Stateful Greybox Fuzzing

Jinsheng Ba\textsuperscript{1}, Marcel Böhme\textsuperscript{2,3}, Zahra Mirzamomen\textsuperscript{2}, and Abhik Roychoudhury\textsuperscript{1}
\textsuperscript{1}National University of Singapore, \textsuperscript{2}Monash University, \textsuperscript{3}MPI-SP
Challenge: Bugs in Stateful Programs
Challenge: Bugs in Stateful Programs
Challenge: Bugs in Stateful Programs

Problem: How to efficiently find stateful bugs?
Challenge: Bugs in Stateful Programs

**Challenge:** Cover the state space without a specification of the required event sequences.
Past Works

• AFL, LibFuzzer

Stateless fuzzers that cannot generate a sequence of inputs in specific orders.

• AFLNet

Can not represent program internal states

• IJON

Requires human knowledge (state specification)
Protocol Implementations (Stateful Programs)
Protocol Implementations (Stateful Programs)

Client

Header Frame

Data Frame

Server

(HTTP/2 Protocol)
State Identification

• They are represented by state variables with named constants.
State Identification

- They are represented by state variables with named constants.

```c
typedef enum enum_h2o_http2_stream_state_t {
    /* stream in idle state (but registered; i.e. priority stream)
    */
    H2O_HTTP2_STREAM_STATE_IDLE,
    /* receiving headers */
    H2O_HTTP2_STREAM_STATE_RECV_HEADERS,
    /* receiving body (or trailers), waiting for the arrival of END_STREAM */
    H2O_HTTP2_STREAM_STATE_RECV_BODY,
    /* received request but haven't been assigned a handler */
    H2O_HTTP2_STREAM_STATE_REQ_PENDING,
    ...
} H2O_HTTP2_STREAM_STATE;
```

```c
typedef enum {
    TLS_ST_BEFORE,
    TLS_ST_OK,
    DTLS_ST_CR_HELLO_VERIFY_REQUEST,
    TLS_ST_CR_SRVR_HELLO,
    TLS_ST_CR_CERT,
    TLS_ST_CR_CERT_STATUS,
    TLS_ST_CR_KEY_EXCH,
    TLS_ST_CR_CERT_REQ,
    TLS_ST_CR_SRVR_DONE,
    TLS_ST_CR_SESSION_TICKET,
    TLS_ST_CR_CHANGE,
    TLS_ST_CR_FINISHED,
    TLS_ST_CLNT_HELLO,
    TLS_ST_CLNT_CERT,
    TLS_ST_CLNT_KEY_EXCH,
    TLS_ST_CLNT_CERT_VERIFY,
    TLS_ST_CLNT_CHANGE,
    TLS_ST_CLNT_NEXT_PROTO,
    ...
} TLS_STATE;
```
Top-50 most widely used protocol implementations use named constants to represent protocol states.

44 of 50 use enumeration type, 6 use #define macro, to define the named constants. Including 16 protocols: FTP, SFTP, TLS, SMTP, HTTP2, RDP, NTP, IMAP, IRC, SMB, DAAP, SIP, DICOM, VNC, RTSP, MQTT.
static void handle_request_body_chunk(
    ...
    if (stream->req.write_req.cb(stream->req.write_req.ctx, payload,
        is_end_stream) != 0) {
        ...
    }
}

static int handle_data_frame(...h2o_http2_stream_t *stream...){
    ...
    if (stream->state != H2O_HTTP2_STREAM_STATE_RECV_BODY & & ...){
        ...
        return 0;
    }
    handle_request_body_chunk(...);
    ...
}

static int handle_incoming_request(...h2o_http2_stream_t *stream...){
    ...
    h2o_http2_stream_set_state(conn, stream,
        H2O_HTTP2_STREAM_STATE_RECV_BODY);
    ...
}

static int handle_headers_frame(...h2o_http2_frame_t *frame...){
    ...
    if (((frame->flags & H2O_HTTP2_FRAME_FLAG_END_HEADERS) != 0) { /* request headers are complete, handle it */
        return handle_incoming_request(conn, stream, ...);
    }
    ...
    return 0;
}

• stream->state
• stream->state
• stream->state
Insight

Approximating state variables by the variables with named constants.

• There may be variables take named constants that are not state variables (e.g., configuration variables, error code variables)
• In our evaluation, over 99% of extracted variables are true states.
Stateful Greybox Fuzzing

- Branch coverage
- Feedback
- Code coverage map
- State Feedback
- State Transition Tree
- Collection
- Fuzzer
- Mutate and execute inputs
- State schedule
In fuzzing, we monitor the changes of values of enumeration variables.

State Transition Tree Construction

Monitor the variable:
stream->state
In fuzzing, we monitor the changes of values of enumeration variables.

Monitor the variable: stream->state
In fuzzing, we monitor the changes of values of enumeration variables.

Monitor the variable: stream->state
In fuzzing, we monitor the changes of values of enumeration variables.

Monitor the variable: stream->state
Is State Transition Tree Meaningful?

Extracting the state machine from the State Transition Tree by merging the nodes with the same value.
Is State Transition Tree Meaningful?

Compare with the official document.

https://datatracker.ietf.org/doc/html/rfc7540#section-5.1
Is State Transition Tree Meaningful?

Compare with the official document.

https://datatracker.ietf.org/doc/html/rfc7540#section-5.1
State Fuzzing Algorithm

1. Save the inputs that trigger new state transitions.
State Fuzzing Algorithm

1. Save the inputs that trigger new state transitions.
2. Assign more energy on the “core-logic” state sequences.
Valuable State Transition Sequence

- **IDLE**
- **RECV_HEADERS**
- **RECV_BODY**
- **REQ_PENDING**
- **SEND_HEADERS**
- **SEND_BODY**
- **BODY_IS_FINAL**
- **END_STREAM**

**Error Handling**

**Core Logic**

- 80% ~ 99%
- 1% ~ 20%

**Error Handle**

**Core Logic**
1. Save the inputs that trigger new state transitions.
2. Assign more energy on the “core-logic” state sequences.

For each input $I$:
- $\text{Num of inputs mutated from } I$
- $\text{Num of mutated inputs that still execute the same state sequence}$
State Fuzzing Algorithm

1. Save the inputs that trigger new state transitions.
2. Assign more energy on the “core-logic” state sequences.
3. Correlate input bytes and state transitions, giving more opportunities on mutating these bytes.
1. Save the inputs that trigger new state transitions.
2. Assign more energy on the “core-logic” state sequences.
3. Correlate input bytes and state transitions, giving more opportunities on mutating these bytes.
1. Save the inputs that trigger new state transitions.
2. Assign more energy on the “core-logic” state sequences.
3. Correlate input bytes and state transitions, giving more opportunities on mutating these bytes.
Implementation

SGFuzz is built on top of LibFuzzer:

10x faster than AFL (In-memory VS Fork).
Implementation

Automatically search the assignments to the enumeration variables by regex match

Source code

```c
... conn->state = H2O_HTTP2_CONN_STATE_HALF_CLOSED;
...
```

Instrument source code by Python

---

Process

SGFuzz

LibFuzzer

Instrumented Program

```c
...
{__sfuzzer_instrument(855, H2O_HTTP2_CONN_STATE_HALF_CLOSED );
conn->state = H2O_HTTP2_CONN_STATE_HALF_CLOSED;}
...
```
Implementation

Source code

```c
...  
  conn->state = H2O_HTTP2_CONN_STATE_HALF_CLOSED;
...
```

Instrument source code by Python

Process

SGFuzz

LibFuzzer

Instrumented Program

```c
...
  __sfuzzer_instrument(855, H2O_HTTP2_CONN_STATE_HALF_CLOSED);
  conn->state = H2O_HTTP2_CONN_STATE_HALF_CLOSED;
}  
...
```
## Evaluation: Benchmarks

<table>
<thead>
<tr>
<th>Subject</th>
<th>Protocol</th>
<th>Fuzz Driver</th>
<th>Commit</th>
<th>Size</th>
<th>Share</th>
</tr>
</thead>
<tbody>
<tr>
<td>H2O</td>
<td>HTTP</td>
<td>h2o-fuzzer-http2</td>
<td>1e7344</td>
<td>337kLoC</td>
<td>12kIPs</td>
</tr>
<tr>
<td>MbedTLS</td>
<td>SSL/TLS</td>
<td>dtlsserver</td>
<td>e483a7</td>
<td>138kLoC</td>
<td>8mIPs</td>
</tr>
<tr>
<td>Curl</td>
<td>Several</td>
<td>curl_fuzzer</td>
<td>aab3a7</td>
<td>202kLoC</td>
<td>-</td>
</tr>
<tr>
<td>Gstreamer</td>
<td>Custom</td>
<td>gst-discoverer</td>
<td>44bdad</td>
<td>235kLoC</td>
<td>15kIPs</td>
</tr>
<tr>
<td>OpenSSL</td>
<td>SSL/TLS</td>
<td>netdriver</td>
<td>1e08f3</td>
<td>673kLoC</td>
<td>&gt;10mIPs</td>
</tr>
<tr>
<td>Live555</td>
<td>RTSP</td>
<td>netdriver</td>
<td>21.Aug'08</td>
<td>17kLoC</td>
<td>12kIPs</td>
</tr>
<tr>
<td>OwnTone</td>
<td>DAAP</td>
<td>netdriver</td>
<td>774d7c</td>
<td>37kLoC</td>
<td>10kIPs</td>
</tr>
<tr>
<td>DCMTK</td>
<td>DICOM</td>
<td>netdriver</td>
<td>24efbf4</td>
<td>38kLoC</td>
<td>3kIPs</td>
</tr>
</tbody>
</table>

Share represents the number of IPs running it on the Internet according to Shodan(www.shodan.io)

23 hours & 20 runs
State Transition Coverage

- We measure the number of state transition sequences in the State Transition Tree

<table>
<thead>
<tr>
<th>Subject</th>
<th>AFLNet</th>
<th>LibFuzzer</th>
<th>IJSN</th>
<th>SGFuzz</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>H2O</td>
<td>-</td>
<td>70.80</td>
<td>91.85</td>
<td>1849.30</td>
<td>26.1</td>
</tr>
<tr>
<td>MbedTLS</td>
<td>-</td>
<td>22.80</td>
<td>32.45</td>
<td>50.80</td>
<td>2.2</td>
</tr>
<tr>
<td>Curl</td>
<td>-</td>
<td>150.25</td>
<td>375.75</td>
<td>14630.80</td>
<td>97.3</td>
</tr>
<tr>
<td>Gstreamer</td>
<td>-</td>
<td>49.40</td>
<td>134.20</td>
<td>4067.30</td>
<td>82.3</td>
</tr>
<tr>
<td>OpenSSL</td>
<td>13.25</td>
<td>23.95</td>
<td>29.60</td>
<td>33.10</td>
<td>1.4</td>
</tr>
<tr>
<td>Live555</td>
<td>138.27</td>
<td>184.15</td>
<td>405.3</td>
<td>1162.30</td>
<td>6.3</td>
</tr>
<tr>
<td>OwnTone</td>
<td>1.00</td>
<td>46.40</td>
<td>426.00</td>
<td>930.15</td>
<td>20.0</td>
</tr>
<tr>
<td>DCMTK</td>
<td>68.10</td>
<td>189.25</td>
<td>267.50</td>
<td>6737.05</td>
<td>35.6</td>
</tr>
</tbody>
</table>

On average, SGFuzz covers state transition sequences 30 times more than the baseline LibFuzzer.
## State Identification Effectiveness

<table>
<thead>
<tr>
<th>Subject</th>
<th>All Nodes</th>
<th>State</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>H2O</td>
<td>6418</td>
<td>6417</td>
<td>99.98%</td>
</tr>
<tr>
<td>MbedTLS</td>
<td>167</td>
<td>167</td>
<td>100.00%</td>
</tr>
<tr>
<td>Curl</td>
<td>35690</td>
<td>35629</td>
<td>99.83%</td>
</tr>
<tr>
<td>Gstreamer</td>
<td>11240</td>
<td>11224</td>
<td>99.86%</td>
</tr>
<tr>
<td>OpenSSL</td>
<td>817</td>
<td>789</td>
<td>96.57%</td>
</tr>
<tr>
<td>Live555</td>
<td>17446</td>
<td>17446</td>
<td>100.00%</td>
</tr>
<tr>
<td>OwnTone</td>
<td>3671</td>
<td>3671</td>
<td>100.00%</td>
</tr>
<tr>
<td>DCMTK</td>
<td>27178</td>
<td>27109</td>
<td>99.75%</td>
</tr>
</tbody>
</table>

**Avg: 99.50%**

In our data structure State Transition Tree, 99.5% nodes are related to the true states.
## New Bugs

<table>
<thead>
<tr>
<th>Subject</th>
<th>Version</th>
<th>Type</th>
<th>Stateful</th>
<th>CVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Live555</td>
<td>1.08</td>
<td>Stack-based overflow in liveMedia/MP3FileSource.cpp</td>
<td>✔️</td>
<td>CVE-2021-38380</td>
</tr>
<tr>
<td>Live555</td>
<td>1.08</td>
<td>Heap use after free in liveMedia/MatroskaFile.cpp</td>
<td>✔️</td>
<td>CVE-2021-38381</td>
</tr>
<tr>
<td>Live555</td>
<td>1.08</td>
<td>Heap use after free in liveMedia/MPEG1or2Demux.cpp</td>
<td>✔️</td>
<td>CVE-2021-38382</td>
</tr>
<tr>
<td>Live555</td>
<td>1.08</td>
<td>Memory leak in liveMedia/AC3AudioStreamFramer.cpp</td>
<td>✔️</td>
<td>CVE-2021-39282</td>
</tr>
<tr>
<td>Live555</td>
<td>1.08</td>
<td>Assertion in UsageEnvironment/UsageEnvironment.cpp</td>
<td>✔️</td>
<td>CVE-2021-39283</td>
</tr>
<tr>
<td>Live555</td>
<td>1.08</td>
<td>Heap-based overflow in BasicUsageEnvironment/BasicTaskScheduler.cpp</td>
<td>✔️</td>
<td>CVE-2021-41396</td>
</tr>
<tr>
<td>Live555</td>
<td>1.08</td>
<td>Memory leak in liveMedia/MPEG1or2Demux.cpp</td>
<td>✔️</td>
<td>CVE-2021-41397</td>
</tr>
<tr>
<td>OwnTone</td>
<td>28.2</td>
<td>Heap use after free in src/misc.c</td>
<td>✗</td>
<td>CVE-2021-38383</td>
</tr>
<tr>
<td>DCMTK</td>
<td>3.6.6</td>
<td>Memory leak in dcmnet/libsrc/dulparse.cc</td>
<td>✗</td>
<td>CVE-2021-41687</td>
</tr>
<tr>
<td>DCMTK</td>
<td>3.6.6</td>
<td>Memory leak in dcmnet/libsrc/dulparse.cc</td>
<td>✔️</td>
<td>CVE-2021-41688</td>
</tr>
<tr>
<td>DCMTK</td>
<td>3.6.6</td>
<td>Heap use after free in dcmqrdb/libsrc/dcmqrsrv.cc</td>
<td>✔️</td>
<td>CVE-2021-41689</td>
</tr>
<tr>
<td>DCMTK</td>
<td>3.6.6</td>
<td>Heap-based overflow in dcmnet/libsrc/diutil.cc</td>
<td>✔️</td>
<td>CVE-2021-41690</td>
</tr>
</tbody>
</table>

We found 12 previously unknown bugs in 23 hours, and 10 of 12 are stateful bugs.
Conclusion

Challenge: Bugs in Stateful Programs

Problem: How to efficiently find stateful bugs?

Protocol Implementations (Stateful Programs)

Client

Server

(Data/Wire Protocol)

Insight

Approximating state variables by the variables with named constants.

- There may be variables take named constants that are not state variables (e.g., configuration variables, error code variables)
- In our evaluation, over 99% of extracted variables are true states.

Stateful Greybox Fuzzing

Branch coverage feedback

State feedback

State Transition Tree Construction

In fuzzing, we monitor the changes of values of enumeration variables.

State Transition Tree

New Bugs

<table>
<thead>
<tr>
<th>Bug</th>
<th>Description</th>
<th>Size</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug1</td>
<td>Stack-based overflow in blockAlert/MyStart.cpp</td>
<td>Small</td>
<td>2021-01-01</td>
</tr>
<tr>
<td>Bug2</td>
<td>Heap use after free in blockAlert/Mysandbox.cpp</td>
<td>Large</td>
<td>2021-02-02</td>
</tr>
<tr>
<td>Bug3</td>
<td>Memory leak in blockAlert/MyStart.cpp</td>
<td>Medium</td>
<td>2021-03-03</td>
</tr>
<tr>
<td>Bug4</td>
<td>Assertion in UsageEnvironment/UsageEnvironment.cpp</td>
<td>Small</td>
<td>2021-04-04</td>
</tr>
<tr>
<td>Bug5</td>
<td>Heap-based overflow in blockAlert/MyStart.cpp</td>
<td>Large</td>
<td>2021-05-05</td>
</tr>
<tr>
<td>Bug6</td>
<td>Memory leak in blockAlert/MyStart.cpp</td>
<td>Medium</td>
<td>2021-06-06</td>
</tr>
</tbody>
</table>

We found 12 previously unknown bugs in 23 hours, and 10 of 12 are stateful bugs.

jinsheng@comp.nus.edu.sg
https://github.com/bajinsheng/SGFuzz