The Network and Distributed System Security Symposium (NDSS) 2024

Large Language Model guided Protocol Fuzzing

Ruijie Meng

ruijie@comp.nus.edu.sg

Co-authors: Martin Mirchev, Marcel Böhme, Abhik Roychoudhury

NUS National University of Singapore

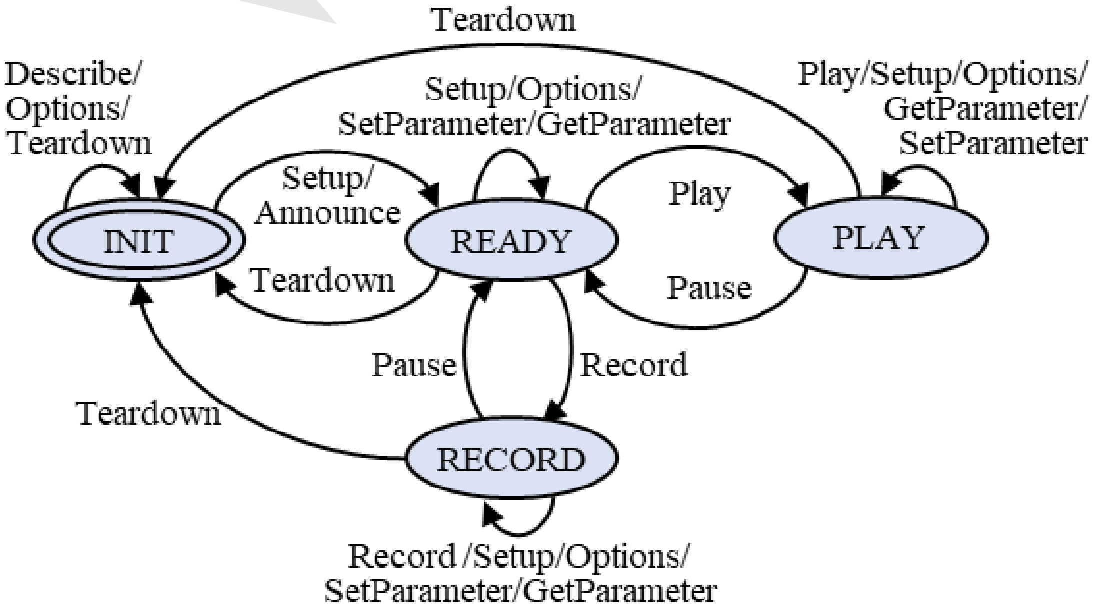
MAX PLANCK INSTITUTE FOR SECURITY AND PRIVACY

MONASH University

Testing Protocol Implementations

- **Protocol implementations are stateful reactive systems**
  - To expose a vulnerability, send the right messages in the right order
  - Message structures and orders are often specified in RFCs

RTSP State Machine



Play Message Structure

```
PLAY rtsp://127.0.0.1:8554/aacAudioTest/ RTSP/1.0\r\n
CSeq: 4\r\n
User-Agent: ./testRTSPClient (LIVE555 Streaming Media v2018.08.28)\r\n
Session: 000022B8\r\n
Range: npt=0.000-\r\n
\r\n
```

Challenges in Protocol Fuzzing

Generator-based Fuzzing:
Generate random message sequences from scratch based on the machine-readable information about the protocol

- Much manual effort involved
  - Some specification missed
  - Tedious and error-prone

Mutation-based Fuzzing (*more widely-used*):
Use a set of pre-recorded message sequences as seed inputs for mutation

Several Challenges:
(C1) Dependence on initial seeds
(C2) Unknown message structure
(C3) Unknown state space

We try to leverage LLMs to resolve these challenges!!

Linkage to Large Language Models

The capabilities of LLMs have various implications for protocol fuzzing:

- ➢ Network protocols are implemented in accordance with RFCs
  - RFCs are written in natural language and often public available, so LLMs should be able to understand RFCs

- ➢ Messages are in text format transmitted between servers and clients
  - LLMs have strong text-generation capabilities
- ➢ Fuzzing is highly automatic and easy-to-use
  - Integrating LLMs into fuzzing can still keep these features

Do LLMs really have the capabilities to resolve challenges in protocol fuzzing?

Case Study

Study the RTSP protocol with Live555

(C1) Enriching Seed Corpus:

About 80% messages generated are correct

(C2) Lifting Message Grammars:

All message grammars are identical to the ground truth

(C3) Inducing Interesting State Transitions:

Of the LLM-generated client requests, 69% to 89% induced a transition to a different state, covering all state transitions for each individual state

LLM-guided Protocol Fuzzing

(C1) Dependence on initial seeds:

- Enriching Initial Seeds

(C3) Unknown state space:

- Inferring state space and surpassing Coverage Plateau

```
Input  :Program P_f, protocol p, initial seed corpus C
Output :Crashing seeds C_x
Grammar G ← ChatGrammar (p)
C ← C ∪ EnrichCorpus (C, p)
PlateauLen ← 0
StateMachine S ← ∅
repeat
    State s ← ChooseState (S)
    Messages M, response R ← ChooseSequence (C, s)
    ⟨M1, M2, M3⟩ ← M
    for i from 1 to AssignEnergy (M) do
        if PlateauLen < MaxPlateau then
            if UniformRandom () < ε then
                M2′ ← GrammarMutate (M2, G)
                M′ ← ⟨M1, M2′, M3⟩
            else
                M′ ← ⟨M1, RandMutate (M2), M3⟩
        else
            M2′ ← ChatNextMessage (M1, R)
            M′ ← ⟨M1, M2′, M3⟩
            PlateauLen ← 0
        R′ ← SendToServer (P_f, M′)
        if IsCrashes (M′, P_f) then
            C_x ← C_x ∪ {M′}
            PlateauLen ← 0
        else if IsInteresting (M′, P_f, S) then
            C ← C ∪ {(M′, R′)}
            S ← UpdateStateMachine (S, R′)
            PlateauLen ← 0
        else
            PlateauLen ← PlateauLen + 1
until timeout T reached or abort-signal
```

(C2) Unknown message structure:

- Grammar-guided Mutation

Evaluation

Research Questions

RQ.1 **State coverage.** How much more state coverage does ChatAFL achieve compared to baselines?

RQ.2 **Code coverage.** How much more code coverage does ChatAFL achieve compared to baselines?

RQ.3 **New bugs.** Is ChatAFL useful in discovering previously unknown bugs?

Subject Programs

- Live555
- ProFTPD
- PureFTPD
- Kamailio
- Exim
- Forked-daapd

Comparisons

- AFLNet
- NSFuzz

Our tool ChatAFL and dataset are publicly available at:

https://github.com/ChatAFLndss/ChatAFL

Artifact Evaluated
NDSS SYMPOSIUM
Available
Functional
Reproduced

State Space Coverage

Subject	CHATAFL	Transition comparison with AFLNET				Transition comparison with NSFUZZ			
		AFLNET	Improv	Speed-up	$\hat{A}_{12}$	NSFUZZ	Improv	Speed-up	$\hat{A}_{12}$
Live555	160.00	83.80	90.98%	228.62×	1.00	90.20	77.38%	63.09×	1.00
ProFTPD	246.70	172.60	42.91%	7.12×	1.00	181.20	36.11%	4.97×	1.00
PureFTPD	281.80	216.90	29.91%	5.61×	1.00	206.10	36.72%	7.94×	1.00
Kamailio	130.00	99.90	30.14%	5.53×	1.00	105.30	23.42%	4.58×	1.00
Exim	108.40	62.70	72.98%	40.27×	1.00	69.50	55.97%	13.25×	1.00
forked-daapd	25.40	21.40	18.65%	1.58×	1.00	20.10	26.52%	1.79×	0.86
AVG	-	-	47.60%	48.12×	-	-	42.69%	15.94×	-

Achieve same transition number 48.12× and 15.94× faster, respectively

Subject	CHATAFL	AFLNET	Improv	NSFUZZ	Improv	Total
Live555	14.20	10.00	41.75%	11.70	21.16%	15
ProFTPD	28.70	22.60	26.84%	24.30	17.81%	30
PureFTPD	27.90	25.50	9.37%	24.00	16.20%	30
Kamailio	17.00	14.00	21.43%	15.10	12.50%	23
Exim	19.50	14.10	38.19%	14.40	35.42%	23
forked-daapd	12.10	8.70	39.74%	8.00	51.39%	13
AVG	-	-	29.55%	-	25.75%	-

Cover 29.55% and 25.75% more states, respectively

Code Coverage

Subject	ChatAFL	Branch comparison with AFLNet				Branch comparison with NSFuzz			
		AFLNet	Improv	Speed-up	$\hat{A}_{12}$	NSFuzz	Improv	Speed-up	$\hat{A}_{12}$
Live555	2,928.40	2,860.20	2.38%	9.61×	1.00	2,807.60	4.30%	21.60×	1.00
ProFTPD	5,143.30	4,763.00	7.99%	4.04×	1.00	4,421.80	16.32%	21.96×	1.00
PureFTPD	1,134.30	1,056.30	7.39%	1.60×	0.91	1,041.10	8.96%	1.60×	1.00
Kamailio	10,064.00	9,404.10	7.02%	12.69×	1.00	9,758.70	3.13%	2.95×	1.00
Exim	3,789.40	3,647.60	3.89%	4.27×	1.00	3,564.30	6.32%	11.33×	0.77
forked-daapd	2,364.80	2,227.10	6.18%	4.63×	1.00	2,331.30	1.43%	1.66×	0.70
AVG	-	-	5.81%	6.14×	-	-	6.74%	10.18×	-

Achieve same branch number 6.14× and 10.18× faster, respectively

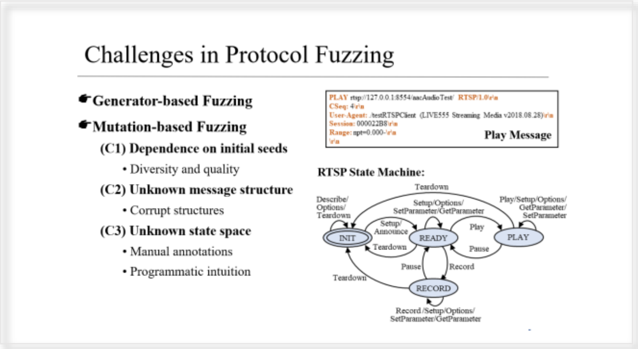
Discovering New Bugs

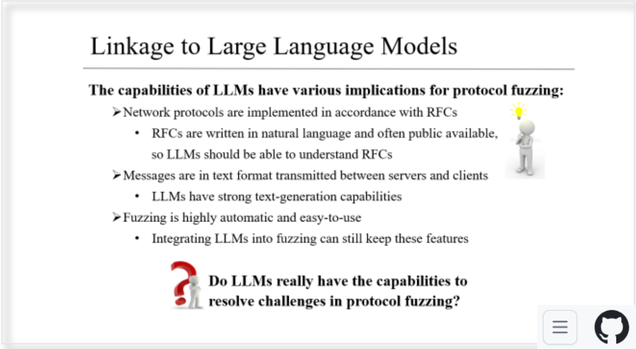
ID	Subject	Version	Bug Description	Potential Security Issue	Status
1	Live555	2023.05.10	Heap use after free in handling PLAY client requests	Remote code execution	Fixed
2	Live555	2023.05.10	Heap use after free in handling SETUP client requests	Remote code execution	Fixed
3	Live555	2023.05.10	Use after return in handling DESCRIBE client requests	Remote code execution	Fixed
4	Live555	2023.05.10	Use after return in handling SETUP client requests	Remote code execution	CVE-2023-37117, fixed
5	Live555	2023.05.10	Heap buffer overflow in handling stream	Remote code execution	Fixed
6	Live555	2023.05.10	Memory leaks after allocating memory for stream parameters	Memory leakage	Reported
7	Live555	2023.05.10	Heap use after free in calling RTPInterface::sendDataOverTCP	Remote code execution	Fixed
8	ProFTPD	61e621e	Heap buffer overflow while parsing FTP commands	Remote code execution	CVE-2023-51713, fixed
9	Kamailio	a220901	Memory leaks after allocating memory in parsing config files	Memory leakage	Reported

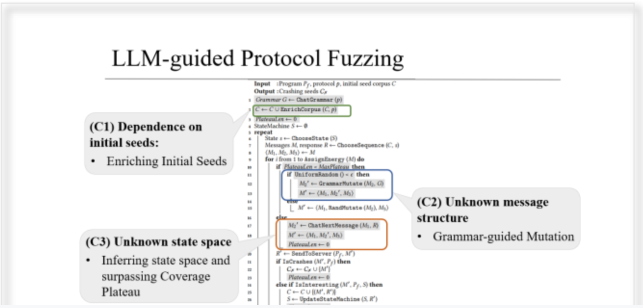
CVSS Severity Score: 9.8 Critical

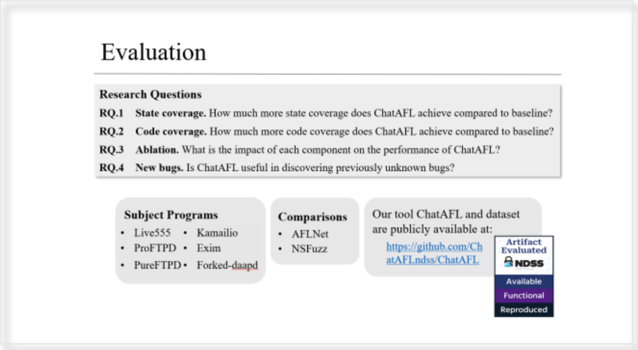
CVSS Severity Score: 7.5 High

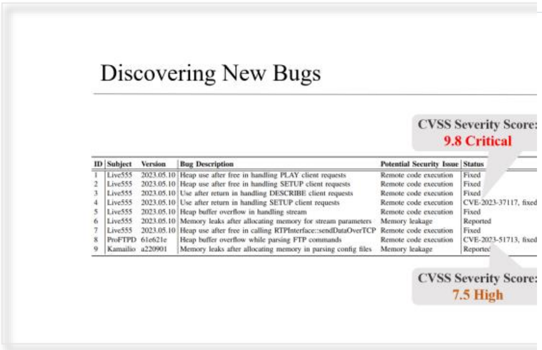
Summary

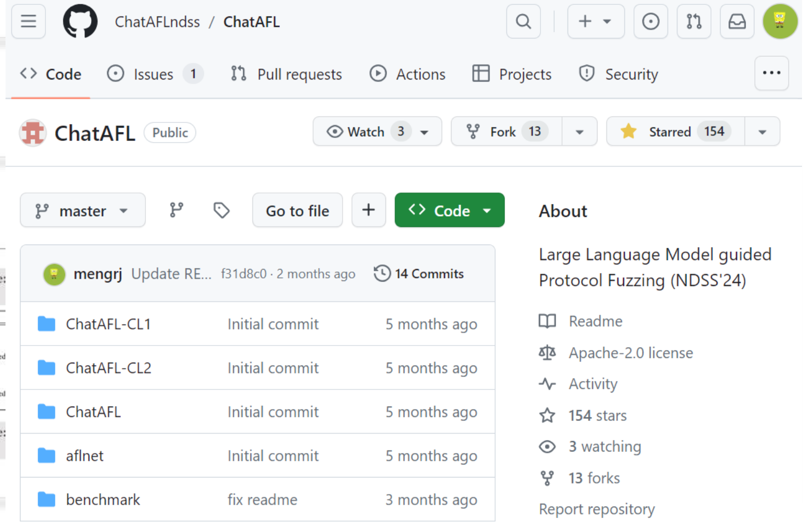












THANKS!!