Two ACCSS initiatives

1) ACCSS working group on Software Security

   Synergy with INTERSCT, C-SIDE and VERSEN

   Contact Olga Katyatskaya

2) ACCSS PhD group as part of CSng

   Synergy with research schools IPA,SIKS, ASCI?

   Contact Cristian Daniele

Fuzzing important as

- quality assurance technique in WP2 Design

  Here Design = Design + rest of SDLC

- bug hunting technique in WP4 Attacks

Fuzzing Stateful Systems

Seyed Andarzian, Cristian Daniele, Erik Poll

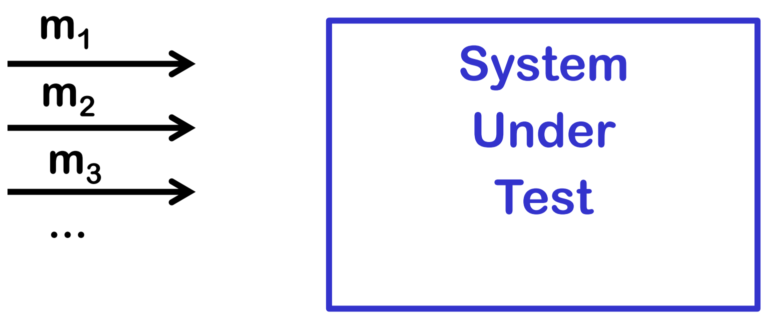
Digital Security

Radboud University Nijmegen

Fuzzing stateless vs stateful systems

Stateless SUT

- Eg pdfviewer, graphics library
- Looking for parsing bugs



m_1

m_2

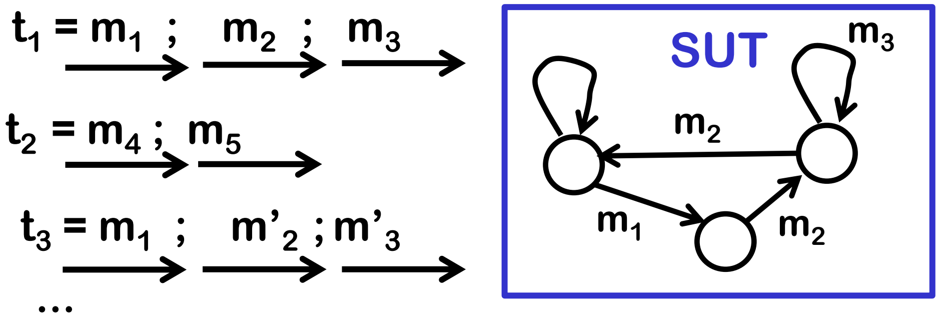
m_3

…

System Under Test

Stateful SUT

- Eg TCP, SSH, WhatsApp
- Two aspects that can be fuzzed:
  1) the messages
  2) the order of messages



$t_1 = m_1 \; ; \; m_2 \; ; \; m_3$

$t_2 = m_4 \; ; \; m_5$

$t_3 = m_1 \; ; \; m'_2 \; ; m'_3$

…

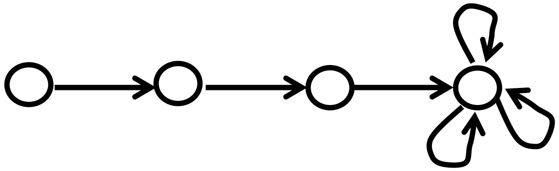
SUT

- Looking for a) parsing bugs and b) program logic bugs

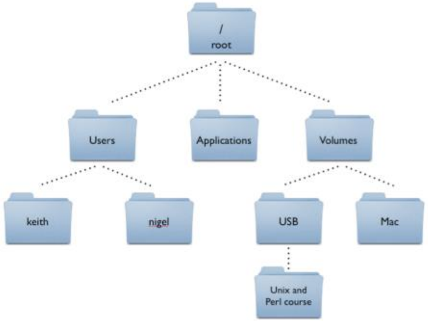
With fuzzing we normally look for crashes & hangs. For stateful SUTs deviations in state behaviour may be interesting bugs, too

Different kinds/origins of state behaviour

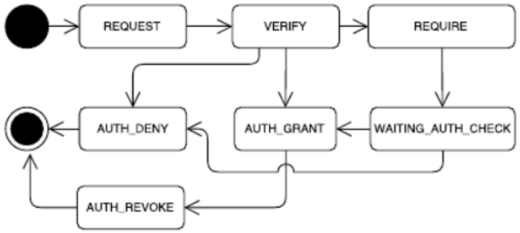
- an initialisation phase



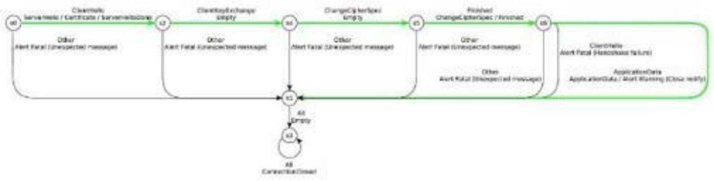
- application menu or directory structure



- application dialogue or protocol
  - incl. protocol for access control



  - incl. crypto protocols, eg TLS

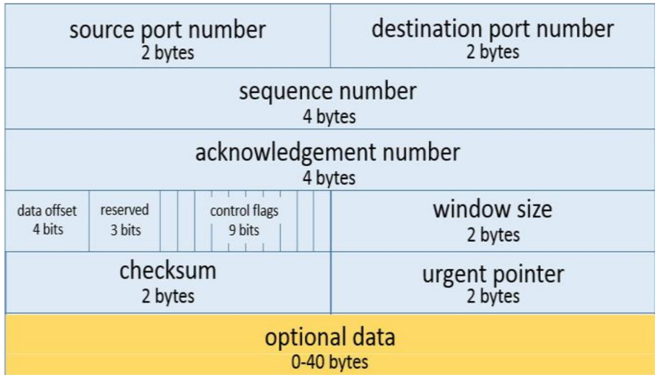


These categories overlap and can be combined

Security-by-Design: LangSec

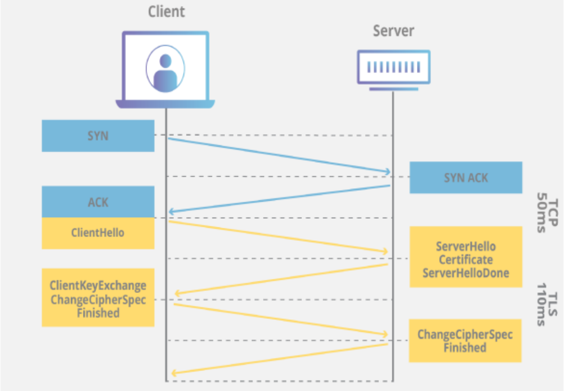
Prevention of input handling bugs by LangSec (language-theoretic security)

1. Provide *clear*, *unambiguous*, *formal* spec of *simple* input protocol



source port number 2 bytes			destination port number 2 bytes
sequence number 4 bytes			
acknowledgement number 4 bytes			
data offset 4 bits	reserved 3 bits	control flags 9 bits	window size 2 bytes
checksum 2 bytes			urgent pointer 2 bytes
optional data 0-40 bytes			

Message format



Protocol state machine

2. Generate code

Eg using Verum Dezyne for protocol state machine.

People don't do this, which is why fuzzing is such a great success

More info: LangSec.org or DARPA SafeDocs

Fuzzers for stateful systems

- Not that many stateful fuzzers around

  compared to stateless, see https://fuzzing-survey.org

  but wide variety in (combination of) approaches

- State space is obviously complicating factor

  (Bigger) combinatorial explosion:
  not just strange messages, but also strange sequences of messages

  Associated coverage criterion: state machine coverage

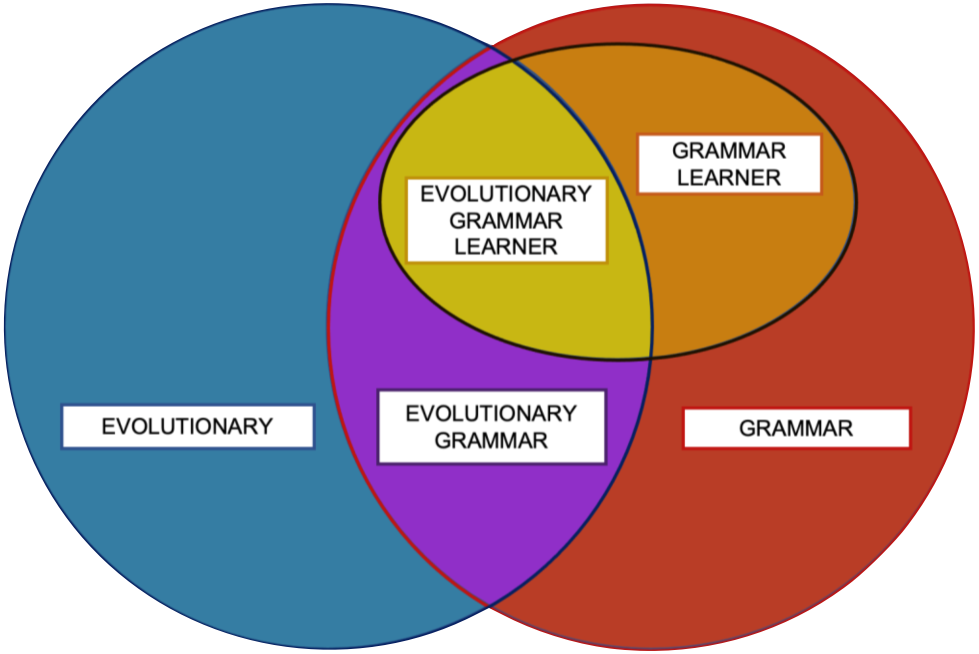
- Very slow (a few tests/sec, not thousands tests/sec) due to

  1. overhead of network stack
  2. having to repeat initial prefix to reach 'interesting' state

Survey “Fuzzers for Stateful Systems” [arXiv:2301.02490, 2023]

7 categories of stateful fuzzers

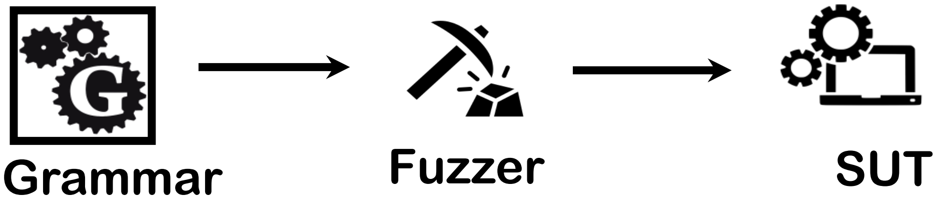
- Grammar-Based fuzzers
- Evolutionary fuzzers
- Evolutionary Grammar-Based Fuzzers
- Grammar Learner Fuzzers
- Evolutionary Grammar Learner Fuzzers
- Machine Learning Based Fuzzers
- Man-in-the-middle Based Fuzzers



Grammar-based & grammar learner fuzzers

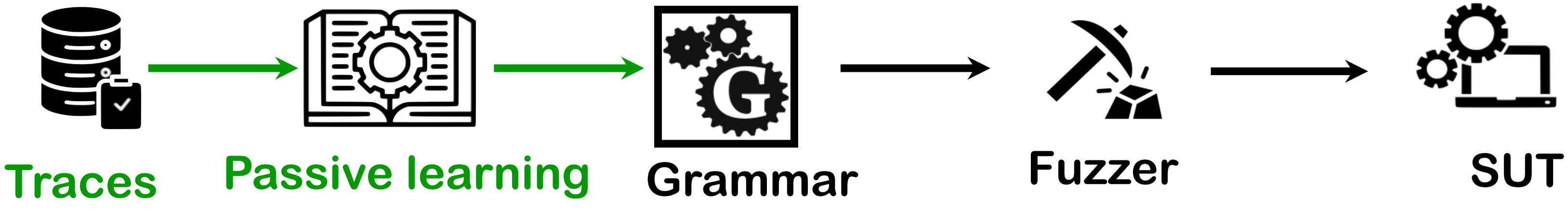
- **Grammar-based**

  user provides grammar for state machine & message format



- **Grammar Learner**

  grammar inferred from traces, eg using passive learning

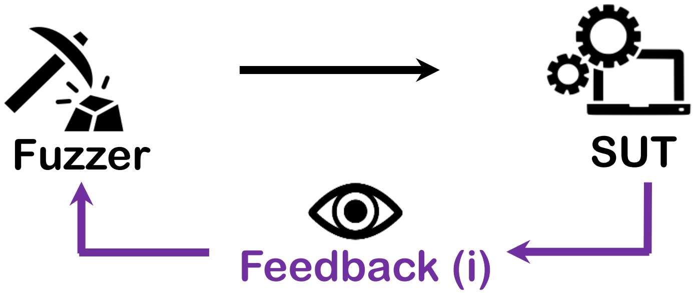


Evolution (i)

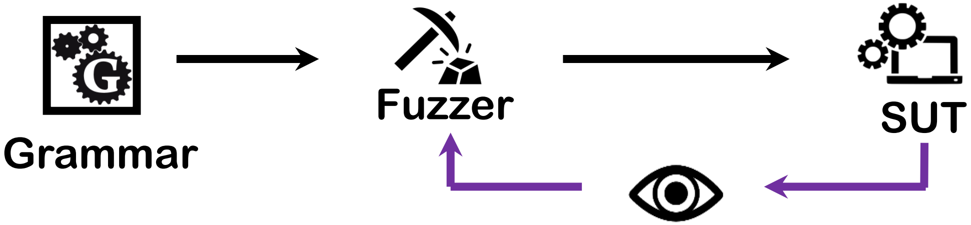
- Evolutionary: mutation of inputs (messages and/or sequences) guided by feedback from SUT

  a) observing branch coverage like afl (nyx-net, SPNS fuzzer)

  b) observing program variables:
  manually annotated (IJON) of automatically inferred (SGFuzz)

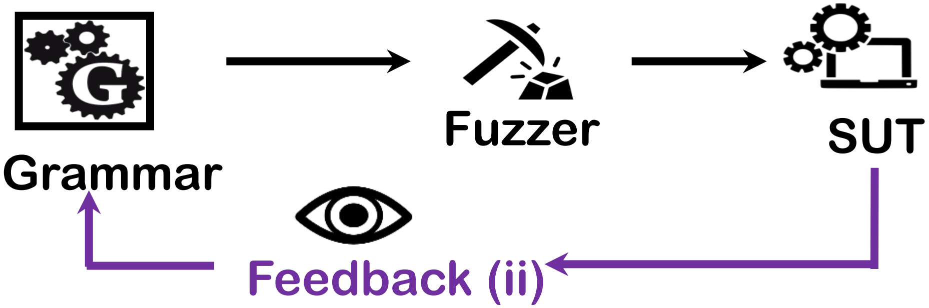


- Can be combined with grammar-based:
  evolutionary grammar-based (RESTler, SPFuzz, EPF)

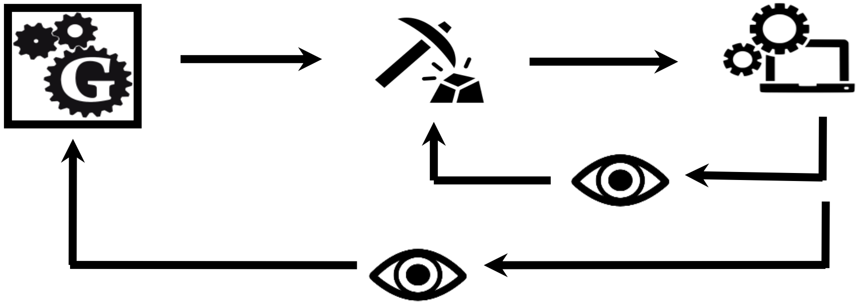


Evolution (ii)

- We can also use feedback to infer/improve the grammar, esp. the state machine: evolutionary grammar learner
  - e.g. system response as feedback (LearnLib/L* aka active learning)



- This can be combined with feedback (i) to mutate messages (aflnet)



Fuzzers for Stateful Systems [arXiv:2301.02490, 2023]

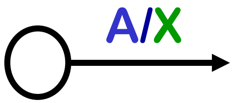
	Feedback I	Feedback II	Requires	Based on/uses
GRAMMAR-BASED			Grammar	
Peach, SNOOZE, Sulley, PROTOS, AspFuzz, ...			"	
BooFuzz			"	Sulley
Fuzzowksi			"	BooFuzz
GRAMMAR LEARNER			Traces & ...	
Hsu et al.			message grammar	Passive learning
Pulsar				Passive learning
Glade				Active learning
EVOLUTIONARY			Traces & ...	
nyx-net	Coverage		protocol spec	AFL
FitM fuzzer	Coverage		client *and* server binary	AFL
SNPS fuzzer	Coverage			AFL
Chen et al.	Coverage & Branches		source code	AFL, manual code annotation
IJon	Coverage & Variables		source code	AFL, manual code annotation
SGFuzz	Coverage & Variables		source code	AFL, automatic code annotation
EVOLUTIONARY GRAMMAR-BASED			Grammar	
RESTler	Response		"	
SPFuzz	Coverage		"	AFL
EPF	Coverage		"	AFL & Fuzzowski
EVOLUTIONARY GRAMMAR LEARNER				
AFLnet	Coverage	Response	Traces	AFL
FFUZZ	Coverage	Response	Traces	AFL, AFLNet
StateAFL	Coverage	Memory	Traces	AFL
SGPFuzzer	Coverage	Memory	Traces	AFL
LearnLib		Response	Set of messages	L*
Doupé et al.		Response	None	Web application crawling
ML-BASED			Traces	
GANFuzz			"	seq2seq
Fuzzing of Network Protocols			"	seq2seq
SeqFuzzer			"	seq-gan
MAN-IN-THE-MIDDLE			Live traffic	
AutoFuzz			"	Passive learning
Live Protocol Fuzzing			"	
SECFUZZ			"	

Active Learning
aka
State Machine Learning

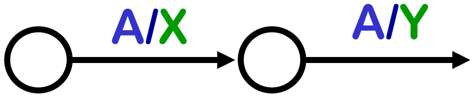
Active Learning aka State Machine Inference

Just try out many sequences of inputs, and observe outputs

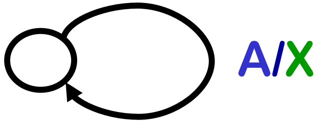
Eg. suppose input A results in output X



- If second input A results in *different* output Y

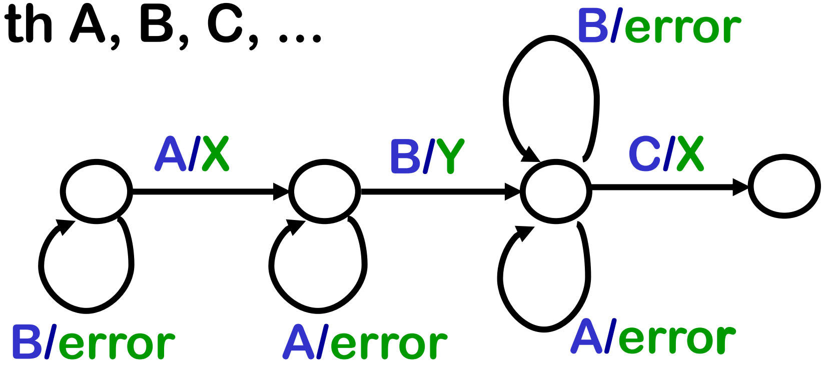


- If second input A results in the *same* output X



Now try more sequences of inputs with A, B, C, ...

to e.g. infer



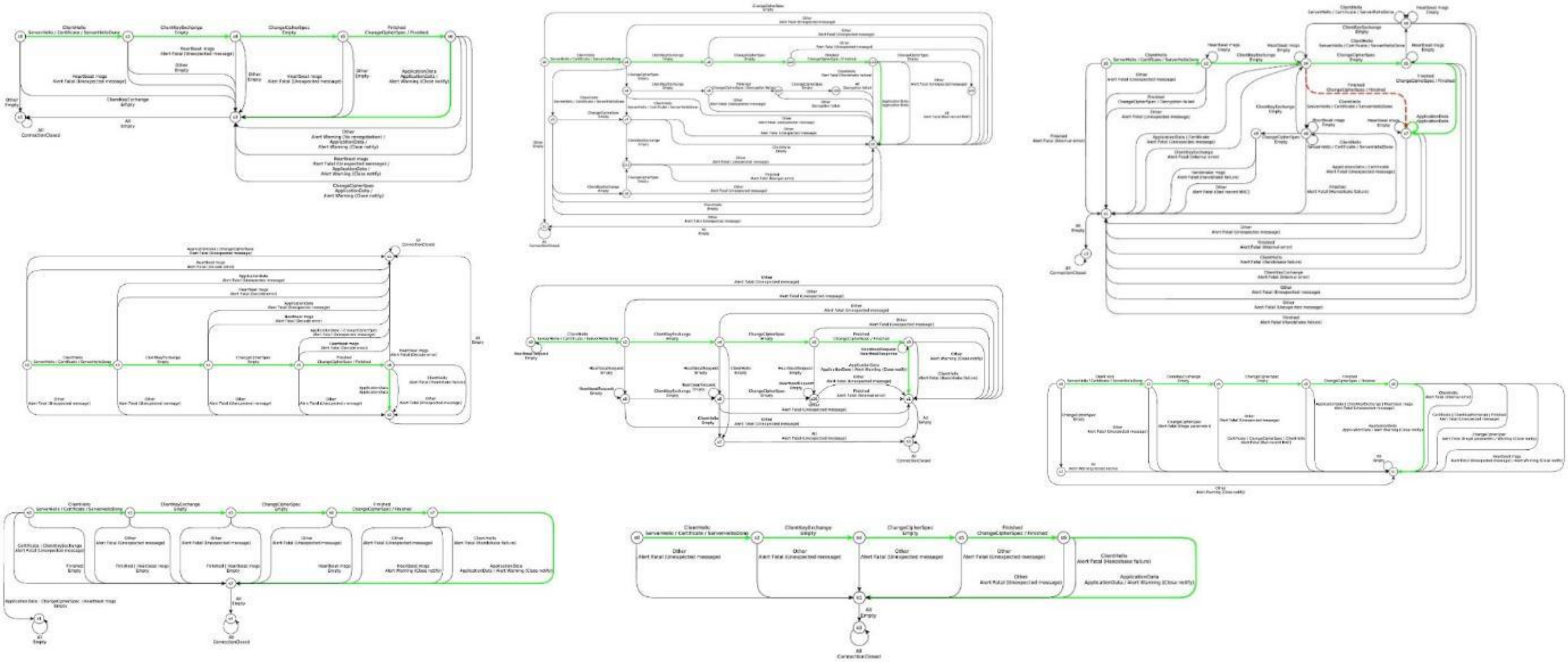
The inferred state machine is an under-approximation of real system

First algorithm for this, L* [Angluin 1987], implemented in LearnLib

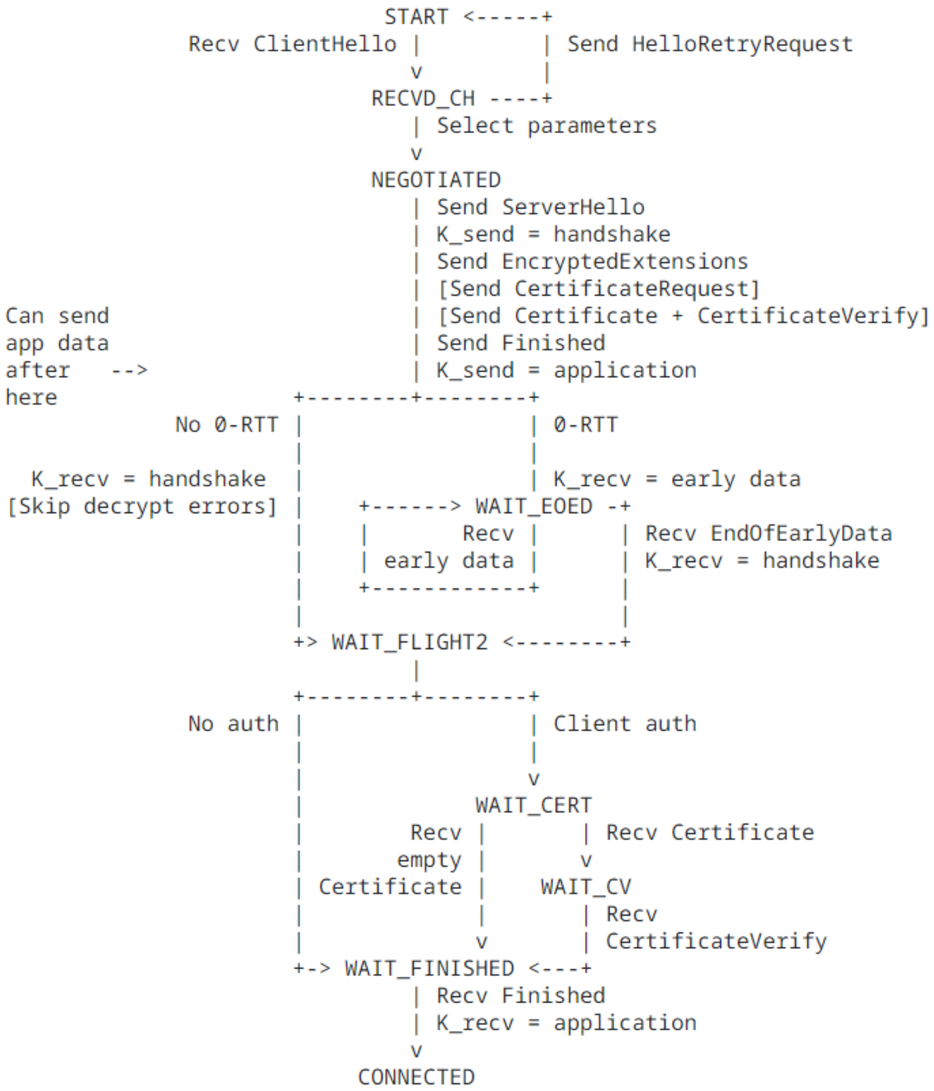
Active Learning (using L* implemented in LearnLib)

- Active learning is limited form of stateful fuzzing:
  we only fuzz the ***message order***, not the messages
- Used on many case studies to reveal surprising differences, incl. some security flaws
  - eg TCP, SSH, TLS, EMV bankcards, ABN-AMRO e.dentifier, DTLS, QUIC, IEC 60870-5-104, MQTT

Different TLS implementations



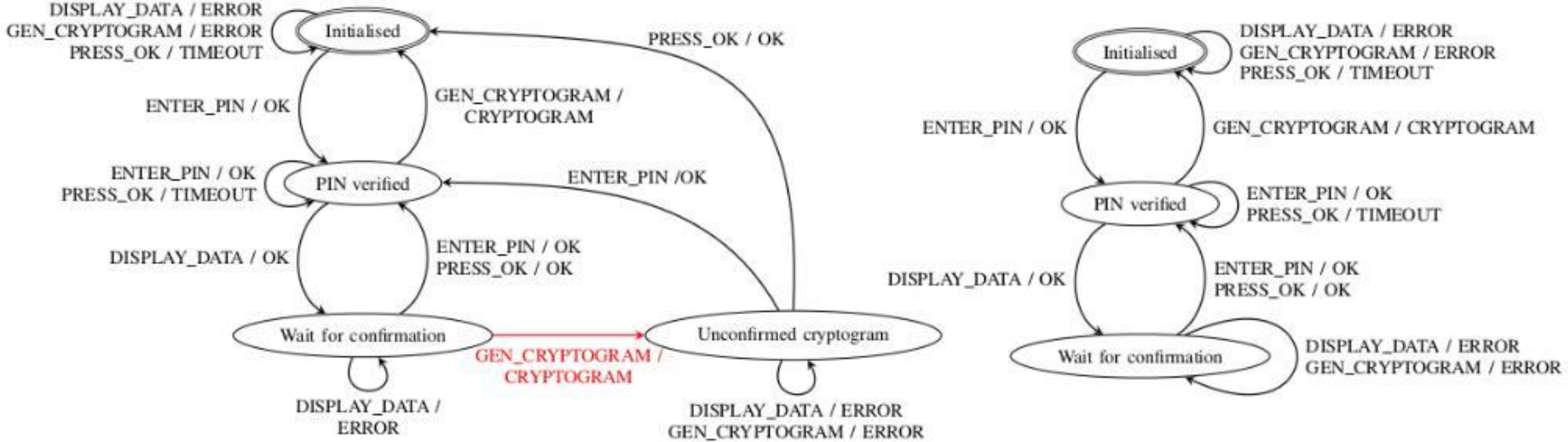
TLS 1.3 [RFC 8446, 2018]

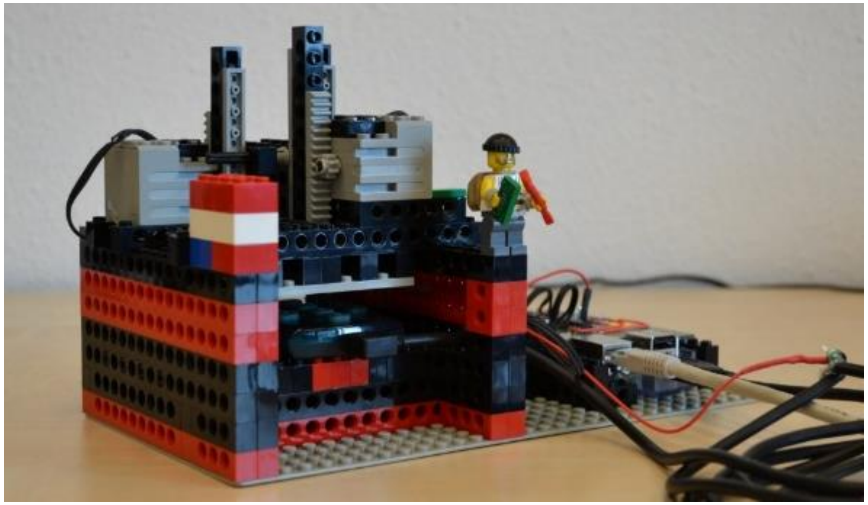


```
                             START <-----+
              Recv ClientHello |         | Send HelloRetryRequest
                               v         |
                            RECVD_CH ----+
                               | Select parameters
                               v
                            NEGOTIATED
                               | Send ServerHello
                               | K_send = handshake
                               | Send EncryptedExtensions
                               | [Send CertificateRequest]
Can send                       | [Send Certificate + CertificateVerify]
app data                       | Send Finished
after   -->                    | K_send = application
here                  +--------+--------+
             No 0-RTT |                 | 0-RTT
                      |                 |
  K_recv = handshake  |                 | K_recv = early data
[Skip decrypt errors] |    +------> WAIT_EOED -+
                      |    |       Recv |      | Recv EndOfEarlyData
                      |    | early data |      | K_recv = handshake
                      |    +------------+      |
                      |                        |
                      +> WAIT_FLIGHT2 <--------+
                               |
                      +--------+--------+
              No auth |                 | Client auth
                      |                 |
                      |                 v
                      |             WAIT_CERT
                      |        Recv |       | Recv Certificate
                      |       empty |       v
                      | Certificate |    WAIT_CV
                      |             |       | Recv
                      |             v       | CertificateVerify
                      +-> WAIT_FINISHED <---+
                               | Recv Finished
                               | K_recv = application
                               v
                           CONNECTED
```

State machine learning for e.dentifier2

State machines inferred for flawed & patched device

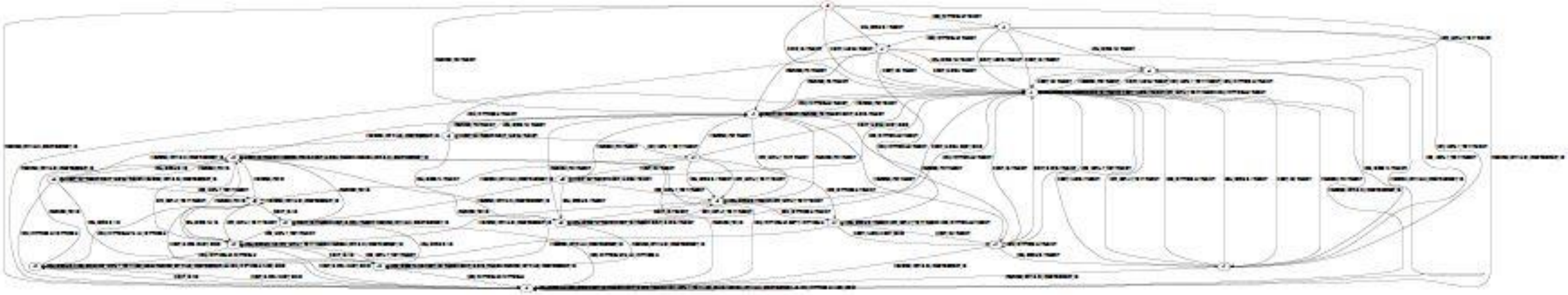




[Georg Chalupar et al.,
engineering using Lego,

Movie at http://tinyurl/legolearn

scary state machine COMPLEXITY

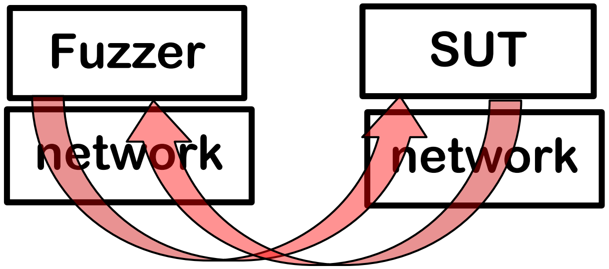


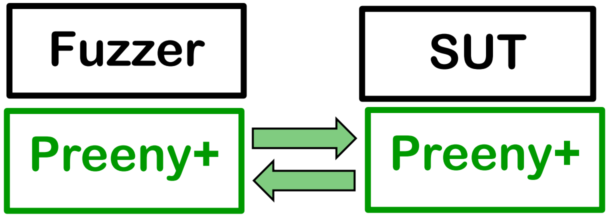
SPEED

Green Fuzzer [work by Seyed Andarzian]

Improving the speed of stateful fuzzing by

1. reducing overhead of network stack,
   by replacing network stack with simulated network stack

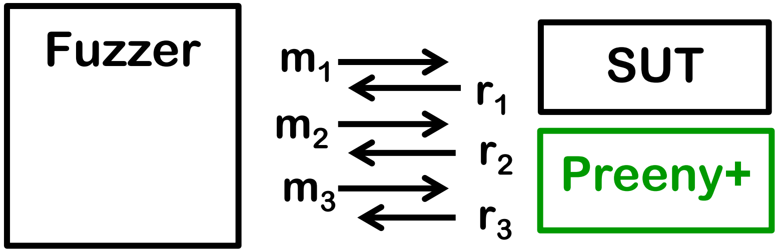


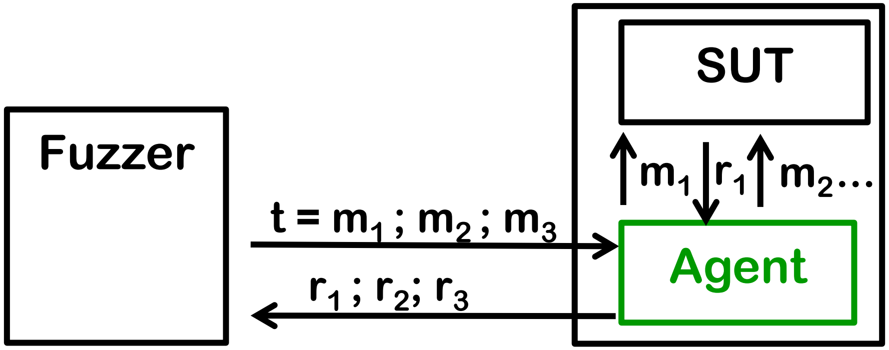


Green Fuzzer [work by Seyed Andarzian]

Improving the speed of stateful fuzzing by

1. reducing overhead of network stack,
   by replacing network stack with simulated network stack
2. reducing the overhead of context switching between SUT & fuzzer:
   instead of sending one message at the time, send whole trace





Green Fuzzer [work by Seyed Andarzian]

Improving the speed of stateful fuzzing by

1. reducing overhead of network stack,
   by replacing network stack with simulated network stack
2. reducing the overhead of context switching between SUT & fuzzer:
   instead of sending one message at the time, send whole trace

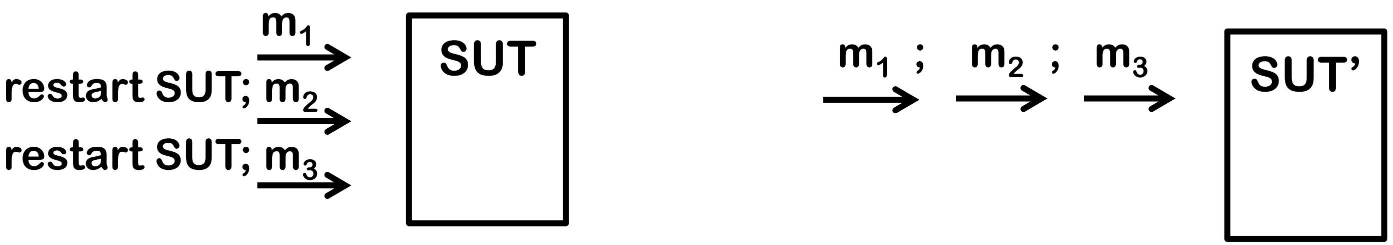
Performance results, messages/sec, on ProFuzzBench case studies

	AFLnet	Desock+	speed-up	Agent	speed-up
lightFTP	12	49	300%	64	30%
dnsmasq	15	19	30%	19	0%
live555	14	29	100%	31	10%
dcmqrscp	17	21	20%	25	20%
tinydtls	82	19	60%	34	80%

Afl* [work in progress by Cristian Daniele]

- Afl has fast persistent mode to speed up fuzzing

  Basic idea: modify SUT so that it can be fed multiple inputs in a row, without restarting (or forking)



- This (obviously!) can be used for fuzzing stateful systems too
  - If one of the messages effectively resets the SUT, then we never have to restart it; otherwise we still do

Afl* [work in progress by Cristian Daniele]

Performance results for LightFTP

	Speed	Time to find bug 1	Time to find bug 2
AlfNet	9 messages/sec	> 24 hr	>24 hr
Afl*	34000 messages/sec	1m 50s	15m 27s

Very fast but not very deep; reaching & fuzzing deeper states will require guidance by smarter strategies.

Open question:

is afl-style branch coverage a good way to observe state coverage?

Conclusion and open problems

- Other/better combinations?
- More cases studies: OPC-UA, 5G
- Benchmarking?

  Comparing stateful fuzzers is hard; big variety in SUT state machines.

  ProFuzzBench only compares speed
- Fuzzer-friendliness?

  Implementations can (should?) be made more fuzzer-friendly, e.g.

  - options to turn off cryptographic checks
  - identification of central loop for persistent fuzzing
  - for stateful systems: adding a reset operation for testing?