
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


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.


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

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

**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 Mutation Analysis of evaluating Static Crypto-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,
- The cryptoAPIs are as vast as the primitives they enable
- Significant manual intervention is critical
- 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 Mutation Analysis of evaluating Static Crypto-API misuse detectors (MASC) framework can help evaluate crypto-detectors by leveraging mutation testing, i.e., by seeding mutants (crypto API misuse).

**Crypto API Misuse Taxonomy**

- Compromising Randomness (5)
  - Must express (i.e., test with) relevant misuse cases across existing crypto-APIs, but,
  - The cryptoAPIs are as vast as the primitives they enable
  - Significant manual intervention is critical
- Compromising Client & Server Secrecy (20)
- Weak Hostname Management
- Small Key Size
- Using 1024 bit DSA
  - Low entropy with DSA
- Using unencrypted socket
- Trusting Self-signed Certificates
- Insecure pinning with ambiguous values
- Not using Secure Pseudo RNG
  - Low entropy in key generation/RNG
- Bad derivation of IV (file/text)
- HMAC for TLS with SHA1
- Using SSLV2
- Using SSL and not using TLS as context
- Allowing all hostnames
- Using RSA with no padding
- Improper following of a cert’s chain of trust
- Trusting all certificates
- Using RSA with 2048 bit private key
- Using RSA with < 2048 bit key
- Using RSA with < 1024 bit key
- Improper Checksum Use (10)
- Using < 64bit salt for password
- Using hardcoded key/password
- PBKDF using HMAC
  - Using < 1000 iterations for PBE
  - Using < 500 iterations for PBE
- ECC < 224 bit
- Using < 64bit salt for password
- Using hardcoded key/password
- PBKDF using HMAC
  - Using < 1000 iterations for PBE
  - Using < 500 iterations for PBE
- ECC < 224 bit
- Improper Validation of Certificate with Host Mismatch
- Key Exchange without Entity Authentication
- Using Blowfish with less than 128 bit key
- Using Seed Cipher
- Using ESAPI Encryptor
- Using RC2 for symmetric encryption
- Using blowfish with less than 128 bit key
- Using Seed Cipher
- Using ESAPI Encryptor
- Using RC2 for symmetric encryption
- Using constant salts for PBE
  - Using < 1000 iterations for PBE
  - Using < 500 iterations for PBE
- Insecure pinning with ambiguous values
- Not using Secure Pseudo RNG
  - Low entropy in key generation/RNG
- Bad derivation of IV (file/text)
- HMAC for TLS with SHA1
- Using SSLV2
- Using SSL and not using TLS as context
- Allowing all hostnames
- Using RSA with no padding
- Improper following of a cert’s chain of trust
- Trusting all certificates
- Using RSA with 2048 bit private key
- Using RSA with < 2048 bit key
- Using RSA with < 1024 bit key
- Improper Checksum Use (10)
- Using < 64bit salt for password
- Using hardcoded key/password
- PBKDF using HMAC
  - Using < 1000 iterations for PBE
  - Using < 500 iterations for PBE
- ECC < 224 bit
- Improper Validation of Certificate with Host Mismatch
- Key Exchange without Entity Authentication
- Using Blowfish with less than 128 bit key
- Using Seed Cipher
- Using ESAPI Encryptor
- Using RC2 for symmetric encryption
- Using constant salts for PBE
  - Using < 1000 iterations for PBE
  - Using < 500 iterations for PBE
- Insecure pinning with ambiguous values
- Not using Secure Pseudo RNG
  - Low entropy in key generation/RNG
- Bad derivation of IV (file/text)
- HMAC for TLS with SHA1
- Using SSLV2
- Using SSL and not using TLS as context
- Allowing all hostnames
- Using RSA with no padding
- Improper following of a cert’s chain of trust
- Trusting all certificates
- Using RSA with 2048 bit private key
- Using RSA with < 2048 bit key
- Using RSA with < 1024 bit key
- Improper Checksum Use (10)
- Using < 64bit salt for password
- Using hardcoded key/password
- PBKDF using HMAC
  - Using < 1000 iterations for PBE
  - Using < 500 iterations for PBE
- ECC < 224 bit
- Improper Validation of Certificate with Host Mismatch
- Key Exchange without Entity Authentication
- Using Blowfish with less than 128 bit key
- Using Seed Cipher
- Using ESAPI Encryptor
- Using RC2 for symmetric encryption
- Using constant salts for PBE
  - Using < 1000 iterations for PBE
  - Using < 500 iterations for PBE
- Insecure pinning with ambiguous values
- Not using Secure Pseudo RNG
  - Low entropy in key generation/RNG
- Bad derivation of IV (file/text)
- HMAC for TLS with SHA1
- Using SSLV2
- Using SSL and not using TLS as context
- Allowing all hostnames
- Using RSA with no padding
- Improper following of a cert’s chain of trust
- Trusting all certificates
- Using RSA with 2048 bit private key
- Using RSA with < 2048 bit key
- Using RSA with < 1024 bit key

**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!

Increasing importance of cybersecurity 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**


**Artifact**

https://github.com/StateCompliance-Labs/W-M/M-ACE-Artifact