detection Ability

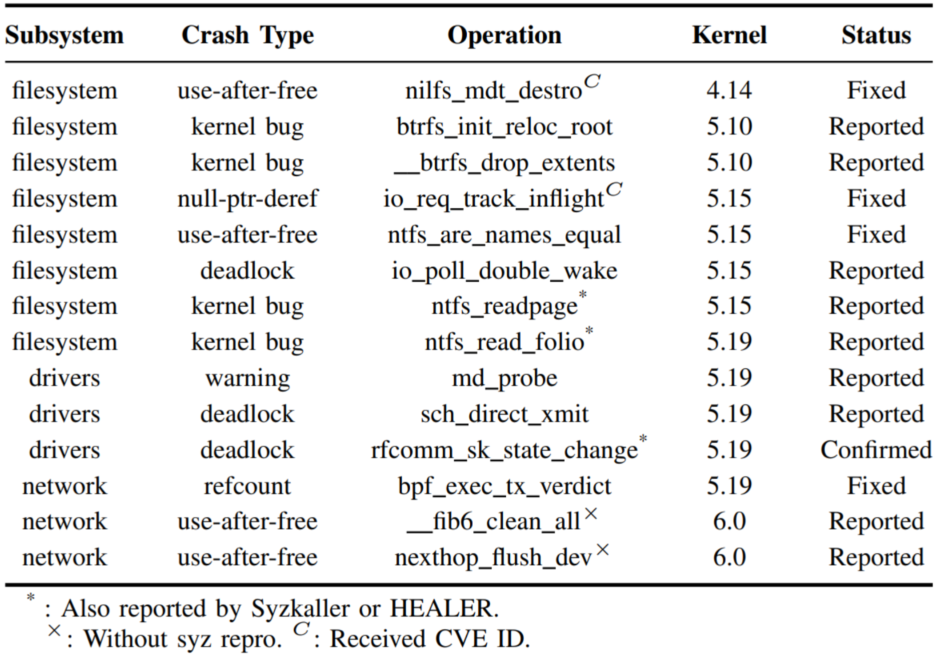
- **MOCK finds 15% more vulnerabilities than SOTAs.**
- **MOCK outperforms in finding vulnerabilities whose triggering requires more interrelated syscall sequences.**



Evaluation (5-6/6)

¤ Real-World Vulnerabilities Discovery

- MOCK found 15 unique vulnerabilities, of which four are confirmed and four are fixed.
- we also received two CVEs.

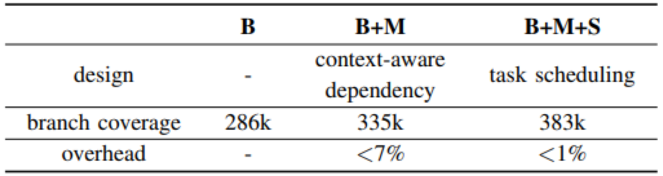


Subsystem	Crash Type	Operation	Kernel	Status
filesystem	use-after-free	nilfs_mdt_destro^{C}	4.14	Fixed
filesystem	kernel bug	btrfs_init_reloc_root	5.10	Reported
filesystem	kernel bug	__btrfs_drop_extents	5.10	Reported
filesystem	null-ptr-deref	io_req_track_inflight^{C}	5.15	Fixed
filesystem	use-after-free	ntfs_are_names_equal	5.15	Fixed
filesystem	deadlock	io_poll_double_wake	5.15	Reported
filesystem	kernel bug	ntfs_readpage*	5.15	Reported
filesystem	kernel bug	ntfs_read_folio*	5.19	Reported
drivers	warning	md_probe	5.19	Reported
drivers	deadlock	sch_direct_xmit	5.19	Reported
drivers	deadlock	rfcomm_sk_state_change*	5.19	Confirmed
network	refcount	bpf_exec_tx_verdict	5.19	Fixed
network	use-after-free	__fib6_clean_all$^{\times}$	6.0	Reported
network	use-after-free	nexthop_flush_dev$^{\times}$	6.0	Reported

*: Also reported by Syzkaller or HEALER.
$^{\times}$: Without syz repro. ^{C}: Received CVE ID.

¤ Further analysis

- every component in MOCK has a crucial role to play.
- our designs introduce negligible overhead.



	B	B+M	B+M+S
design	-	context-aware dependency	task scheduling
branch coverage	286k	335k	383k
overhead	-	<7%	<1%

Future Work

- Incorporate various model structure and extra features (e.g. parameter types, direction) to augment dependency model.
- Extend dependency inference to syscalls in a concurrency space.

Conclusion

- **A new fuzzing solution MOCK to enhance input synthesis.**
- **MOCK infers dependency using data-driven approaches and conducts context-aware mutation with the dependency.**
- **Comprehensive evaluation shows MOCK outperforms the SOTA fuzzers in fuzzing Linux kernels.**

Q&A

Contact: stitch@zju.edu.cn