Poster: Analyzing Semantic Correctness of Security-critical Algorithm Implementations with Symbolic Execution

Author: Sze Yiu Chau (schau@purdue.edu) Affiliation: Purdue University

Poster Abstract: In order to achieve security, protocol implementations not only need to avoid low-level memory access errors, but also faithfully follow and fulfill the requirements prescribed by the protocol specifications at the semantic level. Failure to do so could lead to compatibility issues and damage the security guarantees intended by the original design. In this poster, I will discuss how to use symbolic execution to analyze semantic correctness of implementations of security-critical algorithms. The main intuition is that, while symbolic execution faces scalability challenges, it provides a systematic means of exploring possible execution paths and a formula-based abstraction, both of which are useful in finding semantic level implementation flaws. In many cases, scalability challenges can be avoided with concolic inputs carefully crafted by exploiting features of the input formats used by target protocols, along with optimizations based on domain knowledge that can help prune the search space. As examples, the poster will first present our previous work on analyzing implementations of X.509 certificate validation. Our analysis of 9 small footprint TLS libraries has uncovered 48 instances of noncompliance, as well as some inaccurate claims in a previous work based on blackbox fuzzing. It will then discuss our most recent work on analyzing implementations of PKCS#1 v1.5 RSA signature verification, and explain how some of the implementation flaws we found in crypto libraries and IPSec software suites can lead to authentication bypass and denial-of-service attacks due to new variants of the Bleichenbacher-style low-exponent RSA signature forgery. Altogether, 9 new CVEs of varying degree of severity have been assigned thanks to this series of research.
(1) Motivation

- Semantic correctness is fundamental to achieving security
  - Q: Is an algorithm implementation faithfully following the specification?
  - Just because it doesn’t crash, doesn’t mean that it is correct
- How do we reason about the semantic correctness of an algorithm implementation?

(2) Symbolic Execution … More Than Just Automatic Test Case Generation

- Blackbox fuzzing is a prominent software testing approach, but
  - Hard to reason about code internals with only observable inputs and outputs
- Symbolic execution provides
  - in general better code coverage
  - a very useful abstraction in the form of logical formula
- Scalability challenges of symbolic execution can be worked around with
  - strategically mixing concrete values with symbolic variables
  - Resemble the idea of “Grammar-based whitebox fuzzing” [Godefroid et al., PLDI ’08]
- Get through parsing quickly and focus on the security-critical validation logic
  - other domain-specific optimizations
- Two success stories on finding semantic correctness issues in deployed implementations:
  - X.509 Certificate Validation [Chau et al., IEEE S&P ’17]
  - PKCS#1 v.1.5 Signature Verification [Chau et al., NDSS ’19]

(3) Research Focus

Goal: Expose RFC Violations in X.509 implementations
- Focus our analysis on small-footprint, small code-base libraries

- Domain-specific optimizations: No crypto and simplified string matching

(4) Extracting the Validation Logic

X.509 Certificate Chain Input Universe

- SymCert

Sets of Logical Formulas

- Symmetric Execution Engine

Rejecting Universe (approx.)

Accepting Universe

(5) Finding Flaws Through Simple Inspections and Cross-Validation (Differential Testing)

- Accepting Universe (approx.)

- Rejecting Universe (approx.)

- R1 (approx.)

- R2 (approx.)

- A1 (approx.)

- A2 (approx.)

(6) Summary of Experiments and Findings

- Tested 9 implementations from 4 families of SSL/TLS libraries

<table>
<thead>
<tr>
<th>Library - version</th>
<th>Released</th>
<th>Lines of C code</th>
<th>Total Paths</th>
<th>Exposure Time</th>
<th>Violations</th>
</tr>
</thead>
<tbody>
<tr>
<td>axTLS 1.4.3</td>
<td>Jul 2011</td>
<td>16,283</td>
<td>~ 0.8K</td>
<td>~ 1 Minute</td>
<td>7</td>
</tr>
<tr>
<td>axTLS 1.5.3</td>
<td>Apr 2015</td>
<td>16,832</td>
<td>~ 0.8K</td>
<td>~ 1 Minute</td>
<td>6</td>
</tr>
<tr>
<td>tropicSSL (Ghids)</td>
<td>Mar 2013</td>
<td>13,610</td>
<td>~ 0.2K</td>
<td>~ 1 Minute</td>
<td>10</td>
</tr>
<tr>
<td>* PolarSSL - 1.2.8</td>
<td>Jun 2013</td>
<td>29,470</td>
<td>~ 0.3K</td>
<td>~ 1 Minute</td>
<td>4</td>
</tr>
<tr>
<td>* mbedTLS - 2.14</td>
<td>Jan 2016</td>
<td>53,433</td>
<td>~ 0.6K</td>
<td>~ 2 Minutes</td>
<td>7</td>
</tr>
<tr>
<td>* CySSL - 2.70</td>
<td>Jun 2013</td>
<td>51,786</td>
<td>~ 0.6K</td>
<td>~ 2 Minutes</td>
<td>7</td>
</tr>
<tr>
<td>wolfSSL - 3.6.6</td>
<td>Aug 2015</td>
<td>103,690</td>
<td>~ 32K</td>
<td>~ 1 Hour</td>
<td>2</td>
</tr>
<tr>
<td>* MatrixSSL - 3.4.2</td>
<td>Feb 2017</td>
<td>18,360</td>
<td>~ 0.2K</td>
<td>~ 1 Minute</td>
<td>6</td>
</tr>
<tr>
<td>MatrixSSL - 3.7.2</td>
<td>Apr 2017</td>
<td>35,879</td>
<td>~ 12K</td>
<td>~ 1 Hour</td>
<td>5</td>
</tr>
</tbody>
</table>

Total: 48

(7) Notable Findings and Their Implications

- Various unwaranted leniencies in accepting malformed signatures
  - Some lead to immediate practical signature forgeries when e is small enough
  - 6 new CVEs assigned for the newly discovered and exploitable flaws
    - CVE-2018-15836 assigned for Openswan 2.6.50 (CVSS v3.0 score: 7.5 High Severity)
    - CVE-2018-16151 assigned for strongSwan 5.6.3 (CVSS v3.0 score: 7.5 High Severity)
    - CVE-2018-16152 assigned for strongSwan 5.6.3 (CVSS v3.0 score: 7.5 High Severity)
    - CVE-2018-16253 assigned for axTLS 2.1.3 (CVSS v3.0 score: 5.9 Medium Severity)
    - CVE-2018-16150 assigned for axTLS 2.1.3 (CVSS v3.0 score: 5.9 Medium Severity)
    - CVE-2018-16149 assigned for axTLS 2.1.3 (CVSS v3.0 score: 5.9 Medium Severity)
  - Particularly bad for IPsec software suites because some key generation programs (e.g., ‘ipsec_rsa_signkey’ on Ubuntu) forces e = 3
  - axTLS suffers from both potential signature forgery and Denial of Service
    - Please refer to the paper for detailed attack algorithms and complexity analysis

Symbolic execution can be quite effective in analyzing semantic correctness
- Thanks to its good coverage and formula-based abstraction of the implemented logic
- 2 success stories on analyzing X.509 certificate validation and RSA signature verification