StateFuzz: System Call-Based State-Aware Linux Driver Fuzzing

Bodong Zhao[1], Zheming Li[1], Shisong Qin[1], Zheyu Ma[1], Ming Yuan[1], Wenyu Zhu[2], Zhihong Tian[3], Chao Zhang[1,4]

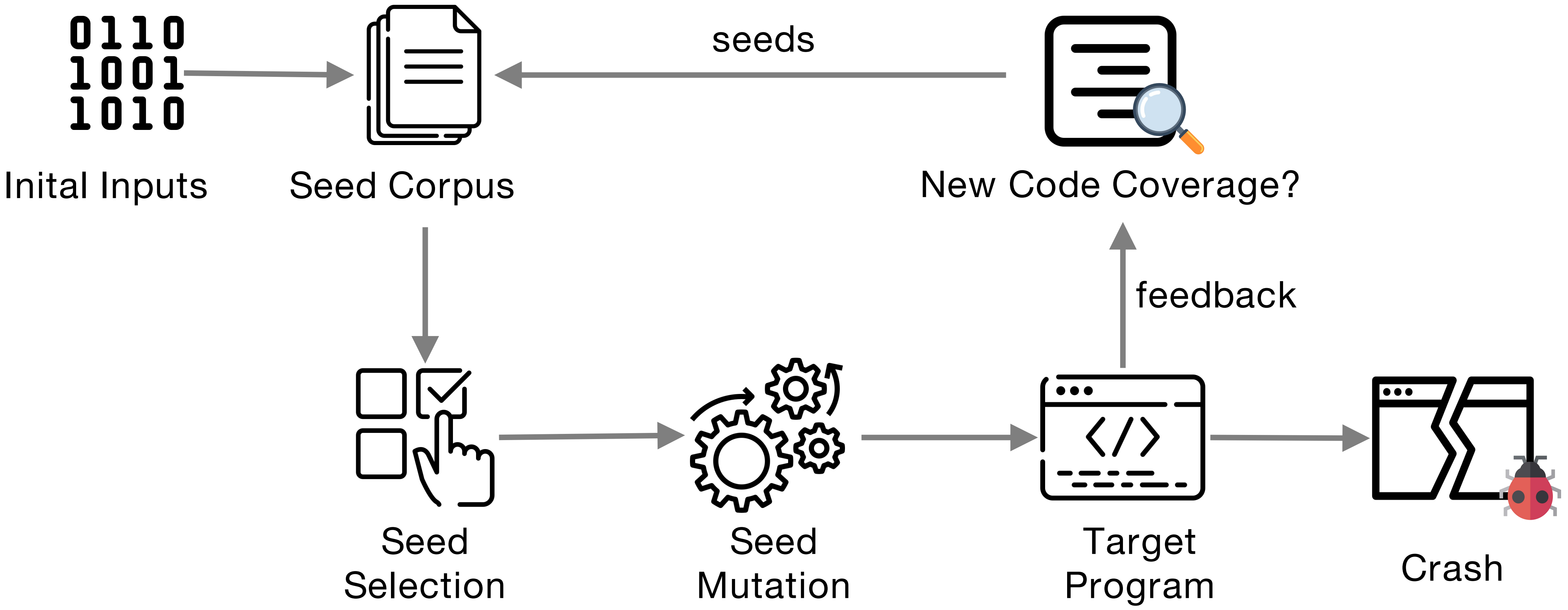
[1]Institute for Network Science and Cyberspace / BNRist, Tsinghua University

[2]Department of Electronic Engineering, Tsinghua University

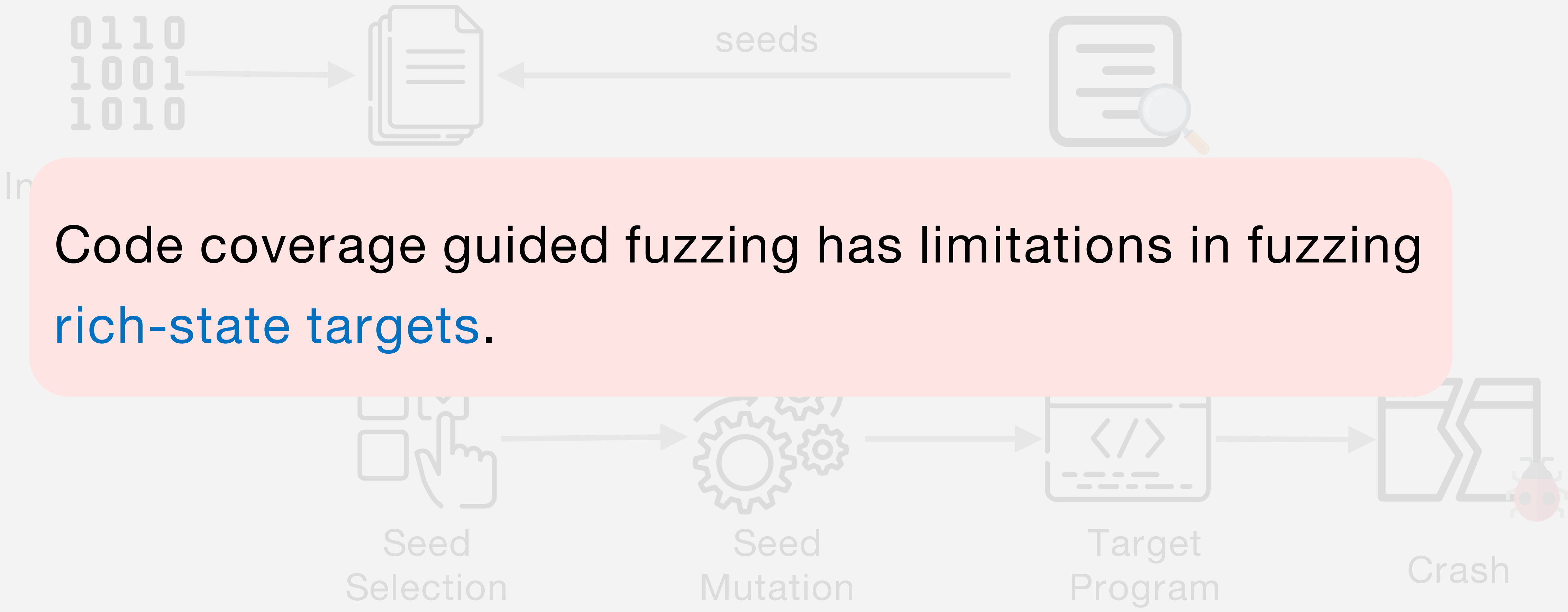
[3]Guangzhou University

[4]Zhongguancun Lab

Code Coverage Guided Fuzzing



Code Coverage Guided Fuzzing



Code coverage guided fuzzing has limitations in fuzzing **rich-state targets**.

Motivation Example

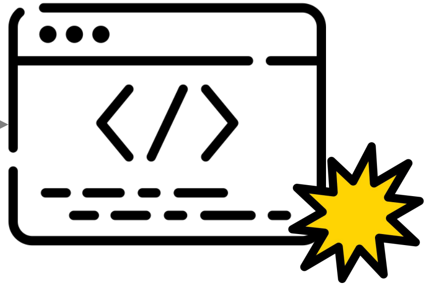
```c
int state_A = 0, state_B = 0;
int buf[BUF_SIZE];

void set_A(int value) {
    state_A = value;
}

void set_B(int value) {
    if (value<=BUF_SIZE && value>=0)
        state_B = value;
}

void vul(int value) {
    if (my_state_A == 0xff) {
        /* OOB bug here */
        buf[my_state_B] = value;
    }
}

void action(char op, int value) {
    switch (op) {
        case 'A': set_A(value);
        case 'B': set_B(value);
        case 'V': vul(value);
    }
}
```

Original State:
state_A = 0;
state_B = 0;

New State:
state_A = 0xff;
state_B = BUF_SIZE;

```
action('A', 0xff)
action('B', BUF_SIZE)
action('V', 0)
```

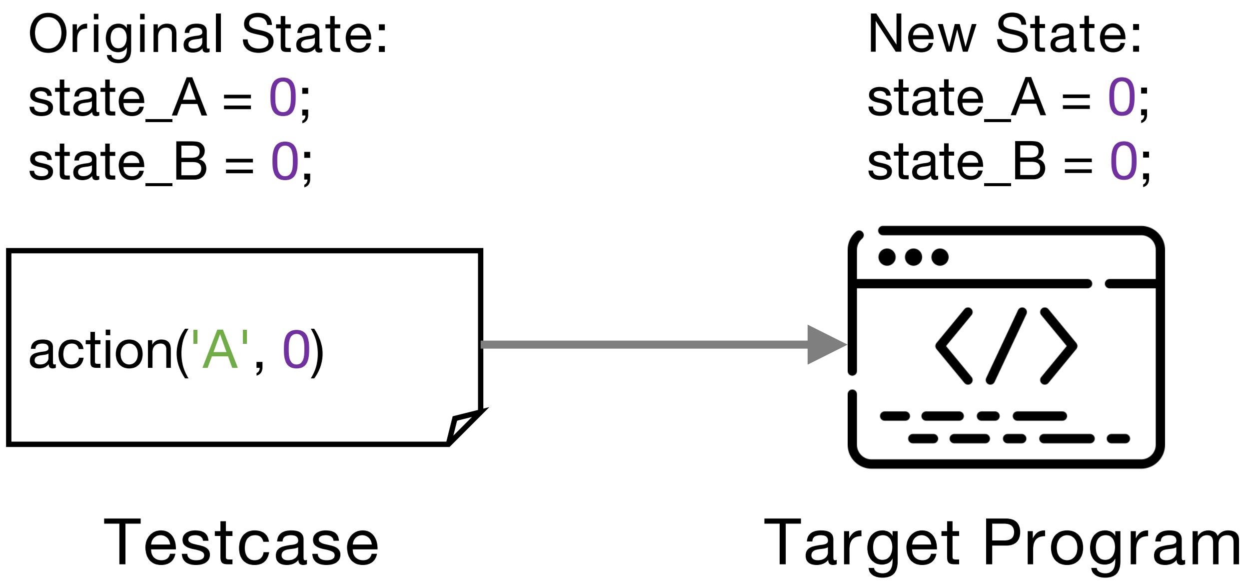
Testcase



Target Program

Motivation Example

```c
int state_A = 0, state_B = 0;
int buf[BUF_SIZE];

void set_A(int value) {
    state_A = value;
}

void set_B(int value) {
    if (value<=BUF_SIZE && value>=0)
        state_B = value;
}

void vul(int value) {
    if (my_state_A == 0xff) {
        /* OOB bug here */
        buf[my_state_B] = value;
    }
}

void action(char op, int value) {
    switch (op) {
        case 'A': set_A(value);
        case 'B': set_B(value);
        case 'V': vul(value);
    }
}
```

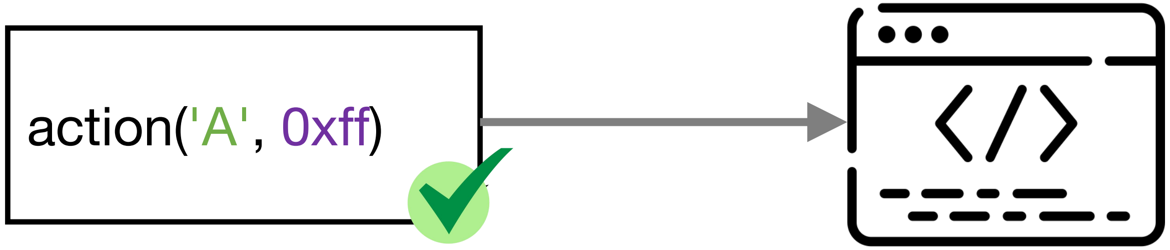


Motivation Example

```
int state_A = 0, state_B = 0;
int buf[BUF_SIZE];

void set_A(int value) {
    state_A = value;
}

void set_B(int value) {
    if (value<=BUF_SIZE && value>=0)
        state_B = value;
}

void vul(int value) {
    if (my_state_A == 0xff) {
        /* OOB bug here */
        buf[my_state_B] = value;
    }
}

void action(char op, int value) {
    switch (op) {
        case 'A': set_A(value);
        case 'B': set_B(value);
        case 'V': vul(value);
    }
}
```

Original State:
state_A = 0;
state_B = 0;

New State:
state_A = 0xff;
state_B = 0;



Testcase

Target Program

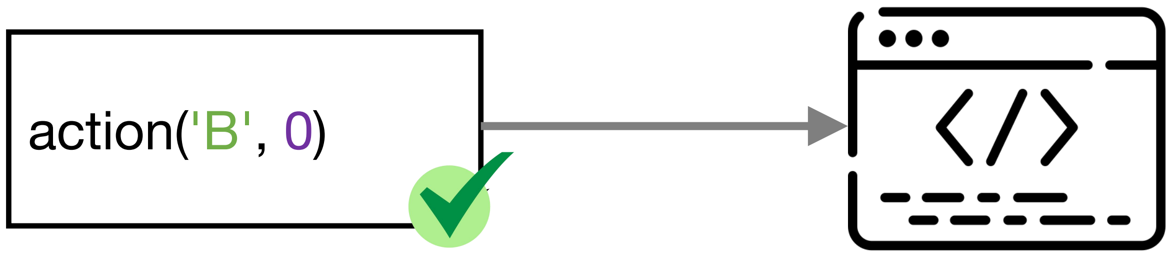
Hit new code, save this testcase.

Motivation Example

```
int state_A = 0, state_B = 0;
int buf[BUF_SIZE];

void set_A(int value) {
    state_A = value;
}

void set_B(int value) {
    if (value<=BUF_SIZE && value>=0)
        state_B = value;
}

void vul(int value) {
    if (my_state_A == 0xff) {
        /* OOB bug here */
        buf[my_state_B] = value;
    }
}

void action(char op, int value) {
    switch (op) {
        case 'A': set_A(value);
        case 'B': set_B(value);
        case 'V': vul(value);
    }
}
```

Original State:
state_A = 0xff;
state_B = 0;

New State:
state_A = 0xff;
state_B = 0;



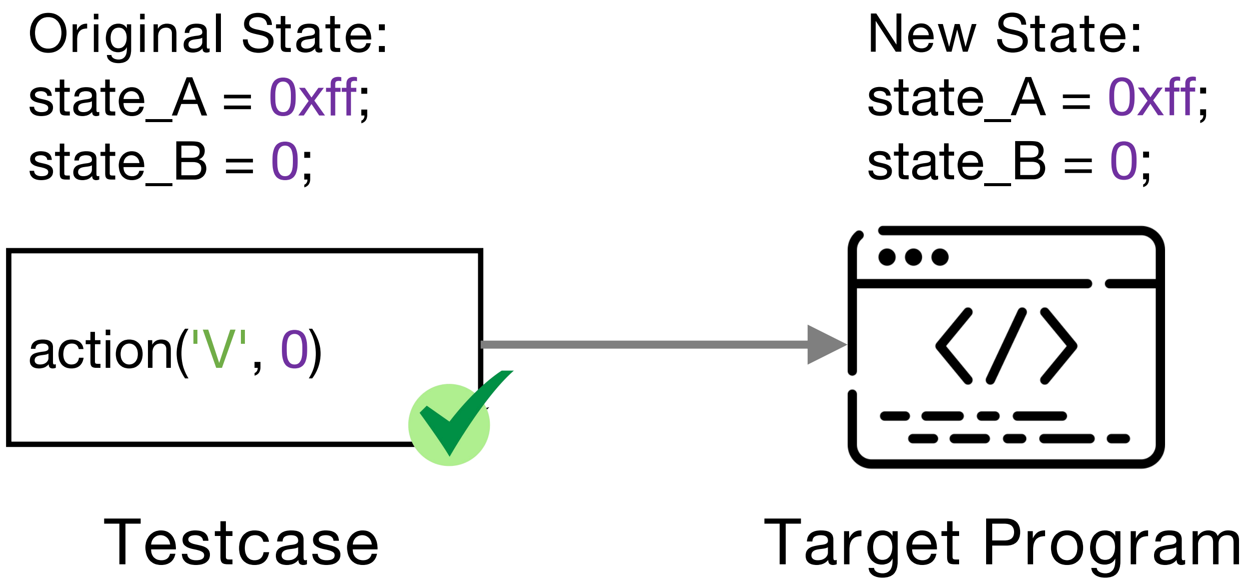
Testcase

Target Program

Hit new code, save this testcase.

Motivation Example

```c
int state_A = 0, state_B = 0;
int buf[BUF_SIZE];

void set_A(int value) {
    state_A = value;
}

void set_B(int value) {
    if (value<=BUF_SIZE && value>=0)
        state_B = value;
}

void vul(int value) {
    if (my_state_A == 0xff) {
        /* OOB bug here */
        buf[my_state_B] = value;
    }
}

void action(char op, int value) {
    switch (op) {
        case 'A': set_A(value);
        case 'B': set_B(value);
        case 'V': vul(value);
    }
}
```



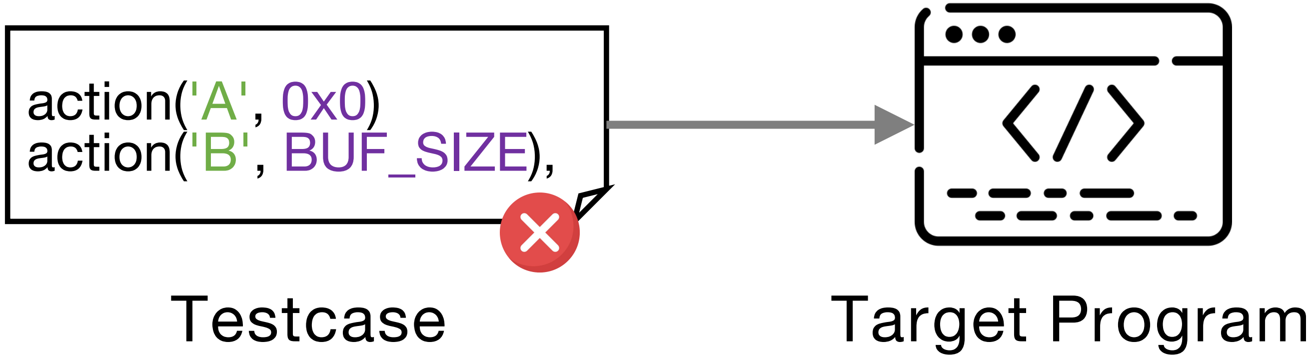
Hit new code, save this testcase.

Motivation Example

```c
int state_A = 0, state_B = 0;
int buf[BUF_SIZE];

void set_A(int value) {
    state_A = value;
}

void set_B(int value) {
    if (value<=BUF_SIZE && value>=0)
        state_B = value;
}

void vul(int value) {
    if (my_state_A == 0xff) {
        /* OOB bug here */
        buf[my_state_B] = value;
    }
}

void action(char op, int value) {
    switch (op) {
        case 'A': set_A(value);
        case 'B': set_B(value);
        case 'V': vul(value);
    }
}
```

Original State:
state_A = 0xff;
state_B = 0;

New State:
state_A = 0;
state_B = BUF_SIZE;



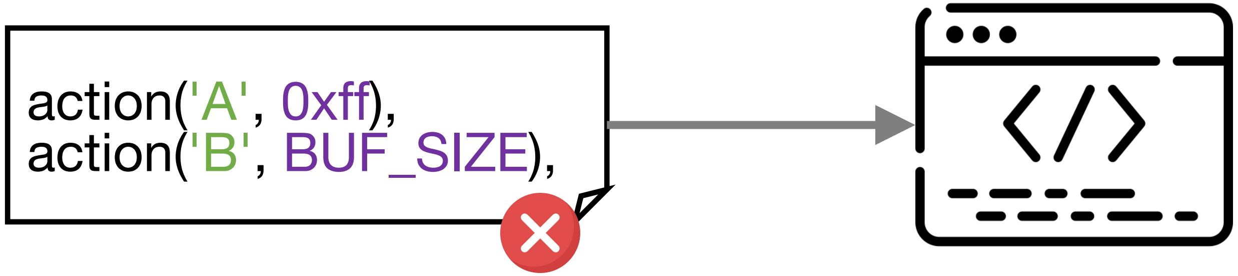
Hit no new code, discard this testcase.

Motivation Example

```c
int state_A = 0, state_B = 0;
int buf[BUF_SIZE];

void set_A(int value) {
    state_A = value;
}

void set_B(int value) {
    if (value<=BUF_SIZE && value>=0)
        state_B = value;
}

void vul(int value) {
    if (my_state_A == 0xff) {
        /* OOB bug here */
        buf[my_state_B] = value;
    }
}

void action(char op, int value) {
    switch (op) {
        case 'A': set_A(value);
        case 'B': set_B(value);
        case 'V': vul(value);
    }
}
```

Original State:
state_A = 0;
state_B = BUF_SIZE;

New State:
state_A = 0xff;
state_B = BUF_SIZE;



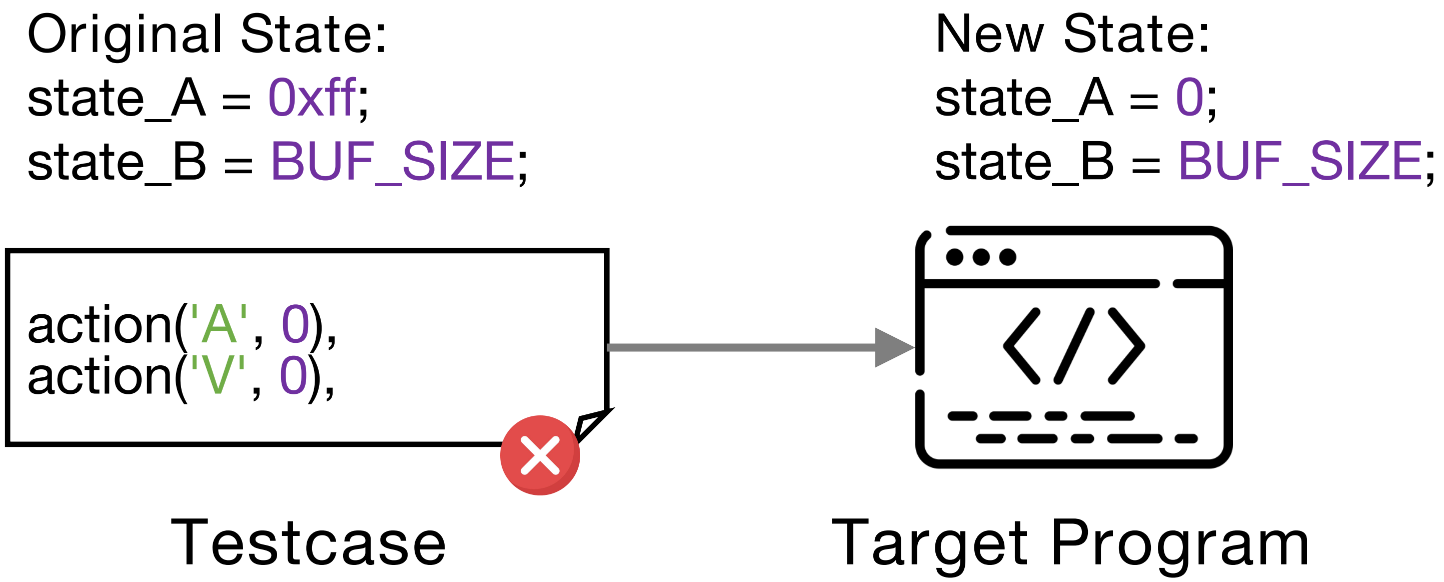
Testcase

Target Program

Hit no new code, discard this testcase.

Motivation Example

```c
int state_A = 0, state_B = 0;
int buf[BUF_SIZE];

void set_A(int value) {
    state_A = value;
}

void set_B(int value) {
    if (value<=BUF_SIZE && value>=0)
        state_B = value;
}

void vul(int value) {
    if (my_state_A == 0xff) {
        /* OOB bug here */
        buf[my_state_B] = value;
    }
}

void action(char op, int value) {
    switch (op) {
        case 'A': set_A(value);
        case 'B': set_B(value);
        case 'V': vul(value);
    }
}
```



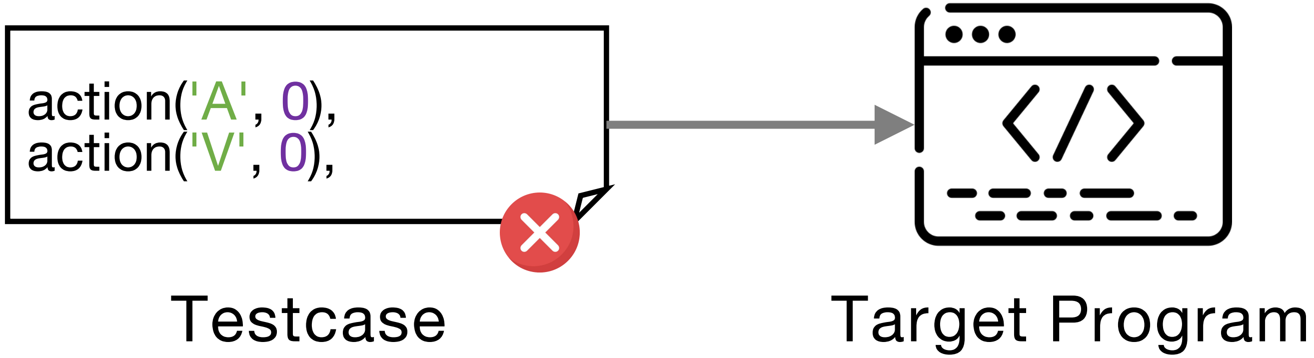
Hit no new code, discard this testcase.

Motivation Example

```c
int state_A = 0, state_B = 0;
int buf[BUF_SIZE];

void set_A(int value) {
    state_A = value;
}

void set_B(int value) {
    if (value<=BUF_SIZE && value>=0)
        state_B = value;
}

void vul(int value) {
    if (my_state_A == 0xff) {
        /* OOB bug here */
        buf[my_state_B] = value;
    }
}

void action(char op, int value) {
    switch (op) {
        case 'A': set_A(value);
        case 'B': set_B(value);
        case 'V': vul(value);
    }
}
```

Original State:
state_A = 0xff;
state_B = BUF_SIZE;

New State:
state_A = 0;
state_B = BUF_SIZE;



It is difficult to trigger the bug.

Motivation Example

```
int state_A = 0, state_B = 0;
int buf[BUF_SIZE];

void set_A(int value) {
    state_A = value;
}

void set_B(int value) {
    if (value<=BUF_SIZE && value>=0)
...
}

...
    }
}

void action(char op, int value) {
    switch (op) {
        case 'A': set_A(value);
        case 'B': set_B(value);
        case 'V': vul(value);
    }
}
```

state_A = 0;
state_B = BUF_SIZE;

Code coverage-guided fuzzers will ignore testcases
that exercise the same code path,
even though they have explored new states.

Testcase

Target Program

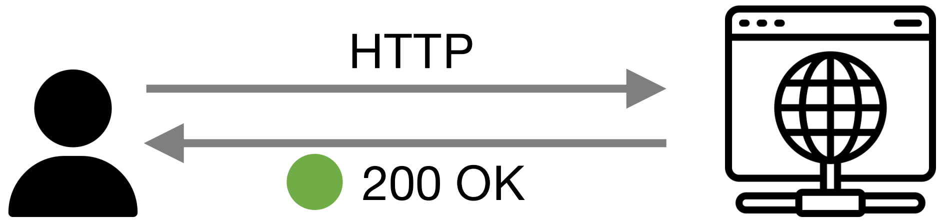
It is hard to trigger the bug.

How to perform state-aware fuzzing

- Three questions to answer
  - Q1: What are program states?
  - Q2: How to recognize and track program states?
  - Q3: How to utilize program states to guide fuzzing?

How to perform state-aware fuzzing

- Q1: What are program states?
  - Values of all memory and registers
    - the number of such states is overwhelmingly large
    - hard to track in practice
  - Manual annotation:
    - human efforts needed
  - Protocol status code:
    - not always available
  - Using variables to represent states is very common



We only focus on a subset of program states represented by variables.

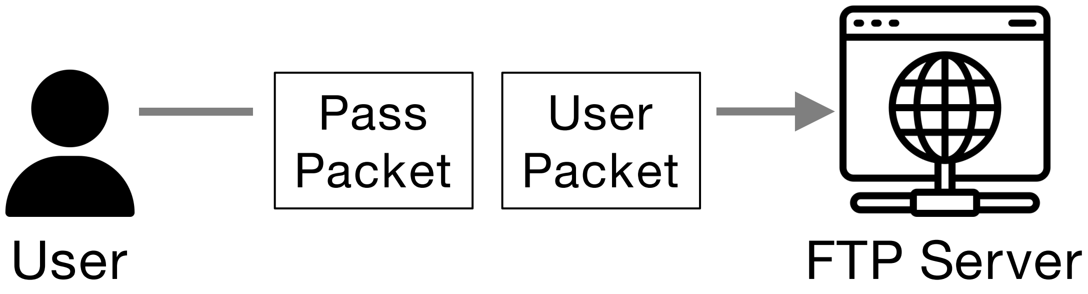
How to perform state-aware fuzzing

- Q1: What are program states?
- Q2: How to recognize and track program states?
  - We only focus on a subset of program states represented by variables.
  - The question is equivalent to how to recognize the state-variables (i.e., the variables that represent program states)?

Recognize State-Variables

- State-variables (i.e., the variables that represent program states)
  - have a long lifetime
  - can be updated (i.e., state transition) by users
  - can affect the program's control flow or memory access

- Observation
  - rich-state programs always require multi-stage inputs.
    - Each stage of input will trigger specific program actions.

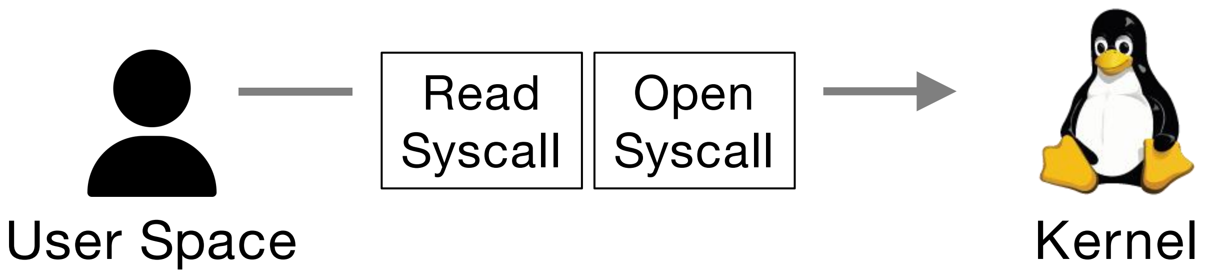



```
int ftpUSER(PFTPCONTEXT context, const char *params);

int ftpPASS(PFTPCONTEXT context, const char *params);
```

Recognize State-Variables

- State-variables (i.e., the variables that represent program states)
  - have a long lifetime
  - can be updated (i.e., state transition) by users
  - can affect the program’s control flow or memory access

- Observation
  - rich-state programs always require multi-stage inputs.
    - Each stage of input will trigger specific program actions.




```
static const struct file_operations hpet_fops = {
    ...
    .read = hpet_read,
    .open = hpet_open,
    ...
}
```

Recognize State-Variables

- State-variables (i.e., the variables that represent program states)
  - have a long lifetime
  - can be updated (i.e., state transition) by users
  - can affect the program's control flow or memory access

- Observation
  - rich-state programs always require multi-stage inputs.
  - state-variables are usually shared by different program actions

```
int ftpLIST(PFTPCONTEXT context, const char *params) {
    if (context->Access == FTP_ACCESS_NOT_LOGGED_IN)
        return sendstring(context, error530);
    ...
}
```

```
int ftpPASS(PFTPCONTEXT context, const char *params) {
    ...
    if ( strcasecmp(temptext, "admin") == 0 ) {
        context->Access = FTP_ACCESS_FULL;
        ...
}
```

How to perform state-aware fuzzing

- Q1: What are program states?
- Q2: How to recognize and track program states?
  - We only focus on a subset of program states represented by variables.
  - The question is equivalent to how to recognize the state-variables (i.e., the variables that represent program states)?

We can recognize state-variables by extracting the variables that have a long lifetime and shared by program actions.
We track program states by monitoring the state-variables.

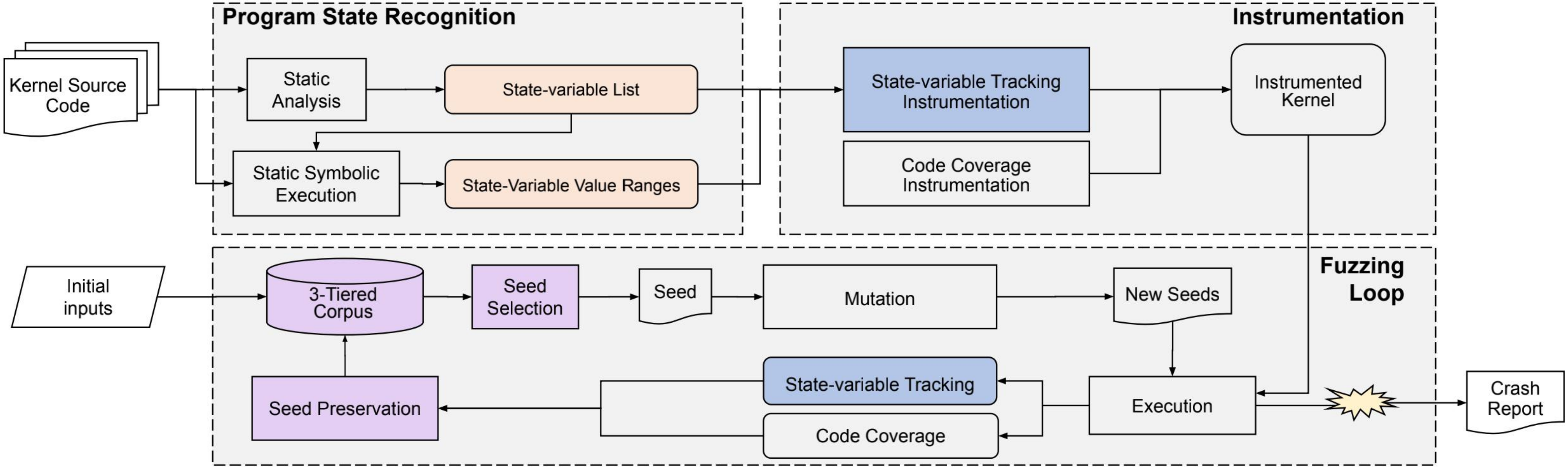
How to perform state-aware fuzzing

- Q1: What are program states?
- Q2: How to recognize and track program states?
- Q3: How to utilize program states to guide fuzzing?
  - Use state coverage as feedback for fuzzing
    - new value ---> new state?
    - too many values (e.g., 2^{32} for a 32-bit variable), causing seed queue explosion
    - merge values representing the same state
    - divide each state-variable's value space into several ranges

Instead of tracking values, we track special value ranges and extreme values of state-variables as feedback for fuzzing.

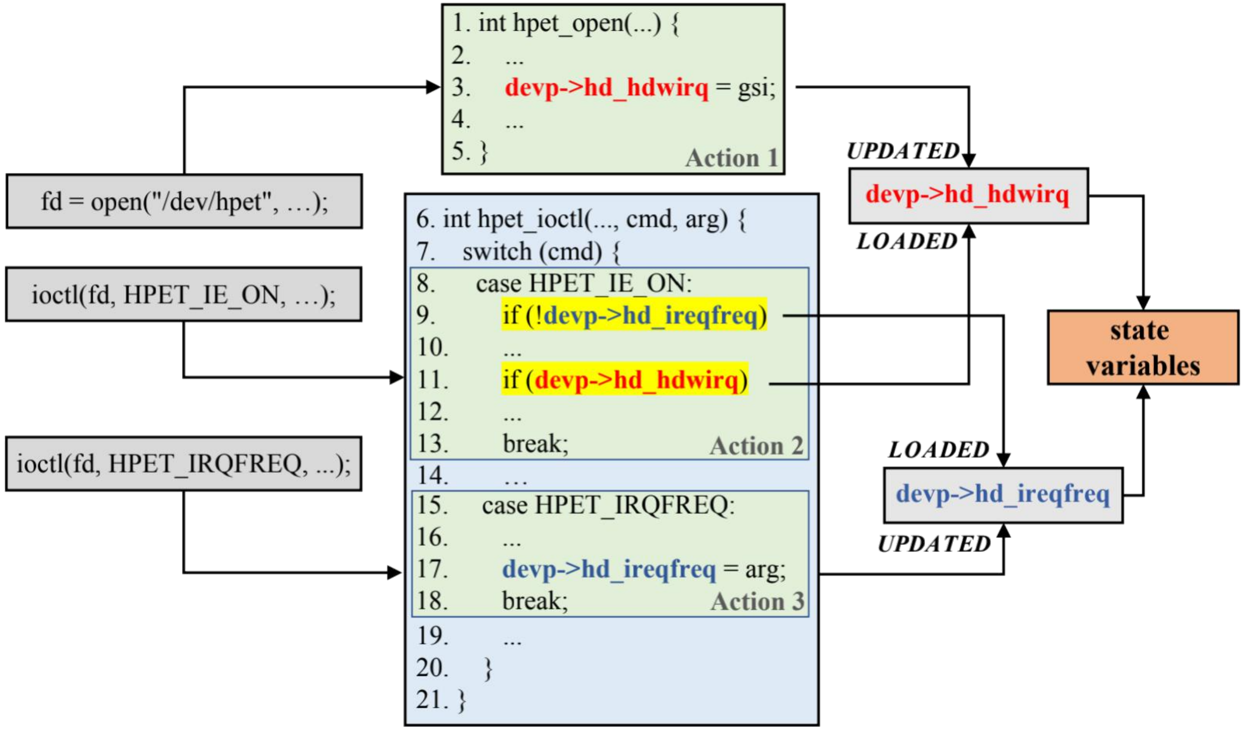
Our Approach: StateFuzz

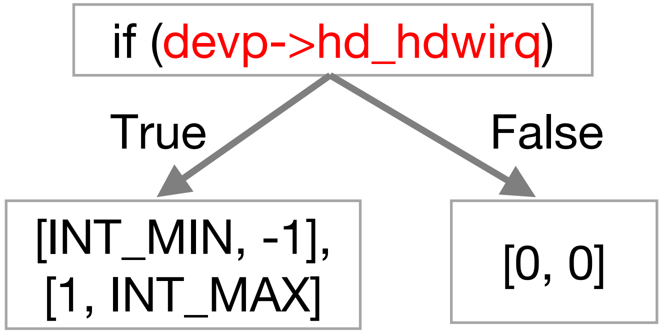
- A prototype for Linux driver fuzzing



Program State Recognition

- Identify program actions
  - handler functions that can be invoked via system calls
  - inter-procedural and path-sensitive analysis based on DIFUZE[1]
- Recognize state-variables
  - extract the variables that shared by program actions by static analysis.
- Infer state-variables' value ranges
  - inter-procedural and path-sensitive static symbolic execution





extract three value-ranges

[1] Corina, Jake, et al. "Difuze: Interface aware fuzzing for kernel drivers." CCS'17

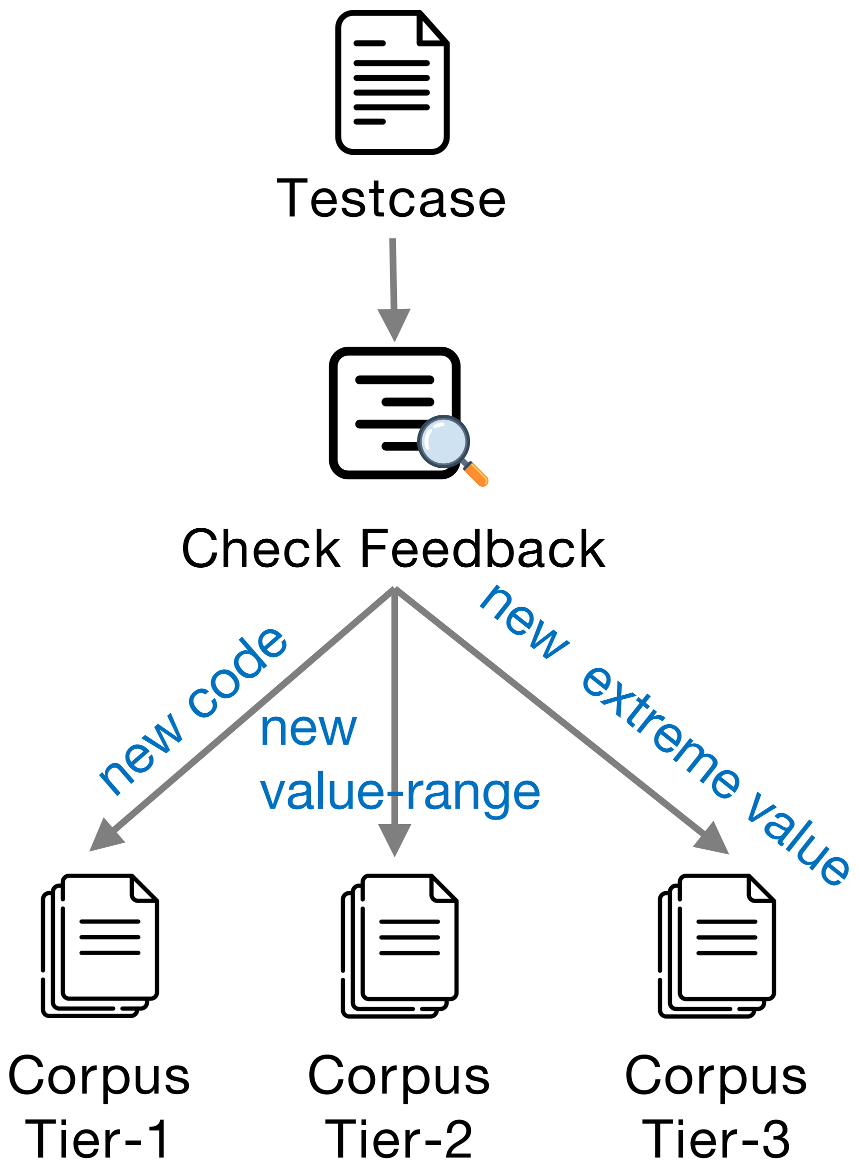
Instrumentation

- Track the stored values for state-variables
  - send the stored values as feedback for the fuzzer
- Use pointer-analysis to instrument alias of state-variables
- Code coverage instrumentation (kcov)

Fuzzing Loop

- Three-dimension feedback mechanism
  - Code coverage dimension
  - Value-range dimension
  - Extreme value dimension

- 3-Tiered corpus
  - seeds are saved based on feedback
  - select seeds from 3 tiers for mutation



Implementation

- State Recognition
  - DIFUZE (for program action recognition)
  - CRIX[2] (for building call graph)
  - Clang Static Analyzer (for static symbolic exectuion)
- Instrumentation
  - LLVM Sancov
  - SVF
- Fuzzing loop
  - Syzkaller

[2] Lu K, Pakki A, Wu Q. Detecting {Missing-Check} Bugs via Semantic-and {Context-Aware} Criticalness and Constraints Inferences(USENIX Security 19)

Evaluation

- RQ1: Are the state representation expressive and meaningful?
- RQ2: Can StateFuzz achieve higher coverage?
- RQ3: Can StateFuzz discover vulnerabilities in Linux drivers?
- Conduct experiments for Linux drivers in two environments:
  - Linux upstream kernel v4.19 on qemu-system-x86_64
  - Qualcomm MSM v4.14 kernel on a Google Pixel-4 phone

+ QEMU

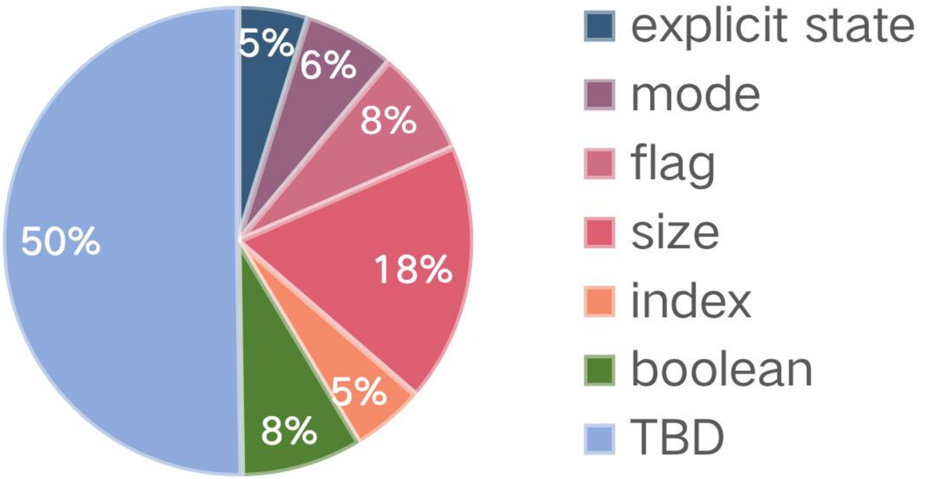
Qualcomm snapdragon +

Evaluation (1/3)

- RQ1: State Model Evaluation
  - Statistics of state-variables
    - ~3 value-ranges for every state-variable

Kernel	# Program Actions	# State-variables	# Value Ranges		
			Total	Avg.	Max
Linux-4.19	840	6055	18921	3.12	157
MSM-4.14	1330	5037	13332	2.65	193

  - Semantic of state-variables
    - by analyzing variable names in the AST



  - False positives and false negatives
    - recall of recognizing program actions: 99%
    - recall of recognizing state-variables: 90%
    - precision of recognizing state-variables: 40%

Evaluation (2/3)

- RQ2: Can StateFuzz achieve higher coverage?
  - state coverage
    - StateFuzz achieves 32% higher value-range coverage than Syzkaller in Linux-4.19
  - code coverage
    - StateFuzz achieves 19% higher code coverage than Syzkaller in Linux-4.19

Evaluation (3/3)

- RQ3: Vulnerability Discovery
  - StateFuzz found 20 vulnerabilities
    - 14 CVEs + ~$20,000 bug bounty from Google and Qualcomm

	Kernel	File	Function	Vulnerability Type	Status	CVE ID
1	Linux-4.19	drivers/input/keyboard/sunkbd.c	sunkbd_reinit	Use-after-free	Confirmed & Fixed	CVE-2020-25669
2		drivers/staging/speakup/spk_ttyio.c	spk_ttyio_ldisc_close	Null-pointer Dereference	Confirmed & Fixed	CVE-2020-28941
3		drivers/staging/speakup/spk_ttyio.c	spk_ttyio_receive_buf2	Null-pointer Dereference	Confirmed & Fixed	CVE-2020-27830
4		drivers/video/console/vgacon.c	vgacon_scrolldelta	Out-of-bounds Read	Confirmed & Fixed	CVE-2020-28097
5		drivers/md/dm-ioctl.c	list_devices	Out-of-bounds Write	Confirmed & Fixed	CVE-2021-31916
6		drivers/bluetooth/		Use-after-free	Reported	
7		drivers/tty/vt/		Deadlock	Confirmed	
8	MSM-4.14	drivers/mfd/adnc/iaxxx-module.c	iaxxx_core_sensor_change_state	Out-of-bounds Read	$Confirmed^B$ & Fixed	CVE-2021-0461
9		drivers/platform/msm/ipa/ipa_v3/ipa_utils.c	ipa3_counter_remove_hdl	Out-of-bounds Read	Confirmed & Fixed	CVE-2021-30265
10		drivers/char/diag/diag_pcie.c	diag_pcie_write	Out-of-bounds Write	$Confirmed^B$ & Fixed	CVE-2021-30298
11		drivers/char/diag/diag_dci.c	diag_send_dci_pkt_remote	Out-of-bounds Write	$Confirmed^B$ & Fixed	CVE-2021-30324
12		drivers/char/diag/diag_dci.c	extract_dci_pkt_rsp	Out-of-bounds Write	$Confirmed^B$ & Fixed	CVE-2021-30325
13		drivers/mfd/adnc/iaxxx-btp.c	iaxxx_btp_write_words	Out-of-bounds Read	$Confirmed^B$ & Fixed	CVE-2021-39717
14		drivers/misc/faceauth_hypx.c	hypx_create_blob_dmabuf	Use-after-free	$Confirmed^B$ & Fixed	CVE-2022-20183
15		drivers/misc/ipu/ipu-core-jqs-msg-transport.c	ipu_core_jqs_msg_transport_kernel_write_sync	Use-after-free	$Confirmed^B$ & Fixed	CVE-2022-20155
16		drivers/mfd/abc-pcie.c	abc_pcie_enter_el2_handler	Use-after-free	$Confirmed^B$ & Fixed	CVE-2022-20185
17		drivers/nfc/		Use-after-free	$Confirmed^B$	
18		drivers/char/diag/		Out-of-bounds Read	Confirmed	
19		drivers/platform/msm/ipa/ipa_v3/ipa_odl.c	ipa3_replenish_rx_cache	User-after-free	$Confirmed^*$ & Fixed	
20		drivers/char/adsprpc.c	get_args	Null-pointer Dereference	$Confirmed^*$ & Fixed	

Future Work

- Apply StateFuzz to network service fuzzing (NSFuzz)

- Apply StateFuzz to other Linux drivers (such as USB) that interact with users through multiple I/O channels rather than system calls.
  - hard to find program actions with static analysis
  - instead, we can trace the value-flow of inputs by lightweight instrumentation to dynamically find the program actions
  - then we can recognize state-variables in the same way as shown in this paper

Conclusion

- A new fuzzing solution StateFuzz for rich-states programs.
- StateFuzz models program states with state-variables.
- StateFuzz uses a new three-dimension feedback mechanism to help the fuzzer efficiently explore program states.
- We implemented a prototype for fuzzing Linux drivers.
- Experiments show that StateFuzz has better performance than Syzkaller in fuzzing Linux drivers.

Thanks!

Q&A

✉ zbd17@mails.tsinghua.edu, chaoz@tsinghua.edu.cn