Stateful Greybox Fuzzing

Jinsheng Ba[1] , Marcel Böhme[2,3], Zahra Mirzamomen[2] , and Abhik Roychoudhury[1]
[1]National University of Singapore, [2]Monash University, [3]MPI-SP
USENIX ’22

mhkang 2022-11-28

Outline

- Introduction
- Related works
- Automatic state identification
- Stateful greybox fuzzer
- State fuzzing algorithm
- Evaluation
- Conclusion

Introduction

- Stateful programs
  - require input messages to be sent in a certain expected order
  - e.g., protocol implementations

- Stateful bugs
  - triggered when a sequence of messages, events, or actions are given as an input

Introduction

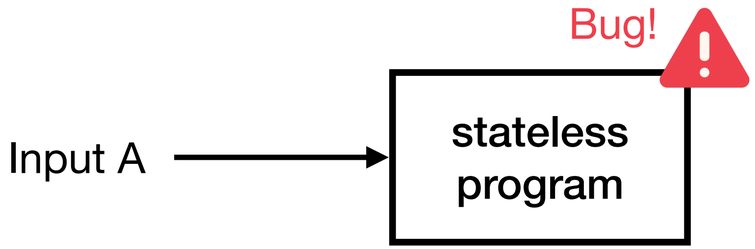
- Stateful programs
  - require input messages to be sent in a certain expected order
  - e.g., protocol implementations

- Stateful bugs
  - triggered when a sequence of messages, events, or actions are given as an input

Introduction

- Stateful programs
  - require input messages to be sent in a certain expected order
  - e.g., protocol implementations

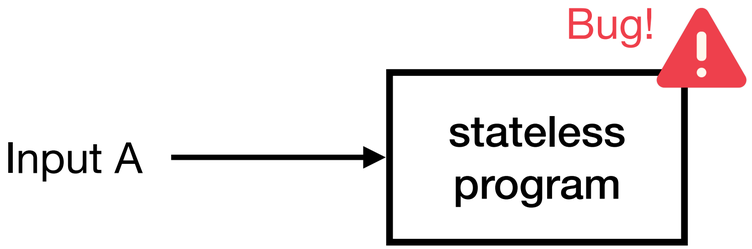
- Stateful bugs
  - triggered when a sequence of messages, events, or actions are given as an input



Introduction

- Stateful programs
  - require input messages to be sent in a certain expected order
  - e.g., protocol implementations

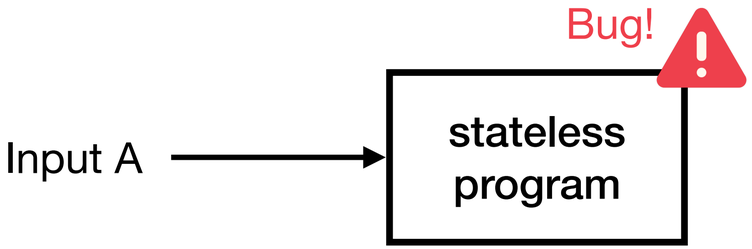
- Stateful bugs
  - triggered when a sequence of messages, events, or actions are given as an input

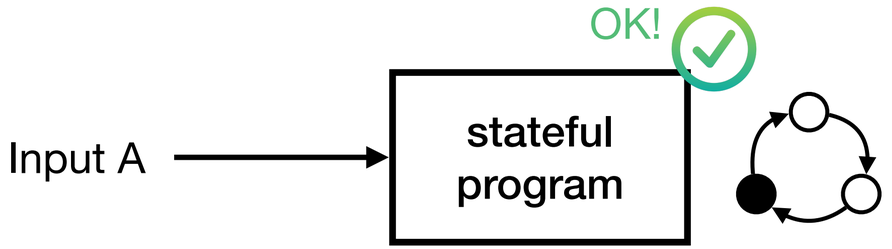


Introduction

- Stateful programs
  - require input messages to be sent in a certain expected order
  - e.g., protocol implementations

- Stateful bugs
  - triggered when a sequence of messages, events, or actions are given as an input

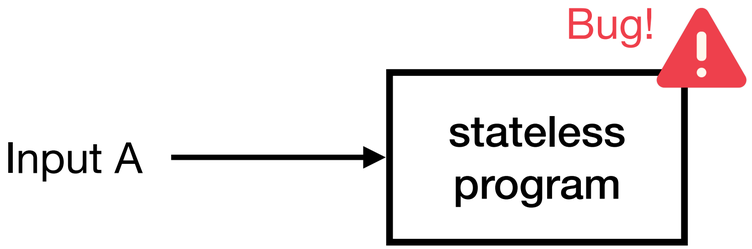


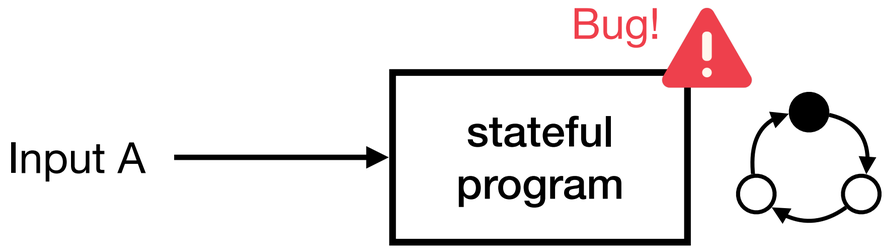


Introduction

- Stateful programs
  - require input messages to be sent in a certain expected order
  - e.g., protocol implementations

- Stateful bugs
  - triggered when a sequence of messages, events, or actions are given as an input





Introduction

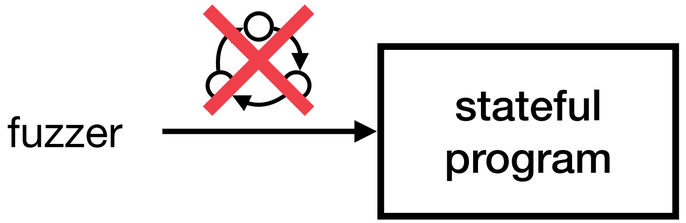
- Stateful programs
  - require input messages to be sent in a certain expected order
  - e.g., protocol implementations

- Stateful bugs
  - triggered when a sequence of messages, events, or actions are given as an input

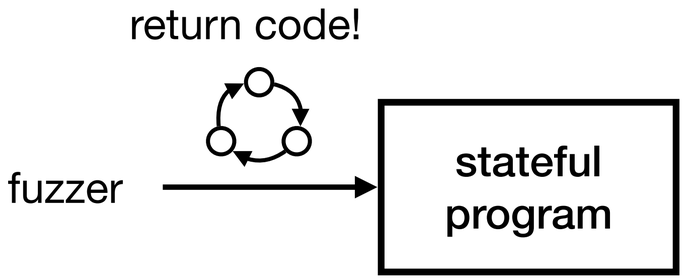
- How to efficiently find stateful bugs?
  - How to **cover the state space without a specification** of the required event sequences

Limitations of Related works

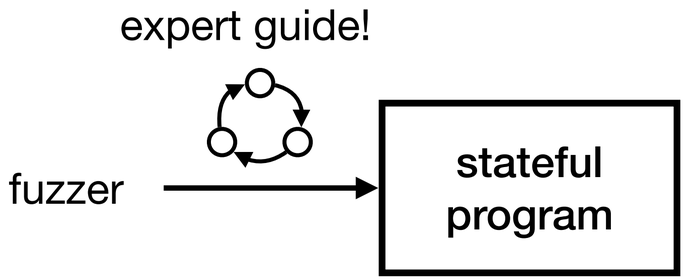
- AFL, Libfuzzer
  - stateless fuzzers
  - cannot generate a sequence of inputs



- AFLNet
  - exploit return code to infer states



- IJON
  - requires manual effort,
    i.e., knowledge of state specification



Automatic state identification

- Most of stateful softwares use named constants to represent internal states
  - e.g., enumeration type or #define macro

- H2O HTTP server

```
35  typedef enum enum_h2o_http2_stream_state_t {
36      /**
37       * stream in idle state (but registered; i.e. priority stream)
38       */
39      H2O_HTTP2_STREAM_STATE_IDLE,
40      /**
41       * receiving headers
42       */
43      H2O_HTTP2_STREAM_STATE_RECV_HEADERS,
44      /**
45       * receiving body (or trailers), waiting for the arrival of END_STREAM
46       */
47      H2O_HTTP2_STREAM_STATE_RECV_BODY,
48      /**
49       * received request but haven't been assigned a handler
50       */
51      H2O_HTTP2_STREAM_STATE_REQ_PENDING,
```

- openssl

```
1007  typedef enum {
1008      TLS_ST_BEFORE,
1009      TLS_ST_OK,
1010      DTLS_ST_CR_HELLO_VERIFY_REQUEST,
1011      TLS_ST_CR_SRVR_HELLO,
1012      TLS_ST_CR_CERT,
1013      TLS_ST_CR_COMP_CERT,
1014      TLS_ST_CR_CERT_STATUS,
1015      TLS_ST_CR_KEY_EXCH,
1016      TLS_ST_CR_CERT_REQ,
1017      TLS_ST_CR_SRVR_DONE,
1018      TLS_ST_CR_SESSION_TICKET,
1019      TLS_ST_CR_CHANGE,
1020      TLS_ST_CR_FINISHED,
1021      TLS_ST_CW_CLNT_HELLO,
```

Automatic state identification

- Most of stateful softwares use named constants to represent internal states
  - e.g., enumeration type or #define macro

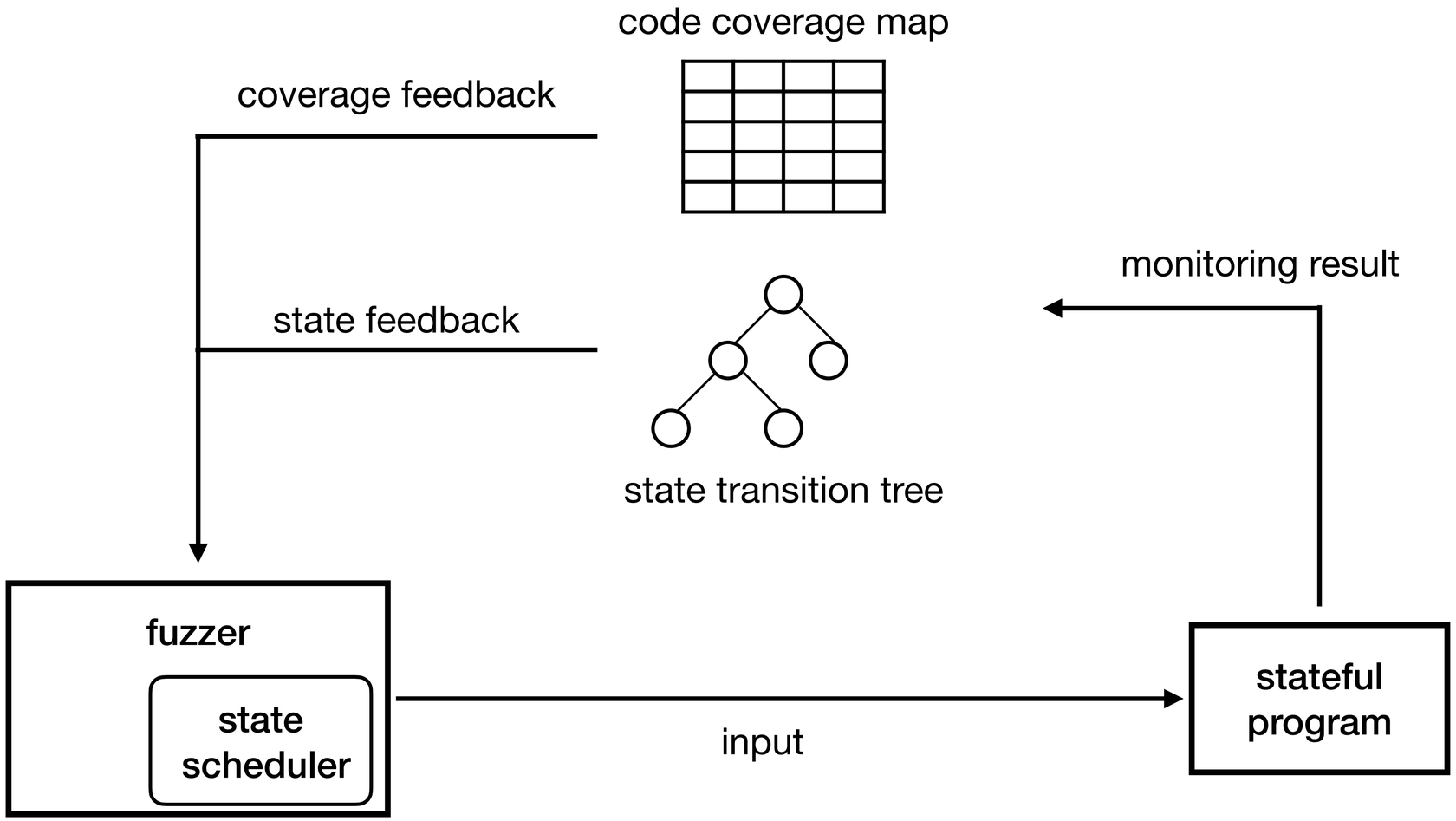
- State representation using named constants can be seen in Top-50 most widely used protocol implementations
  - e.g., FTP, SFTP, TLS, SMTP, HTTP2, RDP, NTP, IMAP, IRC, SMB, DAAP, SIP, DICOM, VNC, RTSP, MQTT
  - 44 use enumeration type and 6 use #define macro

Automatic state identification

- Idea
  - Approximate state variables
    (variables used to represent state in stateful programs)
    by the variables with named constants!

- False positives?
  - other variables such as configuration variables or error code variables sometimes take named constants
  - authors show that over 99% of extracted variables are true state variables!

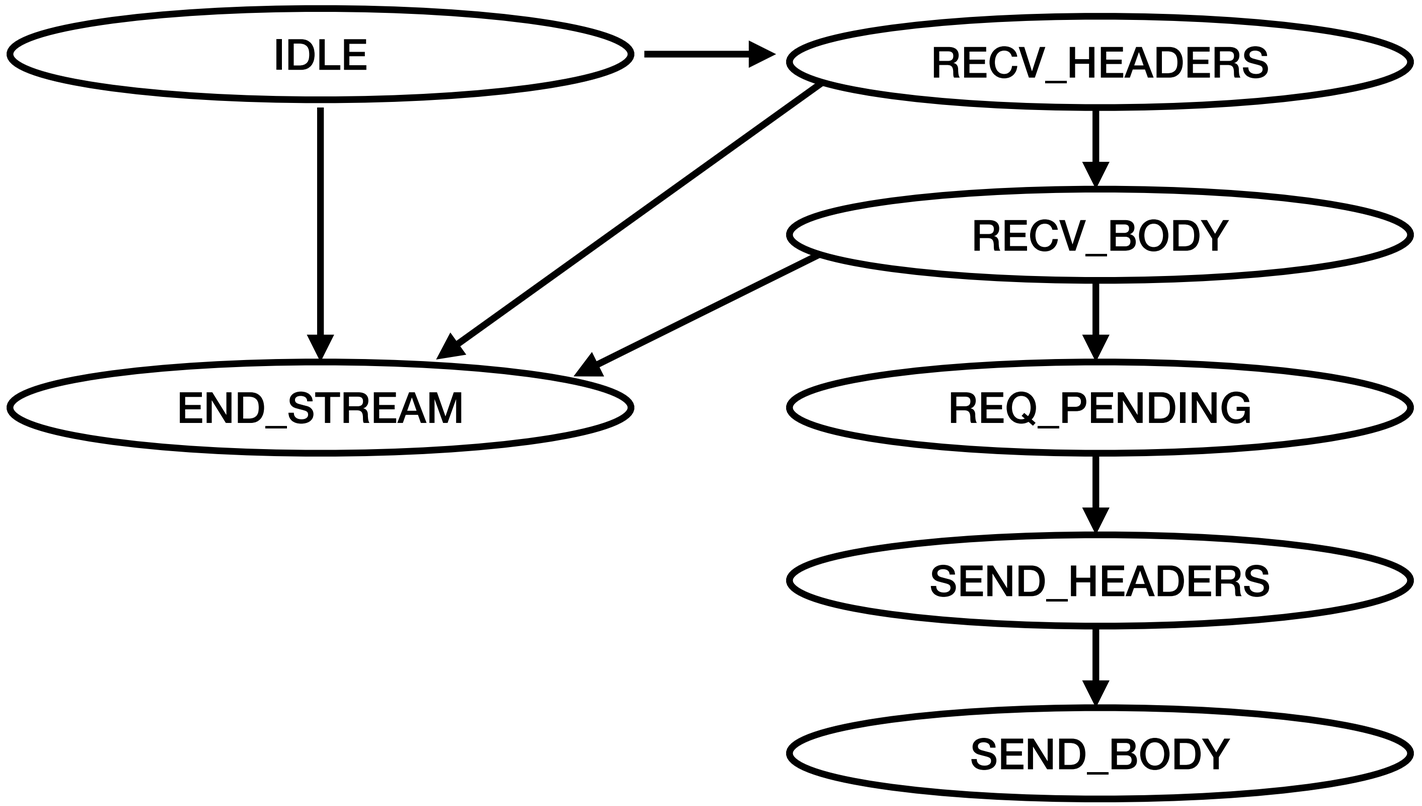
Stateful greybox fuzzer (SGFuzz)



State transition tree construction

- To construct state transition tree, SGFuzz monitors the changes of values of enumeration variables

  - monitor ‘stream-state’ variable in h2o



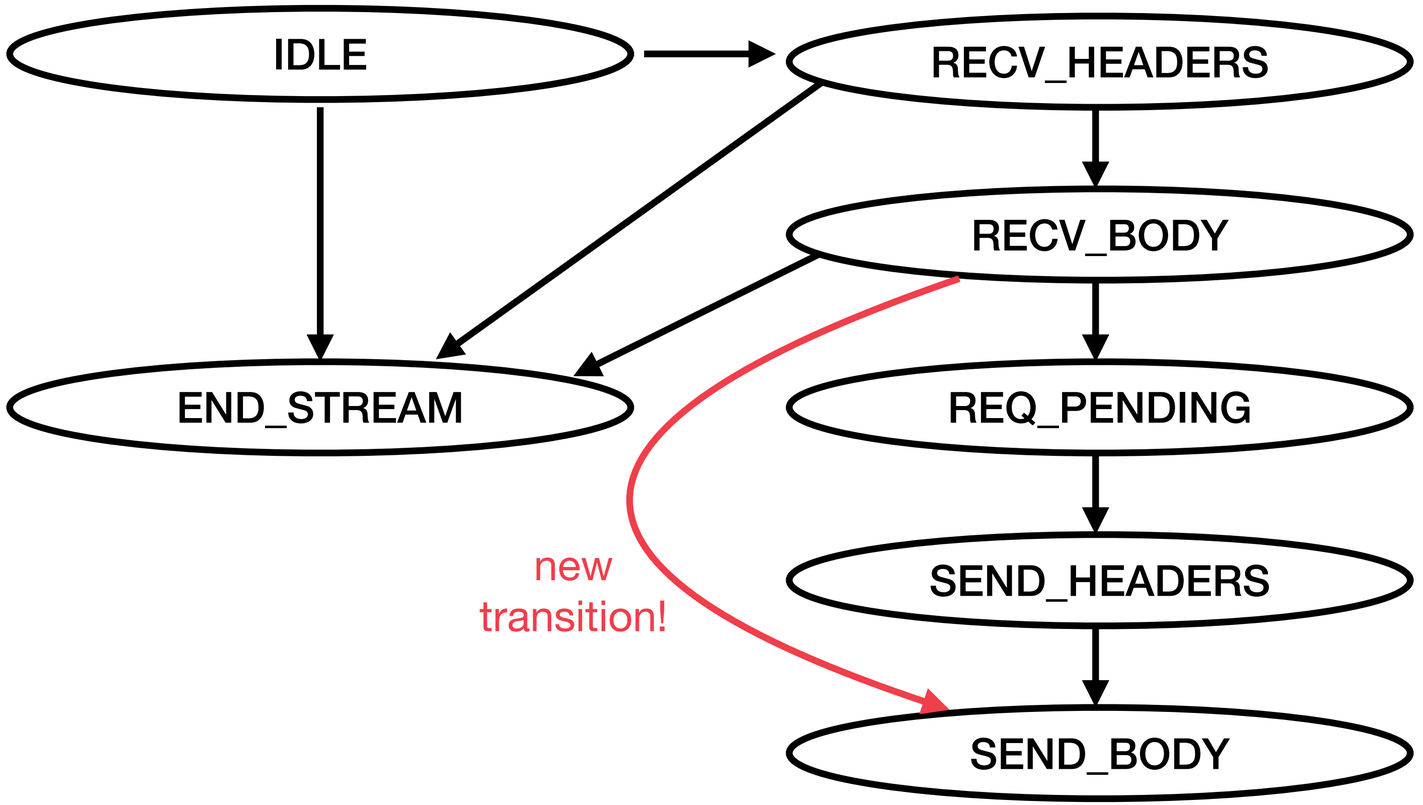
State fuzzing algorithm

- Procedure

  1. save the inputs that trigger new state transition

  2. assign more energy on the “core-logic” state sequences

  3. correlate input bytes and state transitions, giving more opportunities on mutating these bytes



State fuzzing algorithm

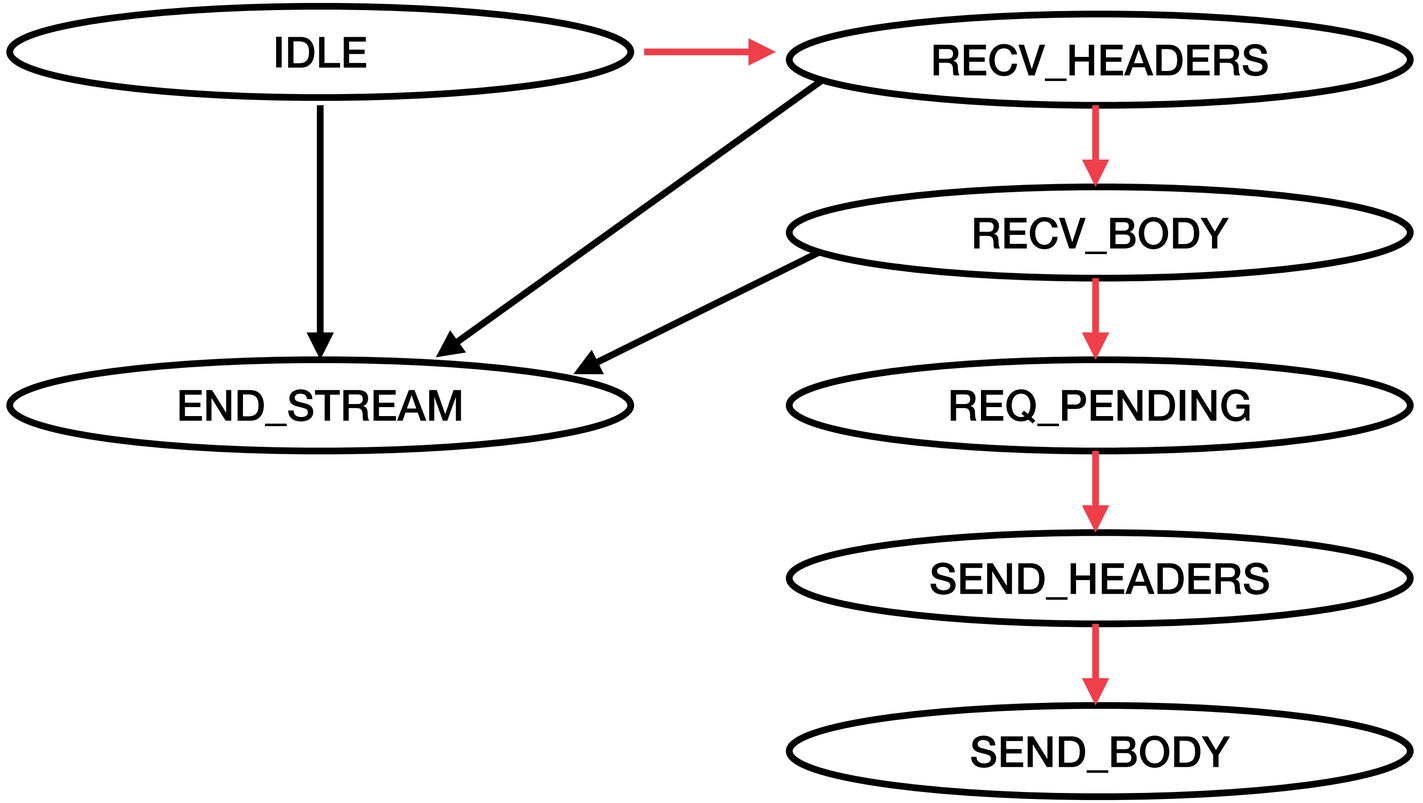
seeds that have triggered core logic transitions are more likely to be chosen

- Procedure

  1. save the inputs that trigger new state transition

  2. assign more energy on the "core-logic" state sequences

  3. correlate input bytes and state transitions, giving more opportunities on mutating these bytes



State fuzzing algorithm

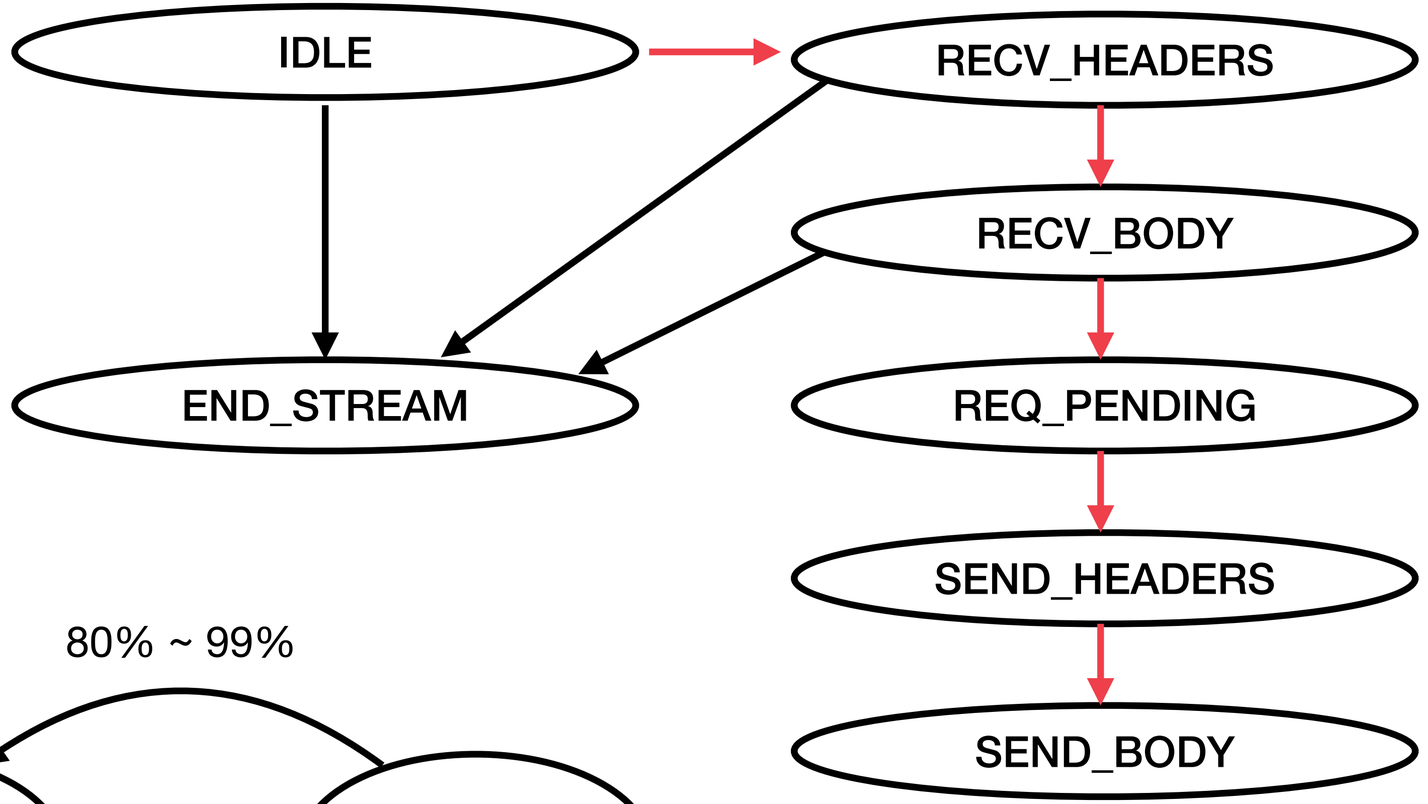
seeds that have triggered core logic transitions are more likely to be chosen

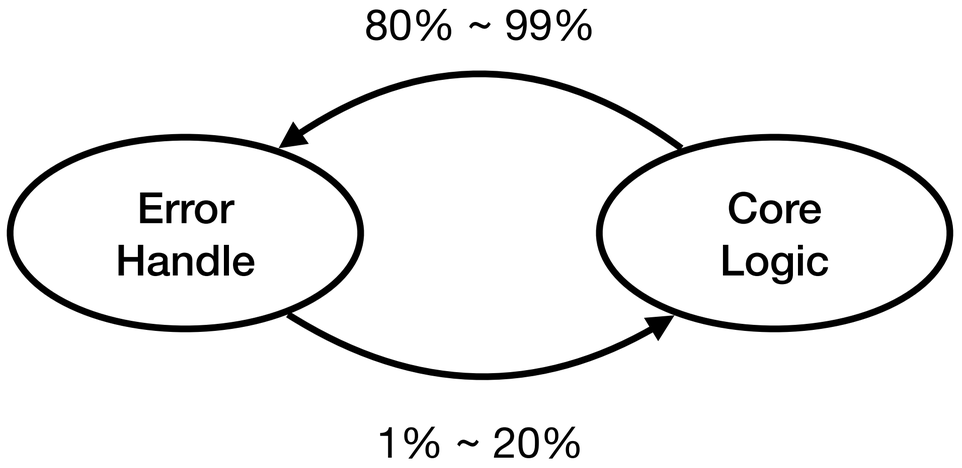
- Procedure

  1. save the inputs that trigger new state transition

  2. assign more energy on the "core-logic" state sequences

  3. correlate input bytes and state transitions, giving more opportunities on mutating these bytes





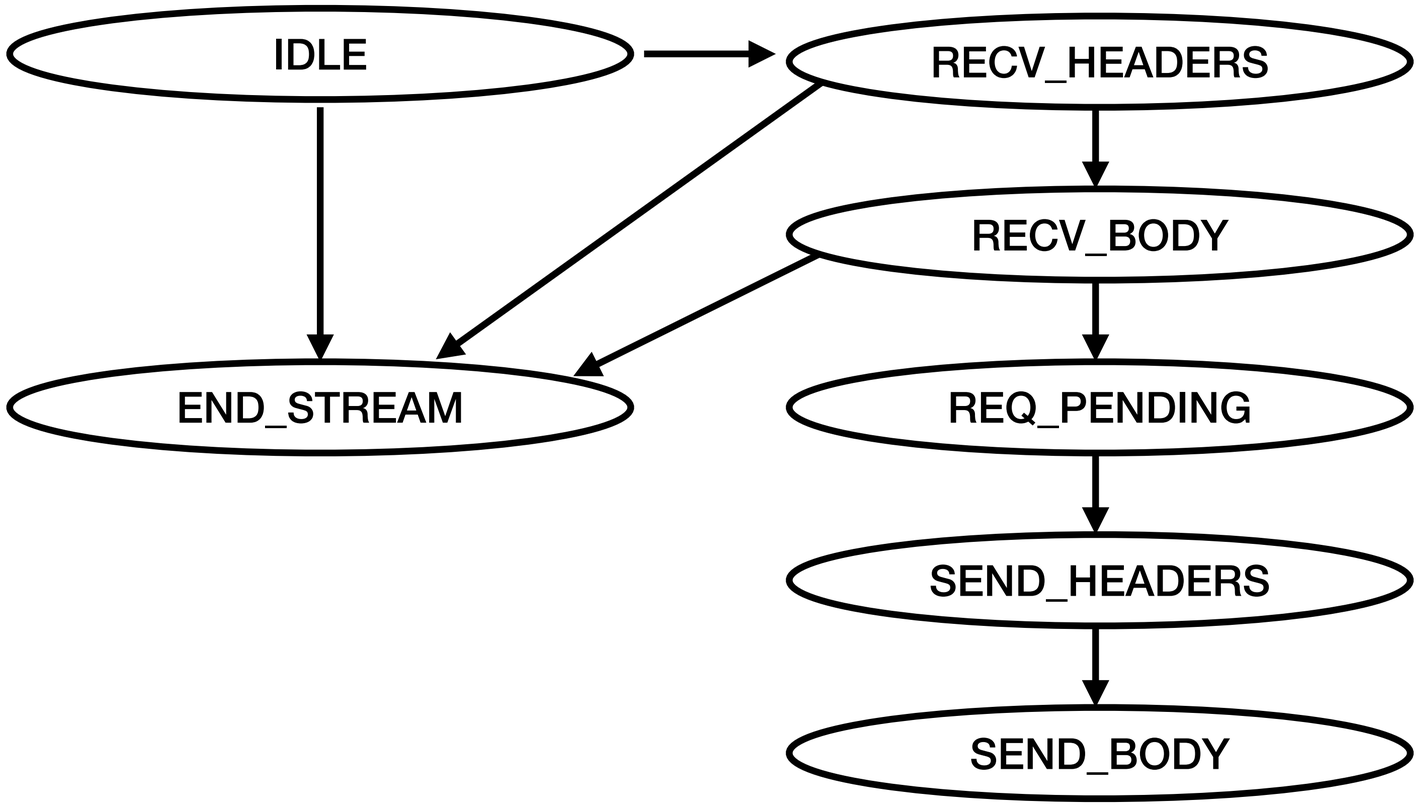
State fuzzing algorithm

- Procedure

  1. save the inputs that trigger new state transition

  2. assign more energy on the “core-logic” state sequences

  3. correlate input bytes and state transitions, giving more opportunities on mutating these bytes



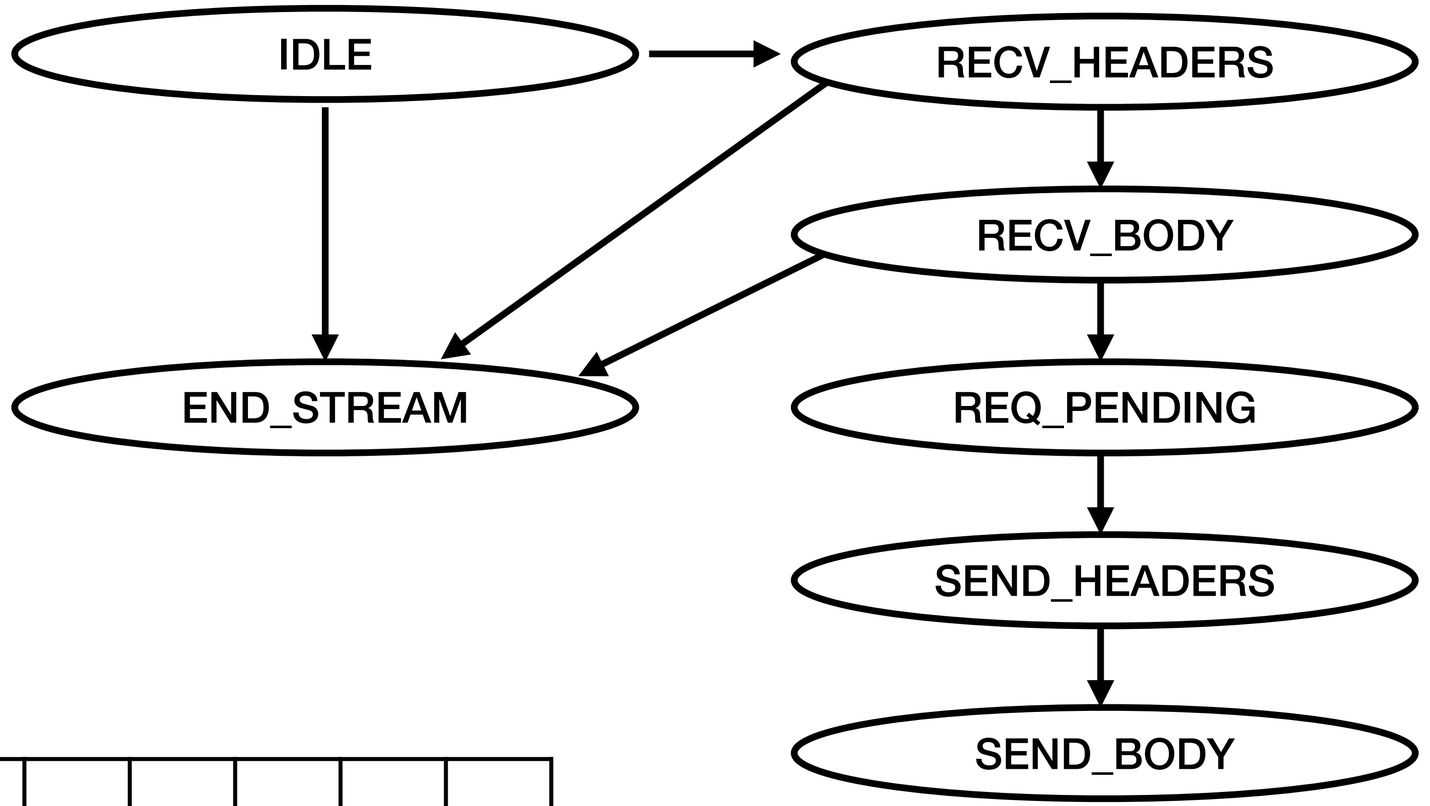
State fuzzing algorithm

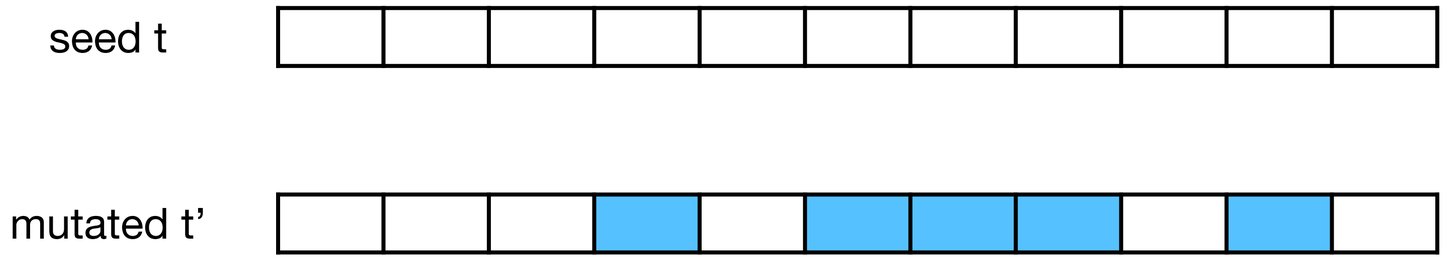
- Procedure

  1. save the inputs that trigger new state transition

  2. assign more energy on the “core-logic” state sequences

  3. correlate input bytes and state transitions, giving more opportunities on mutating these bytes





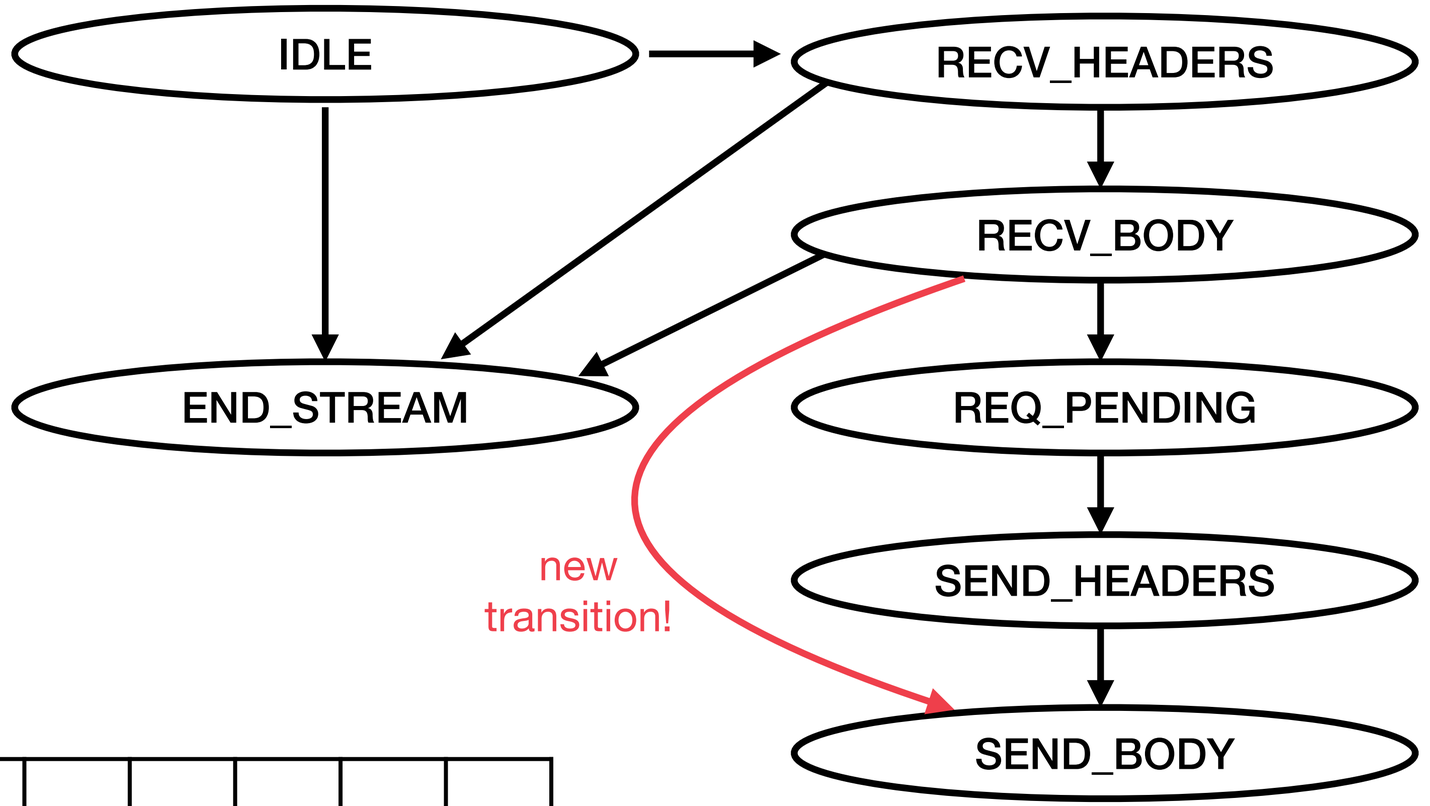
State fuzzing algorithm

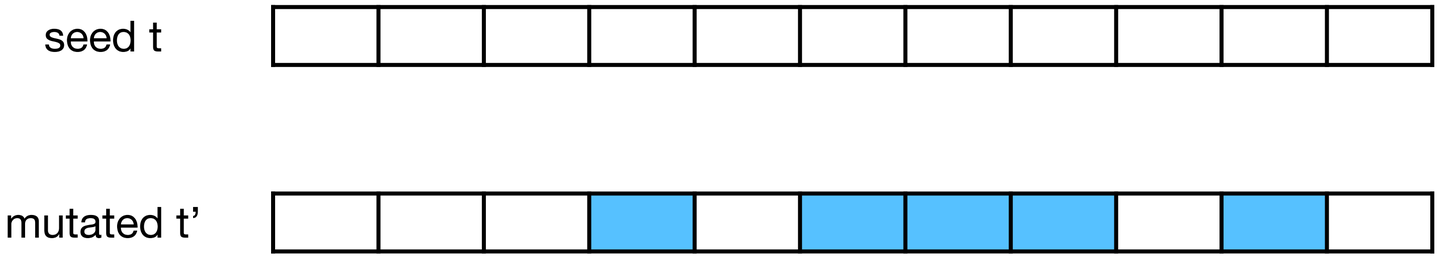
- Procedure

  1. save the inputs that trigger new state transition

  2. assign more energy on the “core-logic” state sequences

  3. correlate input bytes and state transitions, giving more opportunities on mutating these bytes





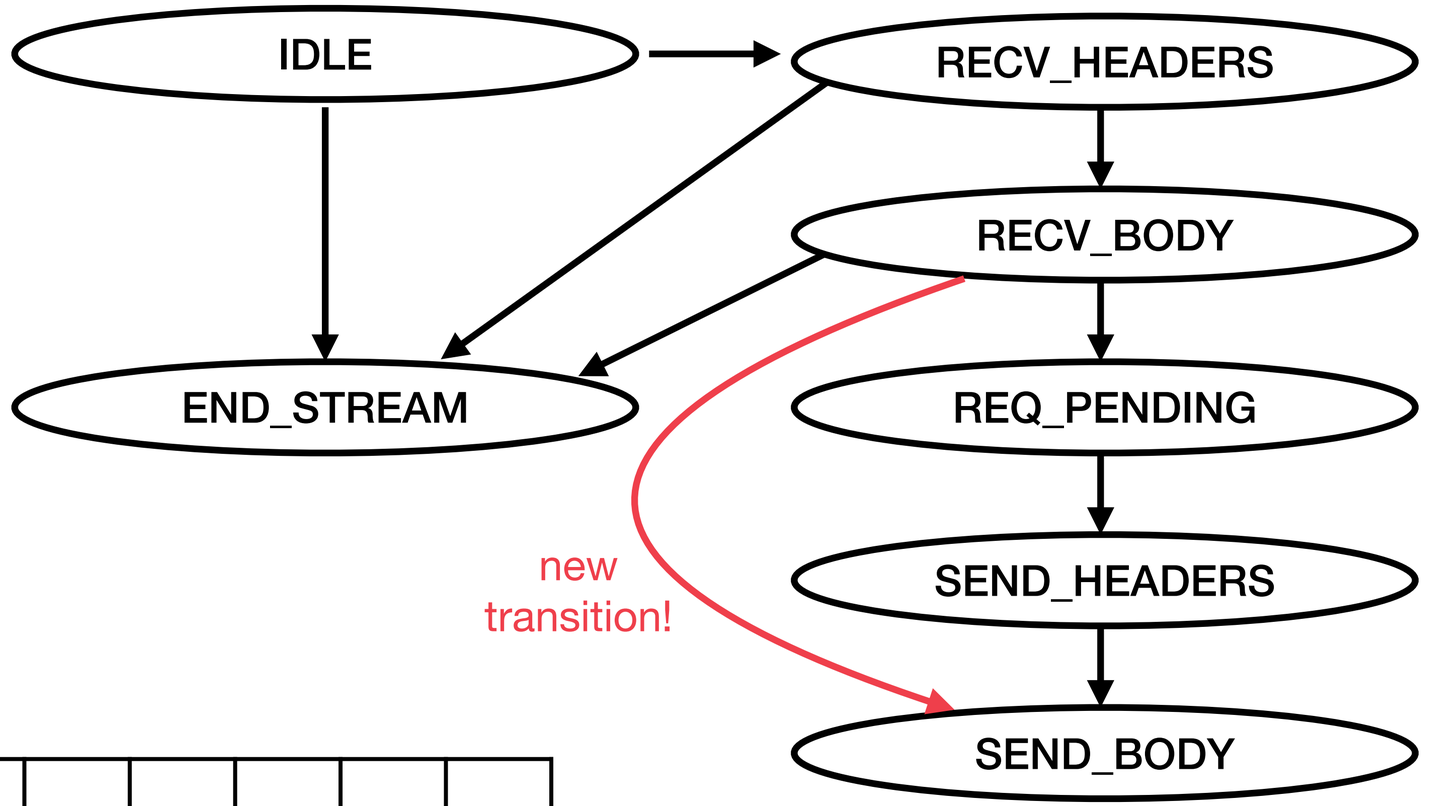
State fuzzing algorithm

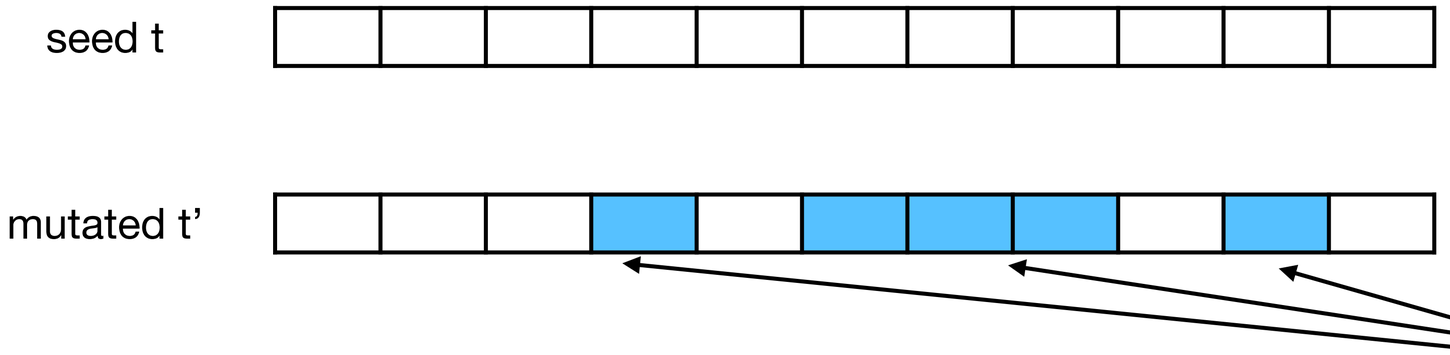
- Procedure

  1. save the inputs that trigger new state transition

  2. assign more energy on the “core-logic” state sequences

  3. correlate input bytes and state transitions, giving more opportunities on mutating these bytes





Evaluation setup

- Target programs
  - run 20 times for each target program
  - fuzz 23 hours for each run

Program	Protocol
H2O	HTTP
MbedTLS	SSL/TLS
OpenSSL	SSL/TLS
Curl	Several
Gstreamer	Custom
Live555	RTSP
Owntone	DAAP
DCMTK	DICOM

Evaluation: state transition coverage

- Measure the number of state transition sequences in the State Transition Tree

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

Avg: 33.9x

- On average, SGFuzz covers state transition sequences 30 times more than the baseline LibFuzzer.

Evaluation: state identification effectiveness

- 99.5% nodes are related to the true states.

Subject	State Transition Tree		
	All Nodes	State	Percentage
H2O	6418	6417	99.98%
MbedTLS	167	167	100.00%
Curl	35690	35629	99.83%
Gstreamer	11240	11224	99.86%
OpenSSL	817	789	96.57%
Live555	17446	17446	100.00%
OwnTone	3671	3671	100.00%
DCMTK	27178	27109	99.75%

Avg: 99.50%

Evaluation: new bugs

- Found 12 previously unknown bugs in 23 hours, and 10 of 12 are stateful bugs

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

Conclusion

- Present Stateful greybox fuzzer (SGFuzz)
  - automatically identify and monitor state of target program
- Show SGFuzz outperforms baseline fuzzers in terms of state transition coverage
  - covers 30 times more than the baseline LibFuzzer
- Show the effectiveness of state identification
  - on average, 99.5% nodes are related to the true states