DY Fuzzing:
Putting a Dolev-Yao attacker in the fuzzing loop

Steve Kremer, Inria Nancy, France

Max Ammann & Lucca Hirschi & Tom Gouville & Michaël Mera
(Trail of Bits & Inria)

Journées du GT MFS 2024

joint work with
Secure Cryptographic Protocols
Cryptographic Protocols

Informal definition
concurrent program relying on cryptography to secure communications

Security goals: confidentiality, integrity/authentication, etc.
Examples: TLS, EMV (credit cards), RFID, e-voting, mobile com., etc.

• Notoriously difficult to design and deploy securely
• Loads of failure stories: attacks, fixes, attacks, fixes, attacks, etc.

💡 What can we do today to avoid such failures in the future?
Retrospective of TLS Failures 2014-2022

Spatial and temporal memory bugs (e.g. buffer-overflow)

HeartBleed
Gnu's GotoFail
CloudBleed
WinShock

Protocol vulnerabilities (e.g. authentication violation)

Apple's GotoFail
CVE-2022-25640
CVE-2021-3449
CVE-2022-25638
POODLE
3SHAKE

DY formal verification

Affects the specification

Renegotiation
LOGJAM

CVE-2022-25638
Apple's GotoFail
POODLE
3SHAKE
1: Formally Verifying Cryptographic Protocols Designs
Dolev-Yao Formal Model

Formal model for analyzing cryptographic protocols amenable to automation

Threat model 🧟:

• active adversary controls the network: intercept, modify, inject messages
• is able to use cryptography
• cryptography considered black-box (attacker’s interface = functionality)

Attacker can use encryption and decryption but does not see the internals (e.g., AES S-box) and cannot exploit potential leaks/biasis

« Messages as formal terms » paradigm: messages model = term algebra

1. Set of function symbols: e.g. senc(·,·), sdec(·,·)
2. Equivalence relation: e.g. sdec(senc(m, k), k) = m
Dolev-Yao Formal Model

✓ Sweet spot between **precision** (of results) and **automation** (verification algorithms) Excel at finding **logical attacks**: protocol vulnerabilities at the design-level

Proverif 🐻 Tamarin

✗ Limited to specifications, *existing* implementations are out of scope (e.g., OpenSSL)
Retrospective of TLS Failures

- Spatial and temporal memory bugs (e.g. buffer-overflow)
  - HeartBleed
  - Gnu's GotoFail
  - CloudBleed
  - WinShock

- Protocol vulnerabilities (e.g. authentication violation)
  - Apple's GotoFail
  - CVE-2022-25640
  - CVE-2021-3449
  - CVE-2022-25638
  - FREAK
  - CVE-2022-25638
  - SKIP
  - Poodle
  - 3SHAKE
  - LOGJAM
  - Renegotiation
  - DY formal verification

- Bit-Level Fuzzers (e.g. AFLnet)

- CVE-2021-3449
- POODLE
- FREAK
- Apple's GotoFail
- CVE-2022-25638

Retrospective of TLS Failures 2014-2022
2: Fuzzing
Cryptographic Protocols Implementations
—State-of-the-Art
Bit-level fuzzing (AFL-like)

What is fuzzing?
- Instrument the PUT to record feedback (e.g. code coverage)
- Store a corpus of test-cases

Fuzzing loop: while true do
  - Pick a test-case
  - Apply random transformation = mutation
  - Execute + collect feedback
  - Add it to the corpus if interesting according to feedback = progress (e.g. new coverage)

Examples of mutations:
- bit flip
- byte increment
- mixing bytes around

Seed corpus → Corpus of test-cases

Test-case: 0xd404
Random mutation
Seed corpus

Test-case: 0xe504
Execute
Seed corpus

Program Under Test

Output: 0x4fad1...
Bit-level fuzzing (AFL-like)

✓ Finds **memory/crash vulnerabilities** in implementations  
   E.g. buffer-overflow, use after free, RCE, etc.

✗ Bitstring-level mutations only
   • No structural message modification  
     e.g. negligible probability of computing a valid signature through bit-level mutations  
   • No message flow modification  
     e.g. protocol executions ≠ one bitstring

_logical attack_ states are **not reached**

+ **miss** some memory vulnerabilities requiring those

✗ Detect crashes only
   Protocol vulnerabilities are **not detected**  
   e.g. authentication bypass (no crash)
Spatial and temporal memory bugs (e.g. buffer-overflow)

- HeartBleed
- Gnu's GotoFail
- Apple's GotoFail
- FREAK
- SKIP

- Renegotiation
- POODLE
- 3SHAKE
- LOGJAM

- Protocol vulnerabilities (e.g. authentication violation)

- Requires structural modifications to messages

- Requires message flow modifications

- Affects the specification

- Bit-Level Fuzzers e.g. AFLnet
- CloudBleed

- DY formal verification

Retrospective of TLS Failures 2014-2022
Retrospective of TLS Failures

2014-2022

Affects the specification

Spatial and temporal memory bugs (e.g. buffer-overflow)

Bit-Level Fuzzers e.g. AFLnet

CloudBleed

Spatial and temporal memory bugs

(e.g. buffer-overflow)

Requires message flow modifications

Requires structural modifications
to messages

Protocol vulnerabilities (e.g. authentication violation)

DY Fuzzing

DY formal verification

CVE-2022-6936 (us)
HeartBleed

CVE-2022-39173 (us)

CVE-2022-38152 (us)

CVE-2022-38153 (us)
Gnu's GotoFail

CVE-2022-42905 (us)
WinShock

CVE-2022-25640

Apple's GotoFail

CVE-2021-3449

CVE-2022-25638

FREAK*

CVE-2022-25638

CVE-2022-25640

CVE-2023-6936 (us)

CVE-2022-38153 (us)

CVE-2022-39173 (us)

CVE-2023-6936 (us)

POODLE

3SHAKE

LOGJAM

Renegotiation

CVE-2021-3449

CVE-2022-38152 (us)

CVE-2022-38153 (us)

FREAK

CVE-2022-25638

SKIP

Retrospective of TLS Failures 2014-2022

Gnu's GotoFail

WinShock

CPE-2022-42905 (us)

CPE-2022-38152 (us)

CPE-2022-39173 (us)

CPE-2022-6936 (us)

CPE-2023-6936 (us)

CPE-2022-38153 (us)

CPE-2021-3449

CPE-2022-25640

CPE-2022-25638

CPE-2022-38152 (us)

CPE-2022-38153 (us)

CPE-2023-6936 (us)

CPE-2021-3449

CPE-2022-25640

CPE-2022-25638

CPE-2023-6936 (us)

CPE-2021-3449

CPE-2022-25640

CPE-2022-25638

CPE-2023-6936 (us)
3: Our proposal: Dolev-Yao Fuzzing


Max Ammann*
Independent Researcher &
Trail of Bits
max@maxammann.org

Lucca Hirschi
Inria Nancy Grand-Est
Université de Lorraine, LORIA, France
lucca.hirschi@inria.fr

Steve Kremer
Inria Nancy Grand-Est
Université de Lorraine, LORIA, France
steve.kremer@inria.fr

Paper accepted at IEEE Security and Privacy 2024
Preprint IACR 2023/057
DY Fuzzing Design
DY Fuzzing: Big Picture

- We build on « messages as formal terms »: and assume a term algebra

- Test cases = symbolic traces expressing DY attacker 👹’s actions

```
tr := out(r, w).tr : r is a role (client/server) and w is a variable (attacker knows)
| in(r, R).tr : R is a term in the term algebra (computed by attacker)
| 0
```

Example: \( tr_a = \text{out(cl, } w_1).\text{in(serv, } w_1).\text{out(serv, } w_2).\text{in(cl, senc(sdec(w_2, k_a), k_b))}.0 \)

Attacker 👹 only relays the message \( w_1 \) to \( \text{serv} \)
Attacker 👹 computes a new term \( R \) out of \( w_2 \) and sends it to \( \text{cl} \)
DY Fuzzing: Big Picture

Symbolic traces ($tr$) are « concretized » with the PUT (or any ref. implem.)

1. out($r, w$) call PUT role function to read bitstring $b_w$ from output buffer of $r$
2. in($r, R$)
   a. call ref/PUT crypto functions to evaluate $R$ into a bitstring $b_R$
      E.g. $eval(sign(R', sk)) = RSA_{PUT}(eval(R'), b_{sk})$
      $eval(w) = b_w$
      $b_{sk}$ is obtained by calling $genKey_{PUT}()$
   b. call PUT role function to write $b_R$ onto input buffer of $r$ + make $r$ progress

Executor (1 + 2.b): require a lightweight instrumentation of the PUT
Mapper (2.a): requires a per-protocol « executable term-algebra »

Do not require a protocol DY model but only a DY attacker model (i.e., term algebra)
DY Fuzzer components

• **State**: test-cases = DY traces, seeds corpus = happy flows

• **Scheduler**: FIFO

• **Mutator**: custom trace mutations

• **Harness**: Mapper + Executor + Claims

• **Obj. Oracle**: DY security properties (e.g. agreement) + ASAN (memory vulns.)

• **Feedback**: PUT code-coverage

LibAFL components (we build on)
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 in the trace (e.g. SHA2 <-> SHA3)
- **Replace-Reuse**: Replace a (sub-)term by another (sub-)term in the trace
- **Replace-and-Lift**: Replace a (sub-)term by one of its sub/terms
DY Fuzzer components

- **State**: test-cases = DY traces, seeds corpus = happy flows
- **Scheduler**: FIFO
- **Mutator**: custom trace mutations
- **Harness**: Mapper + Executor + Claims
- **Obj. Oracle**: DY security properties (e.g. agreement) + ASAN (memory vulns.)
- **Feedback**: PUT code-coverage
DY Objective Oracle

Memory-related objective oracle

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

  +

DY Security properties ★

• Introduce claims triggered by roles executing the PUT (part of Harness/Executor)
  E.g. agreement claims: Agr(client, pk, m)@i client believes to have agreed with server with pk on m @ i th action

• Classical in DY models: security properties expressed as 1st-order formula
  E.g. agreement property ∀pk,m: Agr(client, pk, m)@i ⇒ Run(server, pk, m)@j ∧ j<i

• DY Objective oracle also checks DY security properties
  • Gather all the claims throughout traces executions at the PUT
  • Check all the DY security properties (where terms are concretized into bitstrings)
tlspuffin Implementation
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 (+ new/custom components⭐)

- In-memory buffers, delightfully parallel, fast (700 execs/s/core)

- Modular: new protocol and new PUTs can be added

- For TLS: 189 function symbols, harnessed PUTs: OpenSSL, WolfSSL, BoringSSL, LibreSSL
tlspuffin  Results
tlspuffin findings

• We selected a small benchmark suite: recent logical attacks found on OpenSSL (most used) and WolfSSL (IoT).

• Found by tlspuffin in hours or seconds (SKIP), systematic reproducibility!

• We ran fuzzing campaigns on the harnessed PUTs and found 5 new CVEs ☹️ Not found by other fuzzers

<table>
<thead>
<tr>
<th>CVE ID</th>
<th>AKA</th>
<th>CVSS</th>
<th>Type</th>
<th>New</th>
<th>Version</th>
<th>TLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2021-3449</td>
<td>SDOS1</td>
<td>5.9</td>
<td>Server DoS, M</td>
<td>X</td>
<td>1.1.1j</td>
<td>1.2</td>
</tr>
<tr>
<td>2022-25638</td>
<td>SIG</td>
<td>6.5</td>
<td>Auth. Bypass, P</td>
<td>X</td>
<td>5.1.0</td>
<td>1.3</td>
</tr>
<tr>
<td>2022-25640</td>
<td>SKIP</td>
<td>7.5</td>
<td>Auth. Bypass, P</td>
<td>X</td>
<td>5.1.0</td>
<td>1.3</td>
</tr>
<tr>
<td>2022-38152</td>
<td>SDOS2</td>
<td>7.5</td>
<td>Client DoS, M</td>
<td>✓</td>
<td>5.4.0</td>
<td>1.3</td>
</tr>
<tr>
<td>2022-38153</td>
<td>CDOS</td>
<td>5.9</td>
<td>Server DoS, M</td>
<td>✓</td>
<td>5.3.0</td>
<td>1.2</td>
</tr>
<tr>
<td>2022-39173</td>
<td>BUF</td>
<td>7.5</td>
<td>Server DoS, M</td>
<td>✓</td>
<td>5.5.0</td>
<td>1.3</td>
</tr>
<tr>
<td>2022-42905</td>
<td>HEAP</td>
<td>9.1</td>
<td>Info. Leak, M</td>
<td>✓</td>
<td>5.5.0</td>
<td>1.3</td>
</tr>
<tr>
<td>2023-6936</td>
<td>HEAP2</td>
<td>N/A</td>
<td>Info. Leak, M</td>
<td>✓</td>
<td>5.6.5</td>
<td>1.3</td>
</tr>
</tbody>
</table>
Root causes of **CVE-2022-39173** (WolfSSL, CVSS high)

1. **Attacker acting as client** performs a full TLS handshake, establishing a Pre-Shared-Key (PSK)
2. Forges a malicious `ClientHello([c;...;c])` message such that
   (a) it **resumes previous session with PSK** (needs to apply decrypt, hash, signature) and
   (b) It has as list of supported cipher suites with **duplicates of c** (say n times)

☞ Server calls `refineSuites` to update `suitesS` (ciphers offered both by client and server) bc. of **resumption**
☞ **Flaw 1**: actually computes « multiset-intersection » so `suitesS` will contain **duplicates of c** (say k times)
Root causes of **CVE-2022-39173** (WolfSSL, CVSS high)

1. Attacker acting as client performs a full TLS handshake, establishing a Pre-Shared-Key (PSK)
2. Forges a malicious `ClientHello([c;..;c])` message such that
   (a) it resumes previous session with PSK (needs to apply decrypt, hash, signature) and
   (b) It has as list of supported cipher suites with duplicates of c (say n times)

Server calls `refineSuites` to update `suitesS` (ciphers offered both by client and server) bc. of resumption

**Flaw 1:** actually computes « multiset-intersection » so `suitesS` will contain duplicates of c (say k times)

---

How WolfSSL implements `∩ with refineSuites(suitesC)`@tls13.c:4355

```c
// suitesS initially with offered suites, MAX_SZ allocated
byte suites[MAX_SZ]; int suiteSz = 0; // supposed to compute suitesS ∩ suitesC

for (i = 0; i < suitesS.size; i += 1) {
    for (j = 0; j < suitesC.size; j += 1) { // suitesC.size <= MAX_SZ
        if (suitesS->suites[i] == suitesC->suites[j]) {
            suites[suiteSz++] = suitesC->suites[j]; } } } } 

XMEMCPY(suitesS, &suites, sizeof(suites));
```
Root causes of **CVE-2022-39173** (WolfSSL, CVSS high)

1. **Attacker acting as client** performs a full TLS handshake, establishing a Pre-Shared-Key (PSK)

2. Forges a malicious `ClientHello([c;..;c])` message such that
   (a) it **resumes previous session with PSK** (needs to apply decrypt, hash, signature) and
   (b) It has as list of supported cipher suites with **duplicates of c** (say n times)
   
   ✤ Server calls `refineSuites` to update `suitesS` (ciphers offered both by client and server) bc. of **resumption**
   ✤ **Flaw 1**: actually computes « multiset-intersection » so `suitesS` will contain **duplicates of c** (say k times)
   ✤ No big deal because `suitesS` initially had no duplicate so: \( k = n \leq |suitesC| \leq MAX_SZ = 150 \)
   
   (c) Is ill-formed and will be rejected but **late** (after call to `refineSuites`), mess with `supportGroupExtension`
   ✤ Server rejects it and sends a `HelloRetryRequest` but
   ✤ **Flaw 2**: side-effects of `refineSuites` are **not reverted**
   ✤ From now on, `refineSuites` invariant is broken: `suitesS` contains n duplicates of c

3. Send `ClientHello([c;..;c])` again, `refineSuites` is called again, the resulting buffer `suites` that contains \( k^2 = n^2 \) ciphers c is copied into `suitesS`
   ✤ For \( n = 13 \), we already overwrite the `suitesS` buffer allocated on \( MAX\_ciphers\_list\_length = 150 \)
Root causes of **CVE-2022-39173** (WolfSSL, CVSS high)

An overflow on the stack of max 44700 bytes (controlled by n so is attacker 🤡-controlled).

- Therefore, large portions of the stack can get overwritten, including return addresses (confirmed)
- Potential RCE (unconfirmed)
- Potential for negotiating ciphers that server should reject (downgrade)

(c) is ill-formed and will be rejected but late (after call to `refineSuites`), mess with supportGroupExtension

Server rejects it and sends a HelloRetryRequest but

- **Flaw 2: side-effects of refineSuites are not reverted**
- From now on, refineSuites invariant is broken: `suitesS` contains n duplicates of c

3. Send `ClientHello([c;..;c])` again, refineSuites is called again, the resulting buffer `suites` that contains 
k² = n² ciphers c is copied into `suitesS`
- For n = 13, we already overwrite the `suitesS` buffer allocated on MAX_ciphers_list_length = 150
DY Fuzzing Future Work
Future Work - Evaluation

• tlspuffin always found the new CVEs
• state-of-the art competitive fuzzers never found any of them
We can explain this with qualitative evidences but quantitative evidences are hard to obtain

• **Code-coverage** is a poor metric and prone to exhaustion

A statement reached from an attack state is similarly counted as if reached from the happy flow

*E.g. client accepting a legitimate server’s certificate \( \sim \) coverage accepting illegitimate cert.*

Need for a domain-specific DY-based notion of coverage + balance with code-cov.
Future Work (cont.)

Improved objective oracle
- Differential fuzzing: save $t$ as objective when $\text{WolfSSL}(t) \neq \text{OpenSSL}(t)$
- Or extend the oracle: +properties & +compromise scenarios

[WIP] Combine DY fuzzing with bit-level fuzzing (WIP): reach «deep states» with DY attacker and then smash the PUT with some bit-level mutations

Apply DY fuzzing to more protocols (e.g. WPA2, TelCo) and PUTs

Long-Term:
- (Partially) Automate Mapper and Harness → PUT-agnostic DY fuzzer
- Connect further with DY verifiers (ProVerif, Tamarin)
Summary of Contributions

1. A new approach to fuzzing cryptographic protocols connecting the DY formal approach with fuzzing → captures for the first time the class of logical attacks / DY attacker

2. DY Fuzzing design specification

3. tlspuffin: full-fledged, modular, efficient DY fuzzer implementation for TLS

4. Evaluate tlspuffin on TLS libraries:
   - (re)found 8 vulnerabilities
   - including 5 new ones (incl. 1 critical & 2 high)

Paper will appear at IEEE S&P 2024
Preprint IACR 2023/057


Max Ammann*
Independent Researcher
Trail of Bits
max@maxammann.org

Luca Hirschi
Inria Nancy Grand Est
Université de Lorraine, LORIA, France
luca.hirschi@inria.fr

Steve Kremer
Inria Nancy Grand Est
Université de Lorraine, LORIA, France
steve.kremer@inria.fr

v1.0. — January 18, 2023

Project ANR JCJC
→ Looking for students/postdocs/engineers

PROTOFuzz: Cryptographic Protocol Logic Fuzz Testing
Formal Verification Meets Fuzz Testing
Consortium: PESTO (Inria Nancy)