Registered Report: NSFuzz: Towards Efficient and State-Aware Network Service Fuzzing

Shisong Qin\textsuperscript{1}, Fan Hu\textsuperscript{2}, Bodong Zhao\textsuperscript{1}, Tingting Yin\textsuperscript{1}, Chao Zhang\textsuperscript{1}

1. Tsinghua University
2. State Key Laboratory of Mathematical Engineering and Advance
Vulnerability in Network Service

Vulnerabilities in network service enable attackers to launch remote exploits much easier than in local applications.

Heartbleed from OpenSSL
Remote Confidential Data Leakage

WannaCry from Microsoft’s SMB protocol
Ransomware Cyberattack
The workflow of coverage-guided grey-box fuzzing
## Related Work

<table>
<thead>
<tr>
<th>Black-Box Network Fuzzing</th>
<th>Grey-Box Network Fuzzing</th>
<th>Program State Model Inference</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPIKE</td>
<td>IoTHunter</td>
<td>Prospex</td>
</tr>
<tr>
<td>SNOOZE</td>
<td>yFuzz</td>
<td>PRETT</td>
</tr>
<tr>
<td>KiF</td>
<td>SGPFuzz</td>
<td>IJON</td>
</tr>
<tr>
<td>AutoFuzz</td>
<td>AFLNet</td>
<td>FuzzFactory</td>
</tr>
<tr>
<td>PULSAR</td>
<td>StateAFL</td>
<td>AFLNet</td>
</tr>
<tr>
<td>Peach</td>
<td></td>
<td>StateAFL</td>
</tr>
<tr>
<td>boofuzz</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Have limitations in **fuzzing efficiency** or **service state representation**
Features of Network Service

Multiple Network I/O Interactions

- FTP Client
  - USER username
  - 331 need password
  - PASS password
  - 230 user logged in
  - PASV
  - 227 passive mode
  - LIST dir
  - 226 data transferred
  - ...

- FTP Server

Involving State Transition (Stateful)

- RTSP protocol state model
Challenges in Network Service Fuzzing

- **Service State Representation**
  - Most existing grey-box fuzzers are mainly designed for local stateless applications
  - Fuzzer without state-aware may mislead the evolutionary direction of genetic algorithms due to the stateful of network services

**AFLNet\textsuperscript{[1]}**: Response code based state representation scheme

Challenges in Network Service Fuzzing

- **Testing Efficiency**

  - Network services are always designed as C/S architecture, requiring multiple I/O

  - Fuzzer needs to conduct multiple interactions to fuzz the service in-depth, and the control of interaction is vital to the fuzzing efficiency

---


Case Study

```c
int state = STATE_CONNECTED;
int main() {
    ...
    while (fgets(str, MAXCMD, stdin)) { // event loop
        parsecmd(str);
    }
}

int parsecmd(char *str) {
    ...
    for (i = 0; commands[i].name; i++) {
        if (!strncasecmp(str, commands[i].name, strlen(commands[i].name))){
            // state check
            if (state >= commands[i].state_needed) {
                commands[i].function(str); // invoke handler
                return 0;
            } else {
                switch (state) {
                    case STATE_CONNECTED:
                        response("503 USER expected");
                        return 1;
                    case STATE_USER:
                        response("503 PASS expected");
                        return 1;
                    case STATE_AUTHENTICATED:
                        response("503 RNFR before RNTO expected");
                        return 1;
                }
            }
        }
    }
}

void command_pass(char *password) {
    ...
    if (bftpd_login(password)) {
        state = STATE_CONNECTED; // state update
        return;
    }
}
```

Code snippet from FTP service BFTPD
Case Study

```c
int state = STATE_CONNECTED;
int main() {
    while (fgets(str, MAXCMD, stdin)) { // event loop
        parsecmd(str);
    }
    parsecmd(char *str) {
    ...
    for (i = 0; commands[i].name; i++) {
        if (!strcmp(str, commands[i].name, strlen(commands[i].name)))
            // state check
            if (state >= commands[i].state_needed) {
                commands[i].function(str); // invoke handler
                return 0;
            } else {
                switch (state) {
                    case STATE_CONNECTED:
                        response("503 USER expected");
                        return 1;
                    case STATE_USER:
                        response("503 PASS expected");
                        return 1;
                    case STATE_AUTHENTICATED:
                        response("503 RNFR before RNTO expected");
                        return 1;
                }
            }
        command_pass(char *password) {
            ...
            if (bftpdp_login(password)) {
                state = STATE_CONNECTED; // state update
                return;
            }
    }
}
```

Network Service

- Use an event loop to perform multiple I/O interactions

Code snippet from FTP service BFTPD
Network Service

- Use an event loop to perform multiple I/O interactions
- Use specific variable to record the current service state

```c
int state = STATE_CONNECTED;
int main() {
  ...
  while (fgets(str, MAXCMD, stdin)) { // event loop
    parsecmd(str);
  }
  ...
  int parsecmd(char *str) {
    ...
    for (i = 0; commands[i].name; i++) {
      if (!strcmpcase(str, commands[i].name, strlen(commands[i].name)))
        // state check
        if (state >= commands[i].state_needed) {
          commands[i].function(str); // invoke handler
          return 0;
        } else {
          switch (state) {
            case STATE_CONNECTED:
              response("503 USER expected");
              return 1;
            case STATE_USER:
              response("503 PASS expected");
              return 1;
            case STATE_AUTHENTICATED:
              response("503 RNR before RNDT expected");
              return 1;
          }
        }
    }
  }
  void command_pass(char *password) {
    ...
    if (bftpd_login(password)) {
      state = STATE_CONNECTED; // state update
      return;
    }
  }
}
Case Study

Network Service

- Use an event loop to perform multiple I/O interactions
- Use specific variable to record the current service state
- Execute different code according to current state, and update the state in specific handler

Code snippet from FTP service BFTPD
Insights

- **Service State Representation**
  - Network services always use some specific variables to represent the service state directly.
  - Such "state variable" could represent the service state more accurately and reasonably.

- **Testing Efficiency**
  - Network services always have some clear point to indicate the message processing status.
  - E.g., the beginning of event loop indicates the previous message has been handled.
  - Such "I/O sync point" could give fuzzer timely feedback to enable efficient I/O interaction.
Approach —— NSFuzz

An efficient and state-aware network service fuzzer

- Variable-based accurate service state representation
- Efficient network I/O synchronization mechanism
Overview Design

The workflow of NSFuzz

- Perform static analysis to identify the **event loop (I/O sync point)** and extract **state variables**

- Conduct compile-time instrumentation to enable the target to have the capabilities of signal-based **fast I/O synchronization** and variable-based **service state tracing**

- Carry out efficient and state-aware network service fuzzing loop
Static Analysis

- Event Loop Identification
  - Use the backtrace of probe message to identify event loop
    - network I/O contained, outermost in the nested loop

- State Variable Extraction
  - Use a series of heuristic rules to extract state variables
    - range constraint, operation constraint, variable constraint
Compile-Time Instrumentation

- Signal Feedback Instrumentation
  - Insert signal raising function at the I/O sync point to give fuzzer feedback
    - e.g., `raise(SIGSTOP)`
Compile-Time Instrumentation

- **Signal Feedback Instrumentation**
  - Insert signal raising function at the I/O sync point to give fuzzer feedback
    - e.g., `raise(SIGSTOP)`

- **State Tracing Instrumentation**
  - Setup another shared memory (shared_state) between fuzzer and SUT
  - Insert state tracing function at STORE operation of each state variable
    - $\text{shared}_\text{state}[\text{hash(var}_\text{id}) \oplus \text{cur}_\text{store}_\text{val}] = 1$
    - $\text{shared}_\text{state}[\text{hash(var}_\text{id}) \oplus \text{pre}_\text{store}_\text{val}] = 0$
Fuzzing Loop

- Fast I/O synchronization

The interaction process between fuzzer and SUT in each testcase
Fuzzing Loop

- Fast I/O synchronization
  - Each time the fuzzer sends a message, it waits for the signal feedback from service

The interaction process between fuzzer and SUT in each testcase
Fuzzing Loop

- **Fast I/O synchronization**
  - Each time the fuzzer sends a message, it waits for the signal feedback from service.
  - Service receives the message, processes it to update shared_state, then sends a signal.

The interaction process between fuzzer and SUT in each testcase.
Fuzzing Loop

- **Fast I/O synchronization**
  - Each time the fuzzer sends a message, it waits for the signal feedback from service.
  - Service receives the message, processes it to update shared_state, then sends a signal.
  - Fuzzer receives the signal, collects state representation, then sends the next message.

The interaction process between fuzzer and SUT in each testcase.
Fuzzing Loop

Service State Tracing

The process of shared_state update and state collection

shared_state

| var1 = 1 |
| 1        |
| 0        |
| 0        |

| var2 = 2 |
| 1        |
| 0        |
| 0        |
Fuzzing Loop

Service State Tracing

- Fuzzer hash the shared_state to collect state representation when receiving signal feedback

The process of shared_state update and state collection
Fuzzing Loop

- **Service State Tracing**
  - Fuzzer hash the shared_state to collect state representation when receiving signal feedback.
  - A change in any state variable would lead to a change in the hash of shared_state.

The process of shared_state update and state collection.
Fuzzing Loop

Service State Tracing

- Fuzzer hash the shared_state to collect state representation when receiving signal feedback

- A change in any state variable would lead to a change in the hash of shared_state

- Fuzzer continuous collects state to build transition sequence (model inference)

The process of shared_state update and state collection
Preliminary Evaluation on NSFuzz

➢ RQ1: Accurateness of state module inferred by NSFuzz

➢ Could NSFuzz inference relatively more accurate & reasonable state model based on the state variables during the fuzzing loop?

➢ RQ2: Effectiveness of NSFuzz state-aware fuzzing

➢ Could NSFuzz achieve higher fuzzing efficiency and overall results than other existing approaches?
Experiment Setup

- 7 targets from ProFuzzBench\textsuperscript{[1]}
- Compared with AFLNet\textsuperscript{[2]}/AFLNwe\textsuperscript{[3]}/StateAFL\textsuperscript{[4]}

<table>
<thead>
<tr>
<th>Target Service</th>
<th>Network Protocol</th>
<th>Version/Commit</th>
<th>Transport Layer</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>LightFTP</td>
<td>FTP</td>
<td>5980ea1</td>
<td>TCP</td>
<td>C</td>
</tr>
<tr>
<td>Bftpd</td>
<td>FTP</td>
<td>v5.7</td>
<td>TCP</td>
<td>C</td>
</tr>
<tr>
<td>Pure-FTPd</td>
<td>FTP</td>
<td>c21b45f</td>
<td>TCP</td>
<td>C</td>
</tr>
<tr>
<td>Exim</td>
<td>SMTP</td>
<td>38903fb</td>
<td>TCP</td>
<td>C</td>
</tr>
<tr>
<td>Dnsmasq</td>
<td>DNS</td>
<td>v2.73rc6</td>
<td>UDP</td>
<td>C</td>
</tr>
<tr>
<td>TinyDTLS</td>
<td>DTLS</td>
<td>06995d4</td>
<td>UDP</td>
<td>C</td>
</tr>
<tr>
<td>Kamailio</td>
<td>SIP</td>
<td>2648eb3</td>
<td>UDP</td>
<td>C</td>
</tr>
</tbody>
</table>

\textsuperscript{[1]} https://github.com/profuzzbench/profuzzbench
\textsuperscript{[2]} https://github.com/profuzzbench/aflnet
\textsuperscript{[3]} https://github.com/profuzzbench/aflnwe
\textsuperscript{[4]} https://github.com/stateafl/stateafl
# State Module Inference Evaluation (RQ1)

<table>
<thead>
<tr>
<th>Target Service</th>
<th>LoC</th>
<th>Network Event Loop</th>
<th>State Variable</th>
<th>Analysis Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>LightFTP</td>
<td>4.4k</td>
<td>√</td>
<td>1 Access</td>
<td>0.7s</td>
</tr>
<tr>
<td>Bftpd</td>
<td>4.7k</td>
<td>√</td>
<td>6 state</td>
<td>1.8s</td>
</tr>
<tr>
<td>Pure-FTPD</td>
<td>30k</td>
<td>√</td>
<td>22 loggedin</td>
<td>3.9s</td>
</tr>
<tr>
<td>Exim</td>
<td>101.7k</td>
<td>√</td>
<td>58 helo_seen</td>
<td>45.1s</td>
</tr>
<tr>
<td>Dnsmasq</td>
<td>27.6k</td>
<td>√</td>
<td>15 found</td>
<td>11.4s</td>
</tr>
<tr>
<td>TinyDTLS</td>
<td>10.8k</td>
<td>√</td>
<td>4 state</td>
<td>3.2s</td>
</tr>
<tr>
<td>Kamailio</td>
<td>766.7k</td>
<td>√</td>
<td>58 state</td>
<td>441.9s</td>
</tr>
</tbody>
</table>
### State Module Inference Evaluation (RQ1)

<table>
<thead>
<tr>
<th>Target Service</th>
<th>Fuzzer</th>
<th>State Module</th>
<th>Vertexes</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>LightFTP</td>
<td>AFLNET</td>
<td>23</td>
<td>158</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STATEAFL</td>
<td>11</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NSFuzz</td>
<td>5</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Bftpd</td>
<td>AFLNET</td>
<td>24</td>
<td>126</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STATEAFL</td>
<td>4</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NSFuzz</td>
<td>43</td>
<td>137</td>
<td></td>
</tr>
<tr>
<td>Pure-FTPd</td>
<td>AFLNET</td>
<td>27</td>
<td>260</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STATEAFL</td>
<td>7</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NSFuzz</td>
<td>8</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>Exim</td>
<td>AFLNET</td>
<td>12</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STATEAFL</td>
<td>7</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NSFuzz</td>
<td>128</td>
<td>225</td>
<td></td>
</tr>
<tr>
<td>Dnsmasq</td>
<td>AFLNET</td>
<td>89</td>
<td>271</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STATEAFL</td>
<td>108</td>
<td>467</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NSFuzz</td>
<td>3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>TinyDTLS</td>
<td>AFLNET</td>
<td>9</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STATEAFL</td>
<td>29</td>
<td>69</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NSFuzz</td>
<td>32</td>
<td>115</td>
<td></td>
</tr>
<tr>
<td>Kamailio</td>
<td>AFLNET</td>
<td>13</td>
<td>93</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STATEAFL</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NSFuzz</td>
<td>99</td>
<td>328</td>
<td></td>
</tr>
</tbody>
</table>

The state model inferred by various fuzzers

The state model of LightFTP inferred by NSFuzz
## Fuzzing Efficiency Evaluation (RQ2)

<table>
<thead>
<tr>
<th>Target Service</th>
<th>Fuzzing Throughput (exec/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AFLNet</td>
</tr>
<tr>
<td>LightFTP</td>
<td>8.42</td>
</tr>
<tr>
<td>Bftpd</td>
<td>4.09</td>
</tr>
<tr>
<td>Pure-FTPD</td>
<td>5.29</td>
</tr>
<tr>
<td>Exim</td>
<td>2.69</td>
</tr>
<tr>
<td>Dnsmasq</td>
<td>7.47</td>
</tr>
<tr>
<td>TinyDTLS</td>
<td>2.66</td>
</tr>
<tr>
<td>Kamailio</td>
<td>5.19</td>
</tr>
</tbody>
</table>

The average fuzzing throughput of various fuzzers toward each target service
Fuzzing Efficiency Evaluation (RQ2)

The average branch coverage growth in 12h of various fuzzers toward each target service
## Fuzzing Efficiency Evaluation (RQ2)

| Target Service | Crash Trigger Time (s) |  |
|----------------|------------------------|--|---|
|                | AFLNet | AFLNwe | StateAFL | NSFuzz |
| Dnsmasq        | 990.5s | 989.25s | 878.75s  | 160s    |
| TinyDTLS       | 26s    | 11.75s  | 47.75s   | < 1s    |

The average crash trigger time of various fuzzers toward each target service.
Limitations

- **Scalability**
  - **Service Pattern** Support (libevent-based target)
  - **Service Language** Support (other than C)
  - **False Positive** in state variable extraction (leading to state explosion)

The fragile of **Static Analysis** is the main reason (e.g., ad-hoc analysis rules...)
Conclusion

- Analyzed the *state representation* and *testing efficiency* challenges of network service fuzzing

- Proposed NSFuzz, a network service fuzzer combined with variable-based state representation and efficient I/O synchronization

- Preliminary evaluated NSFuzz on ProFuzzBench, and the results showed NSFuzz could infer a accurate state model and achieve a higher fuzzing efficiency than some other existing solutions
Ongoing Work

- **Annotation API**
  - **I/O Sync Point Annotation**
    - Multiple I/O point supported
    - libevent-based target supported
  - **State Variable Annotation**
    - Eliminate false positive
    - Precise annotation

```c
int main(int argc, char **argv) {
    ... // service initialization
    while (fgets(str, MAXCMD, sock)) {
       // I/O sync point annotation API
       _NSFUZZ_SYNC();
       ...
       parsecmd(str);
    }
    ... return 0;
}
```

```c
enum {
    STATE_CONNECTED, STATE_USER,
    STATE_AUTHENTICATED, STATE_RENAME, STATE_ADMIN
};
// state variable annotation API (global variable)
int _NSFUZZ_STATE(state) = STATE_CONNECTED;
```
Ongoing Work

- Ablation Study

The average branch coverage growth in 12h of various fuzzers toward each target service.

**NSFuzz-V**: NSFuzz with variable-based state representation only enabled.
Thanks for Listening!

Q & A

Contact: qss19@mails.tsinghua.edu.cn