POSTER: *Why Crypto-detectors Fail:* A Systematic Evaluation of Cryptographic Misuse Detection Techniques

Amit Seal Ami*, Nathan Cooper*, Kaushal Kafle*, Kevin Moran†, Denys Poshyvanyk*, and Adwait Nadkarni*
*William & Mary, Williamsburg, VA, USA
{aami@, nacooper01@, kkafle@, denys@cs., nadkarni@cs.}wm.edu
†George Mason University, Fairfax, VA, USA
kpmoran@gmu.edu

Title: *Why Crypto-detectors Fail*: A Systematic Evaluation of Cryptographic Misuse Detection Techniques

Authors: Amit Seal Ami, Nathan Cooper, Kaushal Kafle, Kevin Moran, Denys Poshyvanyk, and Adwait Nadkarni

Venue: 2022 IEEE Symposium on Security and Privacy (S&P), Los Alamitos, CA, USA, 2022

DOI: 10.1109/SP46214.2022.00024

Full Reference: A. S. Ami, N. Cooper, K. Kafle, K. Moran, D. Poshyvanyk, and A. Nadkarni, "Why Crypto-detectors Fail: A Systematic Evaluation of Cryptographic Misuse Detection Techniques," in 2022 IEEE Symposium on Security and Privacy (S&P), Los Alamitos, CA, USA, 2022, pp. 397–414. Available at: https://doi.ieeecomputersociety.org/10.1109/SP46214.2022.00024

Abstract: The *correct* use of cryptography is central to ensuring data security in modern software systems. Hence, several academic and commercial static analysis tools have been developed for detecting and mitigating crypto-API misuse. While developers are optimistically adopting these crypto-API misuse detectors (or crypto-detectors) in their software development cycles, this momentum must be accompanied by a *rigorous understanding of their effectiveness at finding crypto-API misuse in practice*. This paper presents the `MASC` framework, which enables a systematic and data-driven evaluation of crypto-detectors using mutation testing. We ground `MASC` in a comprehensive view of the problem space by developing a data-driven taxonomy of existing crypto-API misuse, containing 105 misuse cases organized among nine semantic clusters. We develop 12 generalizable *usage-based mutation operators* and three *mutation scopes* that can expressively instantiate thousands of compilable variants of the misuse cases for thoroughly evaluating crypto-detectors. Using `MASC`, we evaluate *nine* major crypto-detectors and discover 19 unique, undocumented flaws that severely impact the ability of crypto-detectors to discover misuses in practice. We conclude with a discussion on the diverse perspectives that influence the design of crypto-detectors and future directions towards building security-focused crypto-detectors by design.

Pre-print: https://arxiv.org/pdf/2107.07065.pdf

Artifact: https://github.com/Secure-Platforms-Lab-W-M/MASC-Artifact

POSTER: *Why Crypto-detectors Fail*: A Systematic Evaluation of Cryptographic Misuse Detection Techniques

Amit Seal Ami[§], Nathan Cooper[§], Kaushal Kafle[§], Kevin Moran[†], Denys Poshyvanyk[§] and Adwait Nadkarni[§]

[§] William & Mary, [†] George Mason University

{aami@, ncooper01@., kkafle@., denys@cs, nadkarni@cs.} wm.edu, kpmoran@gmu.edu

https://amitsealami.com

WILLIAM & MARY CHARTERED 1693

Secure Platforms Lab https://spl-wm.github.io/

SEMERU http://www.cs.wm.edu/semeru/

Introduction

- **Correct use of cryptographic primitives is hard**.
- Security researchers make **Crypto API misuse-detectors** (*Crypto-Detectors*) to **prevent API misuse.**
- However, **we know very little regarding the actual effectiveness of crypto-detectors**.
- The **M**utation **A**nalysis for evaluating **S**tatic **C**rypto-API misuse detectors (MASC) framework can help **evaluate crypto-detectors by leveraging mutation testing**, i.e., by seeding mutants (*crypto API misuse*).

Challenges

- Must express (i.e., test with) relevant misuse cases across existing crypto-APIs, but, **cryptoAPIs are as vast as the primitives they enable**.
- Evaluation only using **misuse identified in the wild verbatim may not lead to robust analysis**, as it does not express the various usage patterns of such APIs.
- Efficiently creating and seeding **large numbers of compilable mutants without significant manual intervention is critical** for identifying flaws in crypto-detectors.

Crypto API Misuse Taxonomy

Compromising Client & Server Secrecy (20)

Small Key Size
- Using RSA with < 1024 bit key (7)
- Using RSA with < 2048 bit key (3) +
- Using RSA with 2048 bit private key (1)

Weak Algorithm
- Using RSA with CBC (1)
- Using RSA with no padding (2)
- Using RSA with PKCS1 padding (5)

Weak Certificate Management
- Improper certificate validation expiry check (2) ✔
- Trusting all certificates (3) ✔
- Missing certificate validation (3) ✔
- Improper following of a cert's chain of trust (1) ✔

Weak Hostname Management
- Allowing all hostnames (10) ✔
- Using Default hostname verifier (1) *

Weak SSL Protocol
- Using weak SSL context {`SSLContext.getInstance("SSL")`} (1)
- Using SSL and not using TLS as context (1)
- Using SSLV3 (1)
- Using SSLV2 (1)
- HMAC for TLS with SHA1 (1)
- Using CBC for SSL/TLS with AES (1) *
- Using TLS < v 1.2 (1)
- Using TLS < v 1.1 (3)

Compromising Randomness (5)

Misuse of Randomness
- Bad derivation of IV (file/text) (4) ✔
- Low entropy in key generation/ RNG (3)
- Using static seeds for Secure Random RNG (7)
- Not using Secure Pseudo RNG (7)
- Using Setseed (3)

Compromising Non-Repudiation (3)

Key Signing Misuses
- Low entropy with DSA (1)
- Low entropy with ECDSA (1)
- Using 1024 bit DSA (2)

Unclustered (6)
- Inscure pinning with ambiguous values
- Trusting Self-signed Certificates +
- Using unencrypted server socket
- Using unencrypted socket
- Using export quality ciphers
- Using stateless encryption

API/Program Specific Misuses (17)

API/Program Specific
- Apache `HTTPClient` no host verification (1)
- `Gnutls_certificate_verify_peers2` returns 0 when self signed certificate (1)
- Constant password for android keystore (2)
- JSSE checkTrusted method does not check identify if the algorithm field is null or empty string (1) ✔
- Android Webview incorrect certificate verification (2)
- `Java` defaults to `ECB` for encryption with "`AES`"
- Weberknecht does not have host verification (1)
- Using `DefaultHttpClient` (due to no TLSv1.2) (1)
- ignoring `onReceivedSSLError (3)`
- `SSLSocketFactory` without verifying Hostname (1)
- Reusing counter value in encryption (2)
- Apache `HttpHost` data allows mixed schemes (1)
- Using obsolete algorithm (11) ✔
- Storing sensitive data in Java `String` (3)
- Using `Socket` directly for connection (1)
- No `clearPassword` call after using `PBEKeySpec` (2)
- `PBEKeySpec` initialized without salt (2)

Compromising Secret Keys (12)

Secret Key Misuses
- Using low entropy seeds in key generation (1)
- Password Based Key Derivation Function (PBKDF) Using < SHA224 (1)
- Not using Salts while hashing password (1)
- PBKDF Using HMAC (1)
- PBKDF Using MD5 (3)
- PBKDF Using MD2 (2)
- IVs generated w/o random num generator (1) ✔
- Static IV (4) ✔
- Zeroed IV (2)
- Using hardcoded key / password (3)
- Using Constant Encryption Key (9)
- Using < 64bit salt for password (2)

Compromising Communication Secrecy with Intended Receiver (6)

Communication Secrecy Compromised
- Use of a key past its expiration date (1)
- HTTP and HTTPs mixing (3)
- Key Exchange without Entity Authentication (1)
- Improper Check for Certificate Revocation (1) ✔
- Improper Validation of Certificate with Host Mismatch (1) ✔
- Untrusted CA Signed Certificate (1) ✔

Compromising Secrecy of Cipher Text (26)

Insecure Key Size
- ECC < 224 bit (2)
- Using AES with < 128 bit key (1)
- Using RC2 with < 64 bits (1)

Insecure Number of Iterations/Cycles
- Using < 500 iterations for PBE (1)
- Using < 1000 iterations for PBE (6)#

Using Unsafe Mode
- Using ECB for symm. encryp. with AES (2) ✔
- Using AES with CBC for encryption with PKCS5Padding (1)
- Using Electronic Code Book Mode (ECB) for encryption (11) ✔
- Using AES with CBC for Encryption * (2)
- Using DESede with ECB (1)
- Using DES with CBC3 SHA (1)
- Using CBC without HMAC (1)
- Using 3DES with EDE CBC SHA (1)
- Using non-random IV in Cipher Block Chaining (CBC) for encryption (6)

Using Non-Random Salt
- Using constant salts for PBE (6)

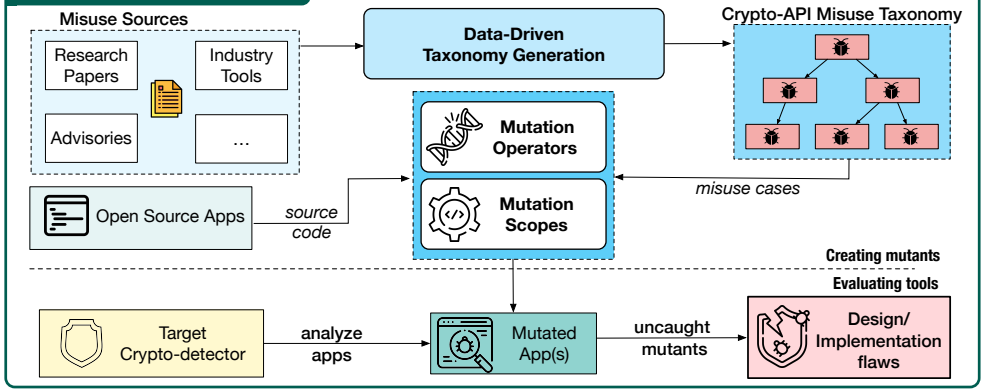
Unsafe Algorithm Usage
- Using RC2 for symmetric encryption (4)
- Using NullCipher to encrypt plain text (1)
- Using Blowfish Algorithm for Encryption (4)
- Using ESAPI Encryptor (1)
- Using 3DES/DESEDE for encryption (4)
- Using RC4 (3)
- Using IDEA Algorithm for Encryption (3)
- Using DES for encryption (8) ✔
- Using EXP1024 for ciphers (1)
- Using Seed Cipher (1)
- Using blowfish with less than 128 bit key (1)

Compromising Integrity through Improper Checksum Use (10)

Compromised Checksums
- Hashing credentials - MD5 (5) ✔
- Hashing Credentials - MD4
- Hashing Credentials - MD2
- Digital Signature Hashes - MD4
- Obsolete Hash Algorithm (7) ✔
- Hashing Credentials - SHA1
- Digital Signature Hashes - MD5 (5) ✔
- Using a custom MessageDigest instead of relying on the SHA-224 (1)
- Digital Signature Hashes - MD2 (4)
- Digital Signature Hashes - SHA1 (5)

* CBC is insecure in TLS/client-server context; + applicable in specific situations; some misuse are newer compared to other in same cluster, # PKCS5 suggestion based

The MASC Framework



Contributions

- **Comprehensive Crypto API Misuse Taxonomy** *Contains:* ***105 misuse cases*** *Covers:* **last 20 years**
- ***Usage-based Crypto-mutation Operators*** allow **expressive instantiation of Crypto misuse cases**
- **Evaluation of 9 major crypto-detectors** from industry, academia, and open-source revealed **19 previously unknown flaws**
- An impact study proved that all **these flaws have real consequences**
- Discussion with **Tool developers** revealed the **factors influencing the design and testing of current crypto-detectors**

Threat Model

Base Case: `Cipher.getInstance("DES");`

- Benign Developer, Accidental Misuse: `Cipher.getInstance("des");`
- Benign Developer, Harmful Fix: `Cipher.getInstance("des".toUpperCase());`
- Evasive Developer, Harmful fix: `Cipher.getInstance("AES".replace("A","D"));`

Flaw Classes

- *FC1*: String case mishandling
- *FC2*: Incorrect value resolution
- *FC3*: Incorrect resolution of complex inheritance and anonymous objects
- *FC4*: Insufficient analysis of generic conditions in extensible crypto-APIs
- *FC5*: Insufficient analysis of context-specific conditions in extensible crypto-APIs

Takeaway

Tool designs *are often based on* technique-centric perspectives, *whereas* ***we need*** **security centric evaluation**.

Some tools want to observe misuse instances frequently in the wild before addressing those!

"*…developers will write almost everything you can think of…*", hence **we should evaluate using more than trivial cases**.

Increasing importance of cyber-security through new legislations means **tools need to become more robust**.

MASC can help tools get better in detection by its rigorous, security-centric evaluation.

References

A. S. Ami, N. Cooper, K. Kafle, K. Moran, D. Poshyvanyk, and A. Nadkarni, "*Why Crypto-detectors Fail*: A Systematic Evaluation of Cryptographic Misuse Detection Techniques," in *2022 IEEE Symposium on Security and Privacy (S&P)*, Los Alamitos, CA, USA,

Artifact: https://github.com/Secure-Platforms-Lab-W-M/MASC-Artifact

This work is supported in part by the NSF-1815336, NSF-1815186, NSF-1955853 grants and CoVA CCI Dissertation Fellowship. Any opinions, findings, and conclusions expressed herein are the authors' and do not necessarily reflect those of the sponsors.

Coastal Virginia