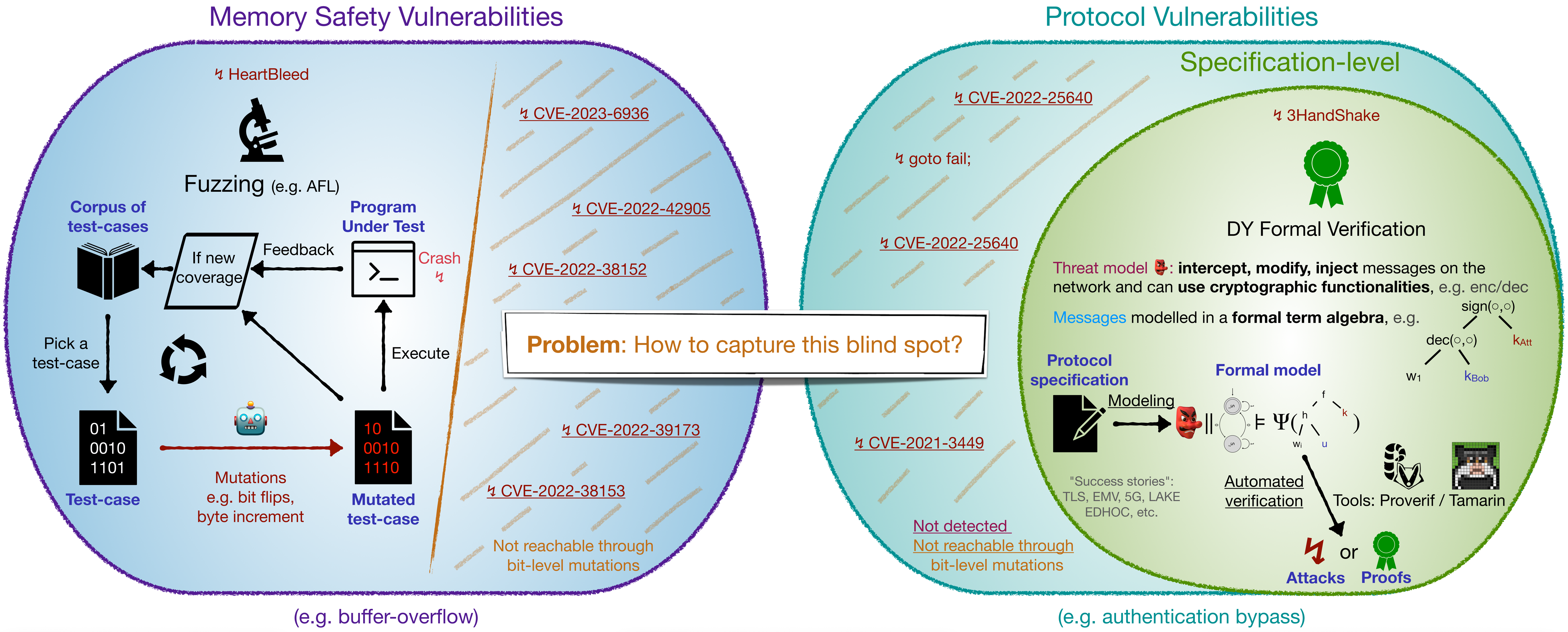
Dolev-Yao Fuzzing:
Formal Dolev-Yao Models Meet Cryptographic Protocol Fuzz Testing

Max Ammann — Trail of Bits, USA

Lucca Hirschi — Inria, France

Steve Kremer — Inria, France



✗ Fuzzing Limitation 1: Reachability
Bitstring-level mutations only

No structural message/message flow modification, e.g. negligible probability of computing crypto and other structural modifications through bit-level mutations

✗ Fuzzing Limitation 2: Detection
Inability to detect protocol vulnerabilities

Protocol vulnerabilities do not manifest themselves as crashes or memory corruption, e.g. authentication bypass

✗ DY Verification Limitation: Specification only
No guarantee on implementation

State: DY Test-Case

- We build on « messages as formal terms » and assume a set of function symbols. Example: dec(∘,∘), enc(∘,∘), sign(∘,∘)
- Test cases = symbolic traces expressing DY attacker's actions

tr := **out**(r, w).tr | **in**(r, R).tr | 0   // R is a term, w a variable, r a role

Example: **out**(client,w_1).
in(serv,w_1). // attacker only relays message w_1 to serv
out(serv,w_2).
in(client, sign(dec(w_2,k_{Bob}), k_{Att}))
// attacker computes a new term out of w_2 and sends it to client

DY Mutations

Action-level Mutations

- Skip: remove random action (in/out)
- Repeat: randomly copy and insert an action

Term-level Mutations

- Swap: Swap two (sub-)terms in the trace
- Generate: Replace a term by a random one
- Replace-Match: Swap two function symbols (e.g. SHA2 <-> SHA3)
- Replace-Reuse: Replace a (sub-)term by another (sub-)term
- Replace-and-Lift: Replace a (sub-)term by one of its sub-terms

tlspuffin: a full-fledge DY fuzzer

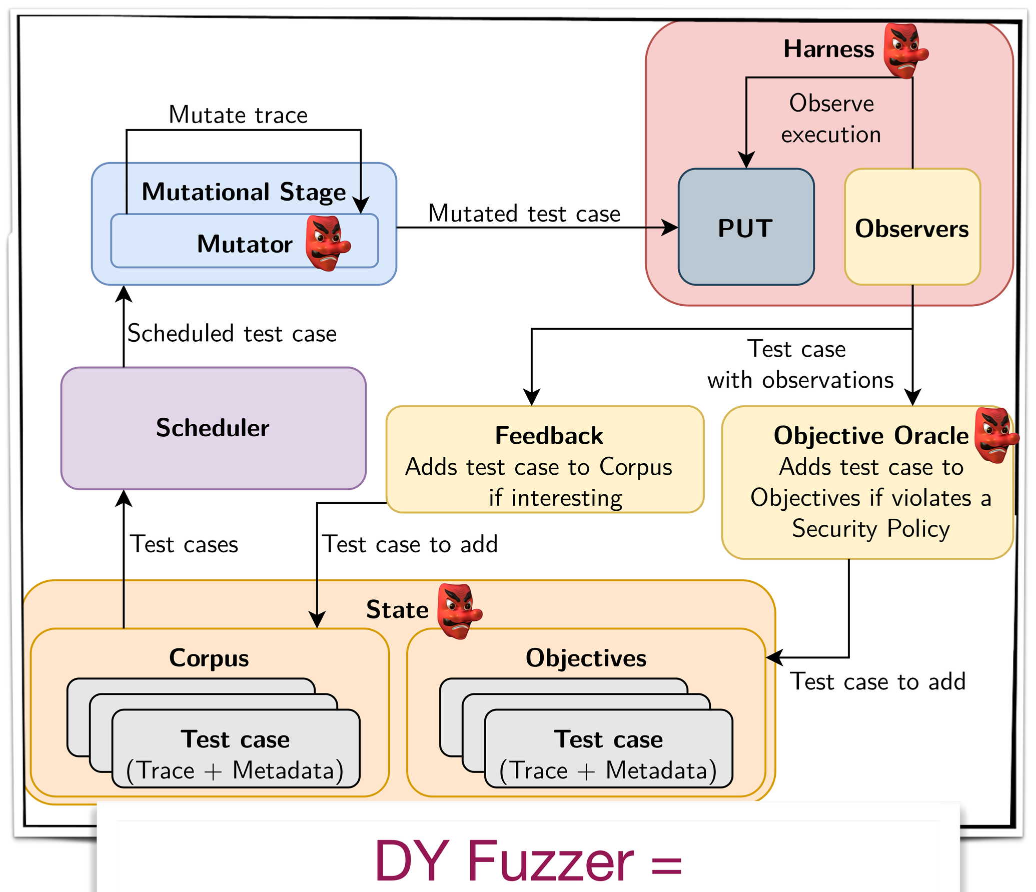
- Open-source project written in Rust (16k LoC) (tlspuffin on Github)
- Built on LibAFL, a modular library to build fuzzers
- Made modular: new protocol and PUTs can be added
- For TLS: 189 function symbols and Open/Boring/Wolf/LibreSSL as PUTs
- We ran tlspuffin on those and found 8 CVE, including 5 new CVEs

Other state-of-the-art fuzzers do not found those, we do thanks to DY Fuzzing

Checkout our website: https://tlspuffin.github.io

Harness: Mapper + Executor

- To each function symbol f, we build an interpretation $[\![f]\!] : [u8]^n \to [u8]$
  Example: $[\![sign]\!](m, key) :=$ `ECDSA(m, key)`
- Mapper can interpret any term by recursively applying interpretations $[\![\cdot]\!] : \mathcal{T}erms \to [u8]$
- Mapper is protocol-dependent but PUT-independent and can be built once-for-all on top of a reference implementation or *any* PUT



DY Fuzzer =
DY attacker in a fuzzing loop

CVE ID	CVSS	Type	New	Target
2021-3449	5.9	Server DoS, M	✗	OpenSSS
2022-25638	6.5	Auth. Bypass, P	✗	WolfSSL
2022-25640	7.5	Auth. Bypass, P	✗	WolfSSL
2022-38152	7.5	Client DoS, M	✓	WolfSSL
2022-38153	5.9	Server DoS, M	✓	WolfSSL
2022-39173	7.5	Server DoS, M	✓	WolfSSL
2022-42905	9.1	Info. Leak, M	✓	WolfSSL
2023-6936	5.3	Info. Leak, M	✓	WolfSSL

Executor

Executor concretizes DY traces (tr) with the PUT (e.g. OpenSSL):

1. Initialize all agents, client and serv, and their IO buffers
2. On output actions: e.g. **out**(client, w)
   a. call PUT to read bitstring b_w from output buffer of client
   b. let client progress
3. On input actions: e.g. **in**(serv, R)
   a. invoke Mapper to concretise term R into a bitstring $b_R := [\![R]\!]$
   b. call PUT to write b_R onto input buffer of serv
   c. let serv progress

DY Objective Oracle

Memory-related objective oracle

- Classical with bit-level fuzzing: code instrumentation with AddressSanitizer (*ASan*)

DY security properties checking

- Introduce claims triggered by roles executing the PUT
  E.g. agreement claims: Agr(client, pk, m)@i means "client believes to have agreed with a server with public key pk on m at i^th action"
- As in DY models: security properties expressed as 1^st-order formula
  E.g. auth. ∀pk,m: Agr(client, pk, m)@i ⇒ Run(server, pk, m)@j ∧ j<i
- Objective Oracle always checks those properties by first applying $[\![\cdot]\!]$

Future Work

- Code-coverage is a poor metric prone to exhaustion, we plan to design a domain-specific DY-based notion of coverage
- Explore differential fuzzing + extend objective oracle (with more properties and compromise scenarios)
- Combine DY fuzzing with bit-level fuzzing: reach deep states and then smash PUTs with bit-level mutations [WIP]
- Apply DY fuzzing to more protocols (e.g WPA*, TelCo, etc.) and PUTs
- Partially automate the Mapper (and Harness) → PUT/Protocol-agnostic
- Connect further with DY verification tools (ProVerif/Tamarin)

IEEE Security and Privacy, May 20, 2024