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

Security-Audit-Report Slant PrivEOS 07.2019

Cure53, Dr.-Ing. M. Heiderich, BSc. F. Fäßler, Prof. N. Kobeissi, N. Hippert

Index

Introduction

"Storing personal data poses huge challenges for dApps. PrivEOS features a key management solution which makes operating with personal data easy for dApp developers and users alike."

From https://www.slant.li/priveos/

This report documents the findings of a security assessment targeting the Slant PrivEOS complex. Carried out by Cure53 in July 2019, this project entailed a source code audit and a review of the cryptographic premise. The assessment focused on evaluating the Slant PrivEOS node network and the related complex of client, library and server software. Only two security-relevant discoveries were made on the Slant PrivEOS scope during this project.

Resources-wise, a team of four testers from the Cure53 team spent a total of twelve person-days on the scope. It should be mentioned that the scope spanned six private Github code repositories, which Cure53 was granted access to by the Slant team. In addition, Cure53 was briefed in detail before the assessment and also got documentation and material to study before starting the reviews.

The project, which was split into two Work Packages, progressed in a timely and efficient fashion. The communications were done using Slack: the Slant team gave Cure53 access to their workspace and created a dedicated private channel for discussing the test tasks. All exchanges were prompt and feedback received by Cure53 from Slant was comprehensive. As for the two findings, one was classified as an actual vulnerability with impact evaluated as "*Low*". This problem was located in a third-party library. The second finding encompasses a general weakness with "*Medium*" severity, pointing to a rather good result overall. While the first item relates to a cryptographic flaw that could lead to a minor info leak, the other can be linked to the underlying Node architecture and missing rate-limiting.

In the following sections, the report will first present the scope and the two WPs in more detail. The discussions then move on to tickets which shed light on the two discoveries one-by-one. The report closes with a conclusion in which Cure53 summarizes the project and issues a verdict about the tested scope. Conclusions about the security and privacy posture of the Slant PrivEOS complex are supplied in the final section of this document.

Scope

- **PrivEOS Node Network**
  - **WP1**: PrivEOS Node Network-, Client- and Library-Sources
  - **WP2**: PrivEOS Test-Environment and locally-/Cure53-ran Setup
  - A detailed scope document was shared with Cure53
- **Sources were made available for Cure53**
  - https://github.com/rawrat/privEOS
  - https://github.com/rawrat/priveos-encryption-service
  - https://github.com/rawrat/priveos-automation
  - https://github.com/rawrat/priveos-client
  - https://github.com/grempe/secrets.js
  - https://github.com/rawrat/eosjs-ecc-priveos

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. *SLA-01-001*) for the purpose of facilitating any future follow-up correspondence.

SLA-01-001 Crypto: Checksum reveals information about encryption key *(Low)*

It was found that an encryption function in a third-party library used by PrivEOS includes a hash of the *AES* encryption key with every ciphertext transmitted over the network (as a *"checksum"* value). While this does not necessarily lead to an attack on the confidentiality of the plain-text, it still allows an observer to, at the very least, determine whether different payloads were encrypted with the same or different encryption keys. From there, an adversary could profile traffic.

Furthermore, the current approach constitutes neither a valid nor secure authenticated encryption scheme.

***Note:** PrivEOS does not use the vulnerable encryption function, but already employs a different authenticated encryption scheme via the NaCL cryptobox construction.*

Affected File:
eosjs-ecc-priveos/src/aes.js

Affected Code:
```
function crypt_shared_secret(S, box, encrypt) {
  let nonce, checksum, message
  if(encrypt) {
    nonce = uniqueNonce()
    message = box
  } else {
    ({nonce, checksum, message} = deserialize(box))
  }
  if (!Buffer.isBuffer(message)) {
      if (typeof message !== 'string')
          throw new TypeError('message should be buffer or string')
      message = new Buffer(message, 'binary')
  }
  assert(Buffer.isBuffer(S), "S is not a buffer")
  assert(Buffer.isBuffer(nonce), "nonce is not a buffer")
```

```
    const ekey_length = S.length + nonce.length
    let ebuf = Buffer.concat([nonce, S], ekey_length)
    const encryption_key = hash.sha512(ebuf)
    const iv = encryption_key.slice(32, 56)
    const key = encryption_key.slice(0, 32)
    // check is first 64 bit of sha256 hash
    let check = hash.sha256(encryption_key)
    check = check.slice(0, 8)
    if (checksum) {
        if (!check.equals(checksum)) {
            throw new Error('Invalid checksum')
        }
        return cryptoJsDecrypt(message, key, iv)
    } else {
        message = cryptoJsEncrypt(message, key, iv)
        return serialize(nonce, check, message)
    }
}
```

A better way for obtaining a *"checksum"* value is to use an authenticated cipher, which produces an "*authentication tag*" over the cipher-text. Validating this authentication tag on decryption allows the decryptor to ascertain the integrity of the cipher-text and the plain-text. Popular primitives that offer authenticated encryption include *ChaCha20-Poly1305* and *AES-GCM*. It is recommended for the third-party library to adopt these ciphers instead. The custom "*checksum"* value should be removed.

Fix Notes: *As recommended, the issue was addressed by the Slant team by removing the checksum value. The fix was verified by Cure53.*

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

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.

SLA-01-002 Node: No rate-limiting protection *(Medium)*

The nodes are publicly listening with an HTTP server to handle various requests like reading or storing files. Due to the design of the nodes, which have to interact with the blockchain and send requests to other nodes, a single request is heavily amplified in terms of resource-usage. This makes the server susceptible to Denial-of-Service.

Affected Files:
/privEOS/broker
/privEOS/kms/kms.js

Code Excerpt:
The following function serves as an example and is being called by a request to */broker/read/*. It issues calls out to the blockchain, database and additional HTTP requests.

```
async function broker_read(req, res) {
        // [...]
        await Backend.get_accessgrant_trace(chain, dappcontract, requester, file, 
txid, timeout_seconds)
        const store_trace = await Backend.get_store_trace(chain, dappcontract, 
file, timeout_seconds)
        // [...]
        const nodes = await get_nodes(chain, payload)

        const promises = nodes.map(async node => {
                log.debug("nodes.map: ", node)
                const url = new URL('/kms/read/', node.url).href
                const res = await axios.post(url, {
                                // [...]
                        })
                return res.data
        })
        log.debug("payload.threshold: ", payload.threshold)
        const shares = await Promise.some(promises, payload.threshold)
        // const shares = data.map(x => x.data)
        // [...]
        res.send({shares, user_key: payload.user_key})
}
```

As can be seen, DoS can easily become an issue for PrivEOS. Thus, as a first step, the nodes should be heavily rate-limited by IP. To prevent some accidental DoS from replayed requests, nodes could secondly serve known responses from a cache. Because of the heavy amplification, it could also be considered to implement some proof of work[1] to make a request for the client as resource-intensive as it is for the server.

Fix Notes: *Slant has mitigated the issue by adding strong throttling settings based on IP for both services. A white-list was added to exclude trusted nodes as well.*

Conclusions

This assessment of the Slant PrivEOS project concludes with a positive verdict. After spending twelve days on the scope in July 2019, four members of the Cure53 team confirm that the Slant PrivEOS system is mostly vulnerability-free. Spotting only two items, both with rather limited implications, points to a very stable and robust nature of the tested Slant PrivEOS scope.

It can be clarified that PrivEOS is based on the cryptographic algorithm called Shamir's Secret Sharing and has been implemented with an EOS smart contract. It is being distributed with nodes running a slim NodeJS application. The PrivEOS whitepaper reviewed as part of this assessment demonstrates that the cryptographic functionality in place mirrors the documented premise. The *Argon2 KDF* is used for key derivation with suitable parameters, ultimately translating to no security flaws identifiable in this realm. Further, the *Zxcvbn* library is being used in order to help guide the user towards a safer password/passphrase choices.

Next up, the aforementioned implementation of Shamir's Secret Sharing was audited and proven to correctly adhere to its specification. Similarly good verdict was reached about *distribution* of Shamir's Secret Sharing. A single and minor crypto-related issue as detected in the way in which "*checksums"* were generated for cipher-texts. This was documented in SLA-01-001.

The NodeJS application *broker* and *kms* can be considered microservices and are kept small and functional. Not only does the lack of complexity makes code reviews more efficient, but it also lowers the likelihood of accidentally introducing bugs. It is also worth noting that the PrivEOS team seems to be aware of the typical NodeJS pitfalls such as prototype pollution, avoiding and mitigating them successfully. However, the services interact with multiple parts of the system - such as the blockchain or other nodes. This means a single request from the outside will consume considerable resources on the

[1] https://www.npmjs.com/package/proof-of-work

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

nodes and can lead to a relatively easy Denial-of-Service (see SLA-01-002). While this could impact the accessibility and stability of the system, it has no direct security impact.

The EOS smart contract is the heart of the system and ensures the integrity of the data being requested and stored. In contrast to other blockchains found with smart contracts, the EOS smart contracts can be upgraded. This practice, in itself, lowers the number of the associated risks tremendously. Moreover, it appears that PrivEOS stays up to date with the EOS Contract Development Kit improvements and, for example, makes us of the *on_notify* modifier instead of implementing a customized dispatcher. One point to note is that the contract code is rather extensive when compared to other parts of the system. As no documentation about this functionality existed during the audit, some consideration can be given to developing it.

The latter also affects other parts of the system. The Slant PrivEOS whitepaper reviewed by Cure53 contains mostly high-level explanations. This means that grasping the system as a whole requires a lot of time and knowledge. Cure53 recommends to invest into proper documentation and specifications of the protocol. This will have a positive impact on the development and security by improving future security audits and reviews coming from the community.

The PrivEOS team should be praised for being available for questions and discussions via Slack, as well as for providing prompt feedback. Despite the lack of certain specifications, this allowed Cure53 to effectively review all parts of the system. Cure53 would also like to highlight that PrivEOS initiated the review during the development phase. In other words, the assessment took place when the system has already been functional but was not yet considered complete. Thus, Cure53 recommends to have another audit before going live. For now, the results of this July 2019 review make Cure53 confident about the PrivEOS complex being on the right path from a security standpoint. .

Cure53 would like to thank Fabian Frank and Angelo Laub from the Slant AG team for their excellent project coordination, support and assistance, both before and during this assignment.