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

Bodka Zhao¹, Zheming Li¹, Shisong Qin¹, Zheyu Ma¹, Ming Yuan¹, Wenyu Zhu², Zhihong Tian³, Chao Zhang¹,⁴

¹Institute for Network Science and Cyberspace / BNRist, Tsinghua University
²Department of Electronic Engineering, Tsinghua University
³Guangzhou University
⁴Zhongguancun Lab
Code Coverage Guided Fuzzing

Initial Inputs \rightarrow Seed Corpus \rightarrow seeds 

Seeds \rightarrow Seed Selection \rightarrow Seed Mutation 

Seed Mutation \rightarrow Target Program \rightarrow Crash 

New Code Coverage? \downarrow feedback
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;

Testcase

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

Testcase
action('A', 0)

New State:
state_A = 0;
state_B = 0;

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);
    }
}
```

Original State:
state_A = 0;
state_B = 0;

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

Testcase

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 = 0xff;
state_B = 0;

Testcase

```c
action('B', 0)
```

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);
    }
}
```

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

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

action('V', 0)

Testcase

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;

Testcase

```
action('A', 0x0)
action('B', BUF_SIZE),
```

Target Program

Hit no new code, discard this testcase.
### Motivation Example

```c
1 int state_A = 0, state_B = 0;
2 int buf[BUF_SIZE];
3
4 void set_A(int value) {
5     state_A = value;
6 }
7
8 void set_B(int value) {
9     if (value<=BUF_SIZE && value>=0)
10         state_B = value;
11 }
12
13 void vul(int value) {
14     if (my_state_A == 0xff) {
15         /* OOB bug here */
16         buf[my_state_B] = value;
17     }
18 }
19
20 void action(char op, int value) {
21     switch (op) {
22         case 'A': set_A(value);
23         case 'B': set_B(value);
24         case 'V': vul(value);
25     }
26 }
```

**Original State:**
- state_A = 0;
- state_B = BUF_SIZE;

**Testcase:**
- action('A', 0xff), action('B', BUF_SIZE);

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

**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);
    }
}
```

**Original State:**
- `state_A = 0xff`
- `state_B = BUF_SIZE`

**New State:**
- `state_A = 0`
- `state_B = BUF_SIZE`

**Testcase**
- `action('A', 0)`, `action('V', 0)`

Hit no new code, discard this testcase.
Motivation Example

```c
1 int state_A = 0, state_B = 0;
2 int buf[BUF_SIZE];
3
4 void set_A(int value) {
5     state_A = value;
6 }
7
8 void set_B(int value) {
9     if (value<=BUF_SIZE && value>=0)
10         state_B = value;
11 }
12
13 void vul(int value) {
14     if (my_state_A == 0xff) {
15         /* OOB bug here */
16         buf[my_state_B] = value;
17     }
18 }
19
20 void action(char op, int value) {
21     switch (op) {
22         case 'A': set_A(value);
23         case 'B': set_B(value);
24         case 'V': vul(value);
25     }
26 }
```

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

New State:
state_A = 0;
state_B = BUF_SIZE;

Testcase
action('A', 0),
action('V', 0),

Target Program

It is difficult to trigger the bug.
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 action(char op, int value) {
    switch (op) {
    case 'A': set_A(value);
    case 'B': set_B(value);
    case 'V': vul(value);
    }
}
```

It is hard to trigger the bug.

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

```
1 int ftpUSER(PFTPCONTTEXT context, const char *params);
2
3 int ftpPASS(PFTPCONTTEXT 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

```c
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

\[1\] 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\textsuperscript{[2]} (for building call graph)
  • Clang Static Analyzer (for static symbolic execution)

• Instrumentation
  • LLVM Sancov
  • SVF

• Fuzzing loop
  • Syzkaller

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
Evaluation (1/3)

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

<table>
<thead>
<tr>
<th>Kernel</th>
<th># Program Actions</th>
<th># State-variables</th>
<th># Value Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Total</td>
</tr>
<tr>
<td>Linux-4.19</td>
<td>840</td>
<td>6055</td>
<td>18921</td>
</tr>
<tr>
<td>MSM-4.14</td>
<td>1330</td>
<td>5037</td>
<td>13332</td>
</tr>
</tbody>
</table>

- explicit state
- mode
- flag
- size
- index
- boolean
- TBD
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
RQ3: Vulnerability Discovery

StateFuzz found 20 vulnerabilities

14 CVEs + ~$20,000 bug bounty from Google and Qualcomm
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