Pentest-Report TextSecure Extension 11.2015

Cure53, Dr.-Ing. Mario Heiderich, Dr. Jonas Magazinius, Fabian Fäßler, Jann Horn

Index

Introduction

“Signal-Browser is a chrome packaged app that links with the Signal-iOS or Signal-Android client already installed on your phone.”

From https://github.com/WhisperSystems/Signal-Browser

This source code audit and a penetration test against the Signal-Browser extension was carried out by four testers from Cure53. The team members have worked on the project for a total of six days. In terms of the scope of the test the focus was placed on a specially created tag available in the public Github repository for the extension. The test covered injection attacks, cryptographic implementations, security issues specific to browser extensions, as well evaluated robustness and transport security. The underlying cryptographic library - *libaxolotl* - was explicitly beyond the scope within this particular assignment.

Minor weaknesses aside, the tested extension presented itself very robust and secure. No serious issues were identified during this audit, even though significant efforts were invested into the process. Not only is the code clean and well-readable, but also common JavaScript pitfalls, extension security issues and cryptographic flaws have been successfully avoided. In further praise to the extension’s developers, the sources need to be described as easy to audit. While some initial setup and connection issues were noted at the beginning, the overall impression was that the extension was easy to build and no major interruptions or difficulties were experienced.

Only six general weaknesses and one actual security issue of medium severity were spotted in the realm of this test and audit. This undeniably indicates an exceptionally rare and positive result, especially for a penetration test against a browser-driven encryption tool. This rather short presentation of the penetration test results is a reflection of a praiseworthy quality of the code and security-conscious development of the software.

Test Scope

- **Signal-Browser Cure53 Tag**
  - https://github.com/WhisperSystems/Signal-Browser/tree/c53

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in a 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. *WSB-01-001*) for the purpose of facilitating any future follow-up correspondence.

WSB-01-004 bower.json lists dependencies with insecure protocols *(Medium)*

The *bower.json* file lists dependencies with insecure protocols. This particularly concerns:

```
"qrcode": "git://github.com/davidshimjs/qrcodejs.git#1c78ccd71",
"cryptojs": "svn+http://crypto-js.googlecode.com/svn/#~3.1.2",
"libphonenumber-api": "git://github.com/codedust/libphonenumber-api",
```

Neither the git protocol nor the SVN over HTTP has any kind of integrity protection, meaning that they just exchange data in plaintext. If someone attempts to build TextSecure with up-to-date dependencies while a MitM attacker[1] is tampering with his Internet connection, this issue could lead to a compromised TextSecure build at the very least.

To solve this problem it is recommended to use secure protocols instead. For Github HTTPS (or SSH if the user has an SSH key that is known to Github) can be employed. For SVN repositories on Google Code, SVN over HTTPS should be used.

[1] https://en.wikipedia.org/wiki/Man-in-the-middle_attack

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.

WSB-01-001 No URL validation on getAttachment() / putAttachment() (*Low*)

When the client attempts to download or upload an attachment, a server-supplied URL is used for the HTTP PUT or GET request without any validation (see *getAttachment()*[2] / *putAttachment()*[3]). Although browsers permit every website to blindly perform GET requests, cross-origin PUT requests are not allowed.

The potential impact of this issue seems very low, partly because the encryption would prevent an attacker from controlling a PUT body or tricking the client into treating another server's response as a valid attachment.

If it can be achieved without too much effort, it is recommended to force the use of a fixed hostname or, alternatively, a hostname that ends with a specific suffix.

WSB-01-002 No JSON schema validation for API call results *(Low)*

The code in *libtextsecure/api.js* parses server responses using *JSON.parse()*[4] without validating the format of the received data. This behavior can potentially allow the server to leave the client in an inconsistent state, doing so by triggering an error in the middle of an operation that would normally be atomic.

ConfirmCode()[5] was the only API call where a crafted server response could leave behind an inconsistent state found during the test. By sending a response with *deviceId={toString:1}*, the server could cause a crash in the call to *setNumberAndDeviceId()* in *createAccount()*, leaving behind a partially initialized storage and a wrong value in *this.server.username*.

It is recommended to validate the structure of the received JSON responses immediately.

[2] https://github.com/WhisperSystems/Signal-Browser/blob/master/libtextsecure/api.js#L260
[3] https://github.com/WhisperSystems/Signal-Browser/blob/master/libtextsecure/api.js#L273
[4] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse
[5] https://github.com/WhisperSystems/Signal-Browser/blob/master/libtextsecure/api.js#L166

WSB-01-003 Debug logs are public gists *(Medium)*

In *console.post()* in *js/debugLog.js,* debug log gists are explicitly requested to be public. It is recommended to consider marking them as private instead, which would help prevent attackers without any kind of special access from seeing the last three digits of everyone listed as those that the user has communicated with. This is especially concerning since public gists are publicly searchable:

Example URL:
https://gist.github.com/search?utf8=%E2%9C%93&q=anon%3Atrue+whispersystems+%22delivery+receipt%22+filename%3AdebugLog.txt&ref=searchresults

Non-public gists have a longer identifier that prevents brute-force attacks against the identifier. They are not indexed for public searches yet are still readable by anyone who knows the link. The only disadvantage of using them in this scenario is that they are more difficult to type out manually. Conversely, as long as users can copy-paste the links, this should not matter.

WSB-01-005 Denial of Service from users communicating with Signal *(Low)*

This issue is not generally related to the Chrome extension but nevertheless came up during testing. When a user wants to use Signal, he or she has to validate their phone number by requesting a token via an SMS or a phone call. Subsequent token requests will invalidate the previous ones. This means that an attacker can prevent a user from signing up for Signal by constantly requesting new tokens, even though it would be very noisy.

Yet another denial of service vector might be possible when the token is guessed. It seems that an attacker can request a new authentication token and start guessing the six digits. If the attacker finds the valid token, the identity is replaced and the victim cannot use Signal anymore. At least for manual tests the authentication code was not invalidated after more than 20 attempts. However, the Chrome extension code hints at possible raid-limit error codes that might mitigate this attack. Even when the token would be invalidated after a certain amount of attempts, it could be used by an attacker to prevent new users from signing up for Signal. It is thus suggested to increase the length of the authentication token.

WSB-01-006 Bimap implementation uses objects as maps *(Low)*

The (currently unused) Bimap implementation in *js/bimap.js* uses objects as maps without ensuring that the *__proto__* property[6] is not written to. This could lead to an unexpected program behavior due to the fact that the properties of the assigned value later also become the inherited properties of the map object. In addition, attempting to delete the *__proto__* property will silently fail.

[6] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/proto

It is recommended to consider and implement one of the following:

- Remove the Bimap implementation if it is no longer used;
- Create map objects using `Object.create(null)`, which removes special properties;
- Use a name prefix that prevents all keys from having a special name.

WSB-01-007 MAC Check uses potentially insecure String Comparison *(Low)*

The function *verifyMAC()*[7] in *libtextsecure/crypto.js* calls *isEqual()* to verify the MAC of a websocket message or attachment. *isEqual()* ends up comparing strings using the == operator. While nothing suggests that the minimal timing differences introduced by near-matches can be exploited in practice, the *String::Equals* implementation in the V8 runtime would probably fall through to *String::SlowEquals()*, which first compares the first characters of the strings, then uses *memcmp()*, which is not completely timing-safe either.

It is recommended to use the usual approach instead, meaning that the bytes of the computed MAC shall be XORed with the bytes of the expected MAC, having the results ORed together next, and the final result getting compared with zero.

Conclusion

The Signal-Browser extension surely presents itself strong and robust from a security standpoint. No major issues were identified, speaking volumes to the fact that the developers of the extension avoid the majority of JavaScript and browser-extension security pitfalls. The final result is a well-written and clean code-base.

This report described all identified issues and gives recommendations on remediation and fixes. Once those are deployed and verified, the browser extension can, as far as security aspect is concerned, be fully recommended for a broader use. It should nevertheless be mentioned that the range of features at this stage of the development was rather small and a re-test should be considered if and when the tool becomes more powerful and richer in features and usability enhancements.

Cure53 would like to thank Lilia Kai of the Whisper Systems for their excellent project coordination, support and assistance. The testing team has consistently benefited from this help both before and during this assignment. We would like to further express our gratitude to the Open Technology Fund in Washington D.C., USA, for generously funding this and other penetration test projects and enabling us to publish the results.

[7] https://github.com/WhisperSystems/Signal-Browser/blob/master/libtextsecure/crypto.js#L28