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.

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

Sze Yiu Chau Purdue University schau@purdue.edu
https://www.cs.purdue.edu/homes/schau/

(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?

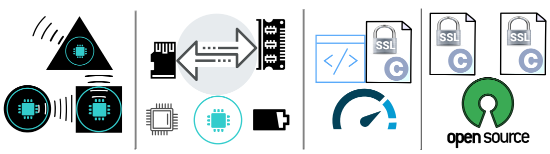
(2) Symbolic Execution . . . More Than Just Automatic Test Case Generation

- 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:
  - → ❶ *X.509 Certificate Validation* [Chau et al., IEEE S&P '17]
  - → ❷ *PKCS#1 v1.5 Signature Verification* [Chau et al., NDSS '19]

❶–① Research Focus

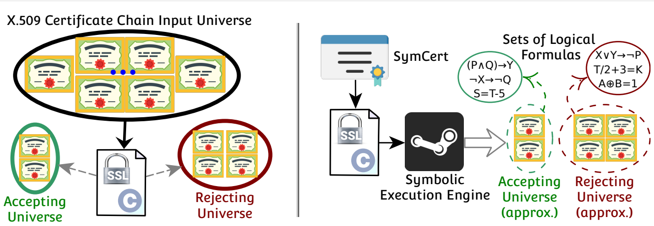
Goal: **Expose RFC Violations** in X.509 implementations

- Focus our analysis on small-footprint, small code-base libraries

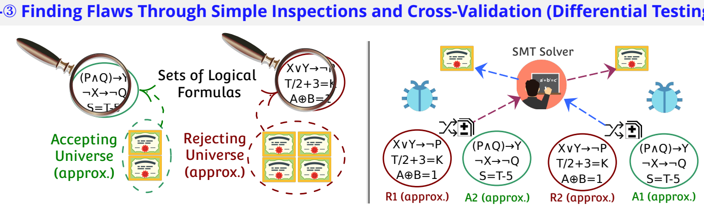


- Domain-specific optimizations: No crypto and simplified string matching

❶–② Extracting the Validation Logic



❶–③ Finding Flaws Through Simple Inspections and Cross-Validation (Differential Testing)



❶–④ Summary of Experiments and Findings

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

Library - version	Released	Lines of C code	Total Paths	Extraction Time	Violations
axTLS - 1.4.3	Jul 2011	16,283	∼ 0.8*K*	∼ 1 Minute	7
axTLS - 1.5.3	Apr 2015	16,832	∼ 0.8*K*	∼ 1 Minute	6
tropicSSL - (Github)	Mar 2013	13,610	∼ 0.2*K*	∼ 1 Minute	10
* PolarSSL - 1.2.8	Jun 2013	29,470	∼ 0.3*K*	∼ 1 Minute	4
mbedTLS - 2.1.4	Jan 2016	53,433	∼ 0.6*K*	∼ 1 Minute	1
* CyaSSL - 2.7.0	Jun 2013	51,786	∼ 0.6*K*	∼ 2 Minutes	7
wolfSSL - 3.6.6	Aug 2015	103,690	∼ 32*K*	∼ 1 Hour	2
* MatrixSSL - 3.4.2	Feb 2013	18,360	∼ 0.2*K*	∼ 1 Minute	6
MatrixSSL - 3.7.2	Apr 2015	37,879	∼ 12*K*	∼ 1 Hour	5
				Total:	**48**

❶–⑤ Notable Findings and Their Implications

- Various libraries misintrept the 2-byte year (*YY*) of *UTCTime*
  - → Some **expiration dates** are **shifted by 100 years**
    - ***CVE-2017-1000415*** assigned for MatrixSSL 3.7.2 (*CVSS v3.0 score: 5.9 Medium Severity*)
    - ***CVE-2017-1000416*** assigned for axTLS 1.5.3 (*CVSS v3.0 score: 5.3 Medium Severity*)
- Overly Permissive comparisons of OIDs in *ExtKeyUsage* (wolfSSL 3.6.6, MatrixSSL 3.7.2)
  - → Compare only the **sum of OID encoded bytes**, collision prone
    - ***CVE-2017-1000417*** assigned for MatrixSSL 3.7.2 (*CVSS v3.0 score: 5.3 Medium Severity*)

❶–⑤ Notable Findings and Their Implications (Continued)

- Reject certificates with *GeneralizedTime* (tropicSSL, axTLS 1.4.3)
- *Country, State, Locality* ignored (axTLS 1.4.3 and 1.5.3)
- Incomplete extension handling (various libraries)

❶–⑥ New Findings Comparing To Previous Work Based On Fuzzing

- Incorrectly reject valid certs due to overly restrictive date time comparisons (CyaSSL 2.7.0)
- Incorrect Extension Parsing (CyaSSL 2.7.0)
- *pathLenConstraint* ignored (CyaSSL 2.7.0, PolarSSL 1.2.8, tropicSSL and wolfSSL 3.6.6)
  - → Previous work inaccurately claimed both CyaSSL 2.7.0 and PolarSSL 1.2.8 incorrectly reject a specific corner case of *pathLenConstraint* [Brubaker et al., IEEE S&P '14]

❷–① Research Focus

Goal: **Find semantic flaws** in implementations of PKCS#1 v1.5 RSA signature verification

- PKCS#1 v1.5 signatures are widely used, e.g., X.509 Certificates, SSH, IPSec, etc.
- Some flaws are known to be exploitable for signature forgery when e is small
  - → "Bleichenbacher-style" low exponent signature forgery attacks
    - First reported by Daniel Bleichenbacher at CRYPTO '06 Rump Session
  - → Many variants found later, e.g. in OpenSSL, GnuTLS, Mozilla Firefox, etc.

❷–② Technical Improvements Over ❶

- Automatically generate concolic test cases based on relations of the components
  - → PKCS#1 v1.5 has diverse input components, e.g. ASN.1, padding w/ implicit length
  - → Programmatically prepare input buffers according to combinations of component lengths
- Easier root cause analysis with Contstraint Provenance Tracking (CPT)
  - → Want to be able to go back to source code from the formula-based abstraction
  - → CPT tracks the source-level origin of each clause of a path constraint

❷–③ Summary of Experiments and Findings

- Tested 15 implementations, including TLS & crypto libraries, SSH and IPSec software suites

Implementation (version)	Test Harness	Execution Time	Total Path (Accepting)	Implementation (version)	Test Harness	Execution Time	Total Path (Accepting)
axTLS (2.1.3)	*TH1*	01:42:14	1476 (6)	MatrixSSL (3.9.1) *CRL*	*TH1*	00:01:55	4574 (21)
	TH2	00:00:05	21 (21)		*TH2*	00:00:04	202 (61)
	TH3	00:00:10	21 (1)		*TH3*	00:00:07	350 (7)
BearSSL (0.4)	*TH1*	00:01:55	3563 (1)	mbedTLS (2.4.2)	*TH1*	00:14:56	51276 (1)
	TH2	00:00:06	42 (1)		*TH2*	00:00:03	26 (1)
	TH3	00:00:00	6 (1)		*TH3*	00:00:00	38 (1)
BoringSSL (3112)	*TH1*	00:06:09	3957 (1)	OpenSSH (7.7)	*TH1*	00:07:00	3768 (1)
	TH2	00:00:08	26 (1)		*TH2*	00:00:08	22 (1)
	TH3	00:00:00	6 (1)		*TH3*	00:00:00	2 (1)
Dropbear SSH (2017.75)	*TH1*	00:46:10	1260 (1)	OpenSSL (1.0.2l)	*TH1*	00:06:31	4008 (1)
	TH2	00:00:11	23 (1)		*TH2*	00:00:56	1148 (1)
	TH3	00:00:15	7 (1)		*TH3*	00:16:16	1673 (1)
GnuTLS (3.5.12)	*TH1*	00:01:35	570 (1)	Openswan (2.6.50)	*TH1*	00:01:07	378 (1)
	TH2	00:00:06	22 (1)		*TH2*	00:00:04	26 (1)
	TH3	00:00:01	4 (1)		*TH3*	00:00:00	6 (1)
LibreSSL (2.5.4)	*TH1*	00:10:27	4008 (1)	PuTTY (0.7)	*TH1*	00:03:22	3889 (1)
	TH2	00:01:40	1151 (1)		*TH2*	00:00:07	42 (1)
	TH3	00:25:45	1802 (1)		*TH3*	00:00:00	6 (1)
libtomcrypt (1.16)	*TH1*	00:01:13	2262 (3)	strongSwan (5.6.3)	*TH1*	00:01:32	2262 (3)
	TH2	00:00:11	805 (3)		*TH2*	00:16:36	15747 (3)
	TH3	00:04:49	7284 (1)		*TH3*	00:00:24	216 (6)
MatrixSSL (3.9.1) *Certificate*	*TH1*	00:01:54	4554 (1)	wolfSSL (3.11.0)	*TH1*	00:04:05	14316 (1)
	TH2	00:00:04	202 (1)		*TH2*	00:00:06	26 (1)
	TH3	00:00:22	939 (2)		*TH3*	00:00:00	6 (1)

❷–④ Notable Findings and Their Implications

- Various unwarranted leniciencies 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_rsasigkey' 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 anlayzing X.509 certificate validation and RSA signature verification