Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Security Audit-Report Safing Jess Library 01.2020

Cure53, Dr.-Ing. M. Heiderich, Dr. N. Kobeissi

Index

Introduction

“Jess is a cryptographic library and cli tool that focuses on usability and freedom.”

From https://github.com/safing/jess/

This report documents the findings of a security assessment targeting the Safing Jess library. Carried out by Cure53 in January 2020, this project looked at the Safing Jess complex as a cryptographic library and a command line interface tool focused on usability and freedom. Therefore, the assessment entailed an audit of the Safing Jess cryptography, as well as a more general security review of its premise.

Resources-wise, this project involved two senior-testers of the Cure53 team and was executed in January 2020, taking a total of five person-days to complete. The methodological approach entailed white-box testing since Cure53 had access to the public source available on GitHub as open source (OSS). In addition, the testers were informed about the *commits* relevant for the audit and supplied with further documentation and material. A detailed and very thorough briefing was held by Cure53 and the Safing maintainer, facilitating understanding of the scope.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

In addition to the kick-off, the audit was accompanied by several fruitful discussions, allowing the team to have an in-depth familiarity with the scope and, consequently, leveraging a very good coverage level of this audit. The discussions also covered the findings Cure53 spotted. The testers and the maintainer of the Safing Jess shared their views on possible remediations and conducted fix verifications for some of the findings that were addressed right after being live-reported. All in all, this audit was a textbook example of a productive and effective assessment, excellently managed by the Safing Jess maintainer in cooperation with Cure53.

As for the findings, the Cure53 team discovered five issues, four classified as security vulnerabilities and one representing a general weakness. One problem was given the *Critical* severity rating with another item marked as *High* in terms of risk. All other issues were flagged as *Medium* threats. The bug noted as *Critical* had already been fixed and Cure53 verified this fix, confirming that the problem no longer exists on the scope.

In the following sections, the report will first reiterate the areas featured in the test's scope in more detail, noting the relevant GitHub URLs and commit ID. The discussions then move on to dedicated, chronologically discussed tickets, which present the discoveries one-by-one. Alongside technical aspects like PoCs, Cure53 furnishes mitigation advice and comments on the status of fixes where applicable. The report closes with *Conclusions* in which Cure53 summarizes this early 2020 project and voices a verdict about the security premise of the investigated Safing Jess complex.

Note: *All issues were addressed with fixes and all fixes were inspected by Cure53 and verified as working. Fix notes were added to each ticket, the rest of the report was left untouched.*

Scope

- **Source Code Audits & Cryptography Review against Safing Jess Library**
  - https://github.com/safing/jess
  - Relevant commit is `648d16d1cc8185ed3704373e43c84a6ab4315498`
    - https://github.com/safing/jess/commit/648d16d1cc8185ed3704373e43c84a6ab4315498

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. *SAF-01-001*) for the purpose of facilitating any future follow-up correspondence.

SAF-01-001 Crypto: Secure key deletion ineffective *(Medium)*

***Note:** This issue has been discussed with the Safing team and a fix has been confirmed by the Cure53 team.*

In many instances, Jess calls *helper* functions that appear to be designed as a way to "burn" cryptographic keys and similarly sensitive material from the device's memory in order to prevent recovery. These "*burn*" functions are called frequently throughout the cryptographic stack of Jess, including locations in the *Signet, RSA, X25519* and *NIST* components.

However, the tactic used in order to achieve key erasure is likely ineffective, due to the limitations of the Go's garbage collection-based memory management model[1]. Furthermore, it does not attempt to target residual values, such as intermediate computations, which thus may remain in memory.

Affected Files:
helper.go
tools/ecdh/nist.go

Affected Code:

```
// Burn gets rid of the given []byte slice(s).
func Burn(data ...[]byte) {
    for _, slice := range data {
        for i := 0; i < len(slice); i++ {
            slice[i] = 0xFF
        }
    }
}

[...]

func (ec *NistCurve) BurnKey(signet tools.SignetInt) error {
    pubKey := signet.PublicKey()
```

[1] https://github.com/golang/go/issues/21865

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
    privKey := signet.PrivateKey()
    if pubKey != nil {
        point, ok := pubKey.(*ecdh.Point)
        if ok {
            point.X.Set(big.NewInt(0))
            point.Y.Set(big.NewInt(0))
        }
    }
    if privKey != nil {
        data, ok := privKey.([]byte)
        if ok {
            ec.Helper().Burn(data)
        }
    }
    return nil
}
```

It is recommended to not communicate any guarantees regarding secure key erasure to users. Cure53 proposes to instead wait until this feature is officially provided in some way or another by the Go programming language. Third-party implementations of such features on top of the Go's memory management model are unlikely to yield effective results and should be avoided.

SAF-01-002 Crypto: Password KDF vulnerable to GPU/ASIC attacks *(Medium)*

Note: *This issue has been discussed with the Safing team and a fix has been confirmed by the Cure53 team.*

It was observed that Jess uses the *PBKDF2* password hashing function for the purpose of strengthening user-passwords before they are employed for deriving encryption and authentication of symmetric keys. In this realm, 20 000 rounds of *PBKDF2* are used, which gives the password hashing functionality a relatively high security margin.[2] However, two weaknesses inherent to *PBKDF2* remain:

- **Vulnerability of speeding-up via optimizations.** Research shows that *PBKDF2* can be accelerated by 50% or more through different optimizations relying on hashing through precomputed values[3] or other acceleration attacks[4].
- **Vulnerability of speeding-up via GPU attacks.** *PBKDF2* is vulnerable to fast brute-force via the usage of hardware that can perform the underlying hash function with high throughput. This can range from commodity GPUs to custom

[2] https://cryptosense.com/blog/parameter-choice-for-pbkdf2/
[3] https://link.springer.com/chapter/10.1007/978-3-319-26823-1_9
[4] https://www.usenix.org/conference/woot16/workshop-program/presentation/ruddick

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

ASICs, with such hardware being prevalent due to its usefulness for cryptocurrency mining.

It is recommended to instead use a password hashing function that is resistant to these attacks, such as *scrypt*[5]. Since Jess deals with generating cipher-text that is vulnerable to offline attacks, it is recommended to adopt the parameters generally used for offline *scrypt* key derivation (i.e. $N=2^{20}$, r=8, p=1).

SAF-01-003 Crypto: Secure channel protocol weaknesses *(High)*

***Note:** This issue has been discussed with the Safing team and a fix has been confirmed by the Cure53 team.*

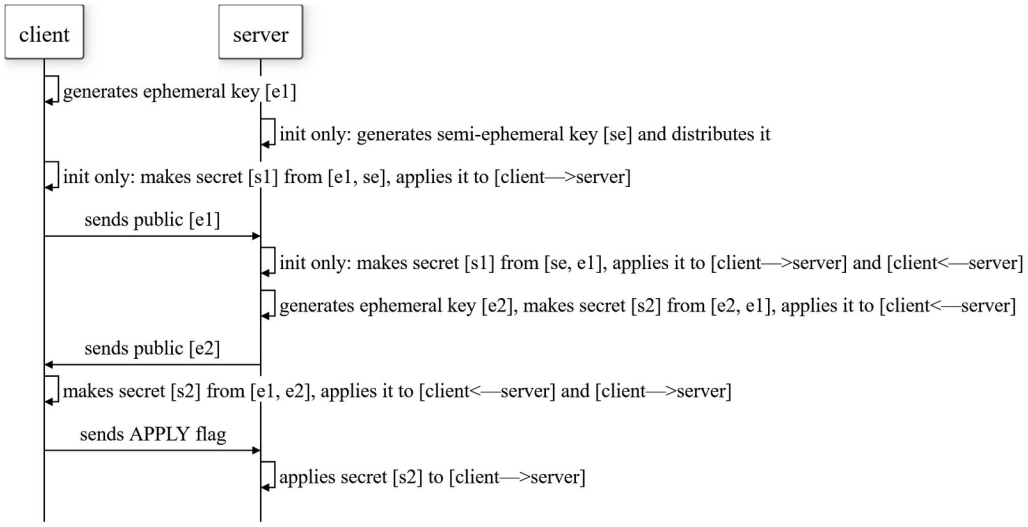

Fig.: Jess' Diffie-Hellman secure channel protocol, based on the NK[6] Noise Protocol Framework Handshake Pattern.

The secure channel protocol deployed by Jess - in both Diffie-Hellman and Key Encapsulation variants, was modeled and evaluated using the Verifpal automated protocol analysis software[7]. The results were the following:

```
Result • authentication? Client -> Server: enc1: When the following values are
mutated by the attacker:
            ge1 → G^nil (originally G^e1)
            enc1, sent by Client and resolving to AEAD_ENC(G^se^e1, msg1,
nil), is successfully used in primitive AEAD_DEC(G^nil^se, AEAD_ENC(G^se^e1,
```

[5] https://link.springer.com/chapter/10.1007/978-3-319-56617-7_2
[6] https://noiseexplorer.com/patterns/NK/
[7] https://verifpal.com

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
msg1, nil), nil)? in Server's state, despite it being vulnerable to tampering by
Attacker.

   Result • confidentiality? msg2: When the following values are mutated by the
attacker:
             ge1 → G^nil (originally G^e1)
             enc1 → AEAD_ENC(G^nil^se, nil, nil) (originally AEAD_ENC(G^nil^se,
nil, nil))
             msg2 is obtained by the attacker as msg2
```

These results outline that due to the lack of long-term public key being provided by the client, it is impossible for the Jess secure channel protocol to provide client authentication. Therefore, the first message sent from the client to the server will never benefit from authentication. In case of an early session compromise by an active attacker, responses from the server can always be decrypted and manipulated by adversaries. Since no real fix to this problem can omit providing long-term keys for clients, it is recommended to instead wait until a handshake is completed before communicating sensitive payloads. This revised approach may render the attack somewhat more expensive to carry out for a potential active attacker on the network.

SAF-01-004 Crypto: Key management/encryption with 1-byte key (*Critical*)

Note: *This issue has been discussed with the Safing team and a fix has been confirmed by the Cure53 team.*

When Jess is initialized with an envelope that is set to use symmetric keys for encryption, it will always mandate a symmetric key size being provided manually by the user with the *--symkeysize* command-line option. It was found that passing *--symkeysize=1* Jess would result in an envelope configuration with a 1-byte key. Said key could then successfully be used for authenticated encryption and decryption. The resulting encryption would be trivially easy to break due to the effective key space of only 256 possible keys.

Affected File:
session.go

Affected Code:

```
err := e.LoopSecrets(SignetSchemeKey, func(signet *Signet) error {
        totalSignetsSeen++
        keySourceAvailable = true
        // FIXME
        return nil
    })
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

This issue has been reported to Safing, which responded by adding the appropriate *session.calcAndCheckSecurityLevel* check to the code mentioned above. By doing so, the necessary checks to avoid small key sizes can be seen as implemented successfully.

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

SAF-01-005 Crypto: Unnecessary configurability considered dangerous *(Medium)*

***Note:** This issue has been discussed with the Safing team and a fix has been confirmed by the Cure53 team.*

Whenever a new envelope configuration is instantiated in Jess, the user is required to configure both a set of security requirements and a cipher-suite. The security requirements rely on standard definitions: payload confidentiality and integrity as well as sender and recipient authentication for messages. The proposed cipher-suites include key derivation functions *(HKDF-SHA2*), symmetric ciphers (*ChaCha20-Poly1305, AES-CTR, AES-GCM, Salsa20*), *RSA* for public-key encryption, and asymmetric elliptic-curve cryptography (*P224, P256, P384, P521, Curve25519*) for Diffie-Hellman and signing operations.

```
? Select to edit Tools          HKDF(SHA2-256), CHACHA20-POLY1305
? Select tools:  [Use arrows to move, space to select, type to filter]
  [x]  HKDF(SHA2-256)       KeyDerivation                                          Hugo Krawczyk, 2010                        RFC 5869
  [x]  CHACHA20-POLY1305    IntegratedCipher    128 b/s                            Daniel J. Bernstein, 2008 and 2005         RFC 7539
  [ ]  AES128-CTR           Cipher              128 b/s                            Vincent Rijmen and Joan Daemen, 1998       aka Rijndael, FIPS 197
  [ ]  AES128-GCM           IntegratedCipher    128 b/s                            Vincent Rijmen and Joan Daemen, 1998       aka Rijndael, FIPS 197
  [ ]  AES192-CTR           Cipher              192 b/s                            Vincent Rijmen and Joan Daemen, 1998       aka Rijndael, FIPS 197
  [ ]  AES192-GCM           IntegratedCipher    192 b/s                            Vincent Rijmen and Joan Daemen, 1998       aka Rijndael, FIPS 197
  [ ]  AES256-CTR           Cipher              256 b/s                            Vincent Rijmen and Joan Daemen, 1998       aka Rijndael, FIPS 197
> [ ]  AES256-GCM           IntegratedCipher    256 b/s                            Vincent Rijmen and Joan Daemen, 1998       aka Rijndael, FIPS 197
  [ ]  ECDH-P224            KeyExchange         112 b/s                            NIST, 2009                                 FIPS 186
  [ ]  ECDH-P256            KeyExchange         128 b/s                            NIST, 2009                                 FIPS 186
  [ ]  ECDH-P384            KeyExchange         192 b/s                            NIST, 2009                                 FIPS 186
  [ ]  ECDH-P521            KeyExchange         256 b/s                            NIST, 2009                                 FIPS 186
  [ ]  ECDH-X25519          KeyExchange         128 b/s                            Daniel J. Bernstein, 2005
  [ ]  Ed25519              Signing             128 b/s                            Daniel J. Bernstein, 2011
  [ ]  HMAC                 MAC                                                    Mihir Bellare et al., 1996                 RFC 2104, FIPS 198

  [ ]  PBKDF2-SHA2-256      PassDerivation                                         Burt Kaliski, RSA Laboratories, 2000/2017  PKCS #5 v2.1, RFC 8018
  [ ]  POLY1305             MAC                 128 b/s                            Daniel J. Bernstein, 2005                  RFC 7539
  [ ]  RSA-OAEP             KeyEncapsulation    dynamic b/s (set manually via --seclevel)
  [ ]  RSA-PSS              Signing             dynamic b/s (set manually via --seclevel)  Mihir Bellare, Phillip Rogaway, 1998  RFC 8017
> [ ]  SALSA20              Cipher              256 b/s                            Daniel J. Bernstein, 2007
```

Fig.: Ciphers offered during envelope configuration, independent from security requirements.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

The following issues are present with this arrangement:

- **Cipher-suites can be manually chosen, independently from security requirements.** After setting the security requirements, users are expected to also figure out which combination of primitives can satisfy these requirements. From there, they are also expected to choose suitable primitives from a pool that contains many primitives performing the same functionality, with some offering drawbacks over the others. In the event that an inadequate cipher-suite is chosen by the user, Jess simply exits.
- **Cipher-suite malleability has no benefits and presents drawbacks.** Besides the potential for user error, there is simply no benefit to custom cipher-suites. Especially because Jess does not perform public key encryption, RSA can be removed entirely and replaced with Diffie-Hellman key agreement, which benefits from more modern primitives such as *Curve25519* (superior both in terms of security and performance to the *NIST* curves also offered by Jess). *AES*-based symmetric ciphers, unauthenticated ciphers and *HMAC* functions can also be ruled out entirely and replaced with a single modern authenticated symmetric encryption primitive such as *ChaCha20-Poly1305.*

Protocol versioning can be adopted to ensure that a set of optimal cipher-suites is chosen for each version. Next, version upgrades could occur only if it is necessary to make changes to the optimal cipher-suite, concurrently to the upgrade period of the previous version becoming deprecated and removed from circulation.

After discussing the issue with Safing, decisions have been made about optimal cipher-suite for version 1 of the Jess secure channel protocols. This was done on the basis of security, performance and compatibility considerations:

```
key_v1:                                    HKDF(BLAKE2b-256), CHACHA20-POLY1305
pw_v1:    SCRYPT,                          HKDF(BLAKE2b-256), CHACHA20-POLY1305
rcpt_v1:  ECDH-X25519,                     HKDF(BLAKE2b-256), CHACHA20-POLY1305
sign_v1:  Ed25519(BLAKE2b-256)
v1:       ECDH-X25519, Ed25519(BLAKE2b-256),HKDF(BLAKE2b-256), CHACHA20-POLY1305
wire_v1:  ECDH-X25519,                     HKDF(BLAKE2b-256), CHACHA20-POLY1305
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Conclusions

Despite spotting five issues containing problems of *Critical* and *High* severities, this January 2020 assessment of the Safing Jess complex concludes on a positive note. After spending five days on the scope and acquiring a complete coverage, two members of the Cure53 team can attest to the proper cryptographic premise and commend the knowledge and engagement of the maintainer responsible for security of the Safing Jess compound.

All of the proposed cryptographic constructions were evaluated, with the Go code found to be clean, readable and well-documented. The accompanying protocols are well-specified, even though there is a fundamental limitation in the Go memory management model. This signifies some minor setbacks for the project, as documented in SAF-01-001. Next, it was found that the password-based key derivation primitive was obsolete due to its vulnerability to optimization attacks as well as GPU and ASIC attacks. A replacement primitive was recommended in SAF-01-002.

Moving on, certain weaknesses in the Jess secure channel protocol were documented via the analysis performed with the Verifpal automated cryptographic protocol analyzer (see SAF-01-003). Most concerning vulnerability marked as *Critical* had been spotted next. This problem relates to Jess allowing users to encrypt and decrypt with 1-byte keys, rendering all encrypted payloads trivially easy to break. This is documented in SAF-01-004. While no immediate security concerns were identified as a result of SAF-01-005, Cure53 discusses how the requirements that the users can specify when manually selecting cipher-suites for Jess can work independently from their security goals. This could result in problems down the road, as well as signifies major inconvenience for usability and deployment of Jess.

To conclude, Cure53 is impressed with the quality of collaboration, especially since all issues have been thoroughly discussed with Safing and examined in relation to possible remediation. With all fixes already verified, Cure53 can only finalized this project by concluding that the Safing Jess complex has certainly become safer and more secure as a result of this January 2020 assessment. The Safing Jess complex is definitely on the right track in terms of security.

Cure53 would like to thank Daniel Hovie and Raphael Fiedler from the Safing team for their excellent project coordination, support and assistance, both before and during this assignment.