Wire Security Review – Phase 2 – Web, Calling for Wire Swiss GmbH

Final Report

2018-03-07

FOR PUBLIC RELEASE

Contents

1 Summary

This report documents the findings identified in the Wire *web application* and *calling* components.

From a total of seven vulnerabilities that were discovered during the test, one was classified as high severity, five as medium, and one as low. No observations without a direct security impact have been made. The issue with the highest severity was an outdated JavaScript framework that could potentially allow injection attacks against the application. All identified vulnerabilities except for a DoS issue require preconditions to be exploited successfully.

The main Wire repositories covered during the review are wire-webapp and wire-audio-video-signaling.

This review does not cover:

- code of audio/video codecs and the WebRTC implementation;
- code of third-party dependencies and template engines.

Wire has patched or mitigated the discovered vulnerabilities after they were reported.

The work was performed between March 7th and July 5th, 2017, by Jean-Philippe Aumasson (Kudelski Security) and Markus Vervier (X41 D-Sec GmbH). A total of 7.5 person-days were spent, with most of the time spent reviewing code and investigating potential security issues.

2 Findings

This security review report covers several components of the Wire suite of applications:

- The web application (section 2.1)
- Signalling components of the calling protocol (section 2.2)

The work was performed between March and July 2017, with a total of 4.5 person-days (Kudelski Security) and 3 person-days (X41 D-Sec). The code subject to this review was available in the repositories wire-webapp (revision ef30665e6824047a60af00405b10758897db46fa) and wire-audio-video-signaling (revision 99799d12f49fcddf7252748b8f3dceff3625d0fe).

2.1 Web Application

We reviewed the Wire web application (available in the repository `https://github.com/wireapp/wire-webapp`, revision ef30665e6824047a60af00405b10758897db46fa) for common web application vulnerabilities. This app is the base for the standalone desktop clients based on Electron[1], and is also available for usage in normal browsers at `https://app.wire.com`. This review was focused on the web application components and does not consider Electron and the potential attack surface that it might provide.

[1] `https://electron.atom.io/`

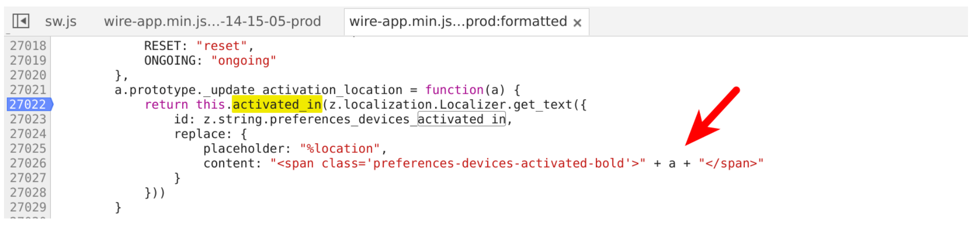
2.1.1 WIRE-P2-00: Possible XSS Via Device Location

Severity:	*MEDIUM*
CWE:	*123*

2.1.1.1 Description

The Wire web application will display the location where a user's device was first activated and does not filter the resolved names for markup.

Location data is resolved using the Google Maps API[2]. Given numeric location coordinates it will resolve them to a name. This name is then used unfiltered in HTML markup as displayed in figure 2.1.


sw.js
wire-app.min.js...-14-15-05-prod
wire-app.min.js...prod:formatted

```
            RESET: "reset",
            ONGOING: "ongoing"
        },
        a.prototype._update_activation_location = function(a) {
            return this.activated_in(z.localization.Localizer.get_text({
                id: z.string.preferences_devices_activated_in,
                replace: {
                    placeholder: "%location",
                    content: "<span class='preferences-devices-activated-bold'>" + a + "</span>"
                }
            }))
        }
```

Figure 2.1: Unfiltered Location Data

This markup is later used in the user preferences to fill a template using Knockout[3] templates:

```
<div class="label-xs" data-bind="html: activated_in"></div>
```

As seen above the `data-bind` attribute has the `html:` option enabled which means that markup will be inserted unfiltered into the page. An attacker able to control data on a compromised backend or the Google Maps API might insert malicious markup into the *wire-webapp* page context.

Wire fixed [4] this issue by replacing the `html` attribute with the `text` attribute.

[2] https://developers.google.com/maps/
[3] https://knockoutjs.com
[4] https://github.com/wireapp/wire-webapp/commit/a28c2cb6d3efd49ea61bbdbd67f3af734753bd8a

2.1.1.2 Solution Advice

We recommend to use the `text:` option of the data-bind attribute to prevent unfiltered data entering the web page. Additionally all untrusted user data should be enclosed in sandboxed iFrames with an opaque origin in order to mitigate potential flaws in the Knockout template engine.

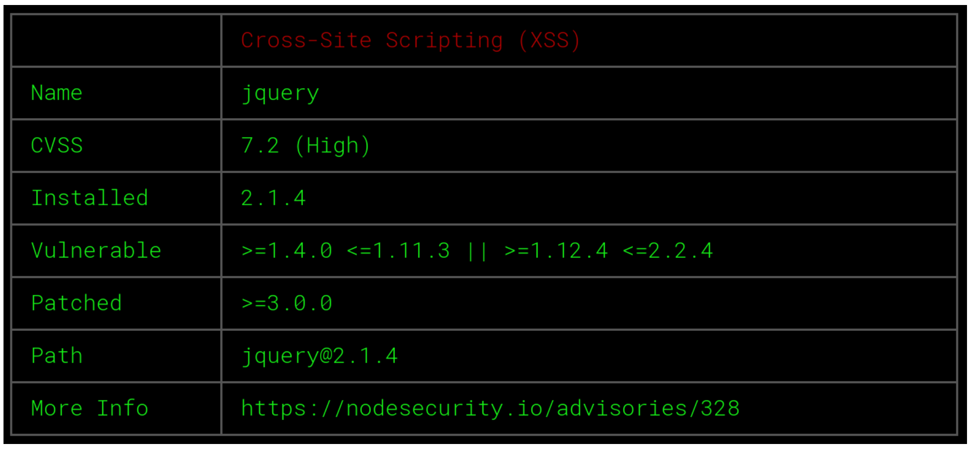
2.1.2 WIRE-P2-01: Outdated JavaScript-Component With Known Vulnerabilities

Severity:	*HIGH*
CWE:	*937*

2.1.2.1 Description

The *antiscroll-2* package used by the wire web application depends on a version of *jQuery* with known vulnerabilities.

Using the tool `nsp`[5], the dependency on jQuery version 2.1.4 was detected as shown in figure 2.2.



```
             Cross-Site Scripting (XSS)
Name         jquery
CVSS         7.2 (High)
Installed    2.1.4
Vulnerable   >=1.4.0 <=1.11.3 || >=1.12.4 <=2.2.4
Patched      >=3.0.0
Path         jquery@2.1.4
More Info    https://nodesecurity.io/advisories/328
```

Figure 2.2: Vulnerable Package.

XSS attacks might be conducted when AJAX calls are made to untrusted sites. During the time given we could not verify if this was the case. It is still recommended to upgrade all packages with security vulnerabilities in order to mitigate potential technical risks.

Wire has patched the version of jQuery in use against this attack in parallel to this review after an internal evaluation. It recently migrated [6] [7] to a newer version of jQuery that

[5] `https://nodesecurity.io/opensource`

[6] `https://github.com/wireapp/wire-webapp/commit/4d016d5b1bca75fb39708502af7f19f4f8effa36`

[7] `https://github.com/wireapp/antiscroll-2/commit/eb1b6a21aca0b2d35fd73231662d801f3f66d265`

has no known vulnerabilities at the time of writing.

2.1.2.2 Solution Advice

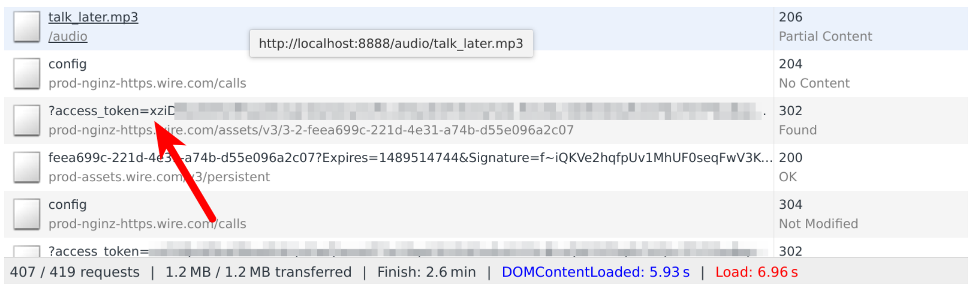
We suggest to update the packages to recent versions without known vulnerabilities. If fixed versions are not available for specific packages, it is recommended to disable them temporarily until a fix is implemented. Additionally an automated process should be implemented that regularly checks for packages with known vulnerabilities.

2.1.3 WIRE-P2-02: Access Token Leak Via URL Parameter

Severity:	*MEDIUM*
CWE:	*123*

2.1.3.1 Description

The access token used to authenticate against the Wire backend is contained in the URL parameters of different HTTP requests. For example it is attached to the URL used to download encrypted assets such as images as displayed in figure 2.3. The token could for example potentially be cached in proxy server logs. Wire stated that the scenarios where this is exploitable are not part of their threat model and the issue is risk accepted.



```
talk_later.mp3                                                                   206
/audio              http://localhost:8888/audio/talk_later.mp3                   Partial Content
config                                                                           204
prod-nginz-https.wire.com/calls                                                  No Content
?access_token=xziD[redacted]                                                     302
prod-nginz-https.wire.com/assets/v3/3-2-feea699c-221d-4e31-a74b-d55e096a2c07      Found
feea699c-221d-4e31-a74b-d55e096a2c07?Expires=1489514744&Signature=f~iQKVe2hqfpUv1MhUF0seqFwV3K... 200
prod-assets.wire.com/v3/persistent                                               OK
config                                                                           304
prod-nginz-https.wire.com/calls                                                  Not Modified
?access_token=[redacted]                                                         302
407 / 419 requests | 1.2 MB / 1.2 MB transferred | Finish: 2.6 min | DOMContentLoaded: 5.93 s | Load: 6.96 s
```

Figure 2.3: Access Token Leak Via URL Parameter

2.1.3.2 Solution Advice

Sensitive tokens and credentials should not be stored in URL parameters. It is recommended to use dedicated authorization request headers or HTTP-POST parameters to transmit them.

2.1.4 WIRE-P2-03: Lax Parsing of Base64 Strings

Severity:	*LOW*
CWE:	*228*

2.1.4.1 Description

The base64 decoding function ***z.util.base64_to_array*** tolerates invalid base64 strings. This function is used to decode encrypted messages in the following code (***z.cryptography.CryptographyRepository.decrypt_event***):

```
    session_id = @_construct_session_id event.from, event.data.sender
    ciphertext = z.util.base64_to_array(event.data.text or event.data.key).buffer
    return @cryptobox.decrypt(session_id, ciphertext).then (plaintext) ->
 ↪   z.proto.GenericMessage.decode plaintext
```

It follows that an encoded message modified in transit (accidentally or maliciously) may be accepted as valid, if the invalid base64 string decodes to the correct array of bytes. In the following test invalid characters are tolerated:

```
z.util.array_to_base64(z.util.base64_to_array("d2lyZQ==*** ??? ***"))
"d2lyZQ=="
```

As RFC 3548 explains, “Non alphabet characters may be exploited as a "covert channel", where non-protocol data can be sent for nefarious purposes.” Although MIME for example tolerates non-base64 characters, it is generally safer to reject them.

A malicious peer may use this lax decoding to send large invalid strings that the base64 decode as short valid byte arrays.

Also the base64 decoder tolerates invalid base64 encodings as for example single-character or padding-less strings.

The root cause is the use of ***sodium.from_base64***, a base64 decoder part of libsodium.js that will accept malformed base64 strings (first by stripping non-base64 characters, and then by lazily parsing the data received, decoding even invalid strings).

However, this behavior will not allow attackers to forge valid data with malicious content, because the underlying data is encrypted and authenticated. The security risk is therefore low.

The issue was patched [8] by libsodium and Wire migrated [9] to the patched version of libsodium while refactoring.

2.1.4.2 Solution Advice

A strict base64 decoding should be used, similar to that in JavaScript's ***atob*** function.

[8] `https://github.com/jedisct1/libsodium.js/issues/89`

[9] `https://github.com/wireapp/wire-web-packages/commit/4a3c0f0fcf6cedc10a37e9309a57ac45fe2493e3`

2.2 Calling

We reviewed core components of the new calling mechanism, namely

- The cryptographic protocol leveraging the Proteus session in order to establish an SRTP master key, and its implementation in the *wire-audio-video-signaling* repository, revsion 99799d12f49fcddf7252748b8f3dceff3625d0fe.
- Key implementation aspects of the calling protocol, such as the parsing of signaling messages. For example, we fuzz-tested the decoder of SDP descriptors (using afl).

The review focused on the changes introduced in the new calling mechanism, and does not cover all components nor all potential risks associated to the Wire calls. For example, we did not perform an in-depth review of WebRTC security as used in Wire, nor did we assess risks associated with traffic analysis or with the ICE functionality.

All of these issues have been addressed by Wire, and we reviewed the relevant patches to confirm their effectiveness.

Full disclosure: We contributed to the design of the key establishment protocol, in particular we recommended the key derivation mechanism involved in the channel binding.

2.2.1 WIRE-P2-08: Denial-of-Service Via Malformed JSON

Severity:	*MEDIUM*
CWE:	*730*

2.2.1.1 Description

Signalling messages are JSON-formatted strings sent by calling peers to one another. They are decoded by a function ***jzon_decode*** that relies on the *libre* library's ***json_decode***.

However, this function will enter a virtually infinite loop when fed certain values, allowing a peer to DoS the other peer. For example, the (valid) JSON string `"[1e10000000000000000000]"`, although it just encodes the number 1, will force the decoder to enter a loop of 2^{63} iterations, in the following function from *decode.c*:

```
static inline uint64_t mypower10(uint64_t e)
{
        uint64_t i, n = 1;

        for (i=0; i<e; i++)
                n *= 10;

        return n;
}
```

We could not verify the exploitability of this property, since we did not find any string that crashes the decoder (based on test cases from e.g. `https://github.com/nst/JSONTestSuite`). However, we believe that a more thorough investigation will allow attackers to create malicious strings triggering this infinite loop.

The issue was fixed[10] upstream and adopted by Wire.

[10] `https://github.com/creytiv/re/commit/f8b24b462dc9cf95242d450d82e7fadbf4b0d0fd#diff-49720871786ef0465bcfdb6f67547ff0`

2.2.1.2 Solution Advice

To eliminate this class of DoS strings, the decoder should implement a less naive exponentiation algorithm (such as basic square-and-multiply, running in linear time rather than exponential time in the exponent's bit-length).

2.2.2 WIRE-P2-09: Late Zeroization of Keys

Severity:	*MEDIUM*
CWE:	*730*

The function ***kase_get_sessionkeys*** computes SRTP master keys for the transmitted and received streams using libsodium's new API:

```
if (is_client) {
                if (crypto_kx_client_session_keys(session_rx, session_tx,
                                                  kase->publickey,
                                                  kase->secretkey,
                                                  publickey_remote) != 0) {

                        warning("kase: Suspicious server public key\n");
                        return EPROTO;
                }
        }
        else {
                if (crypto_kx_server_session_keys(session_rx, session_tx,
                                                  kase->publickey,
                                                  kase->secretkey,
                                                  publickey_remote) != 0) {

                        warning("kase: Suspicious client public key\n");
                        return EPROTO;
                }
        }
```

The peer's ephemeral Diffie-Hellman secret *kase->secretkey* is then no longer used. If all goes well, the private key is zeroized when the mediaflow destructor is called, at the end of the session (through ***mem_deref(mf->kase)*** which will call ***kase_destructor***). But if the session crashes before the mediaflow session is properly closed, then the private key will remain in memory. An attacker running unprivileged processes on the same machine may therefore recover the keys after reusing the memory previously allocated to storing the private keys. This attack scenario is nonetheless unlikely here.

Wire fixed[11] this issue by zeroizing the secret key at the end of the session key generation function.

2.2.2.1 Solution Advice

It would be safer to zeroize the private key just after it's used, namely right after the above session keys computations.

[11]https://github.com/wireapp/wire-audio-video-signaling/blob/cea426e906d2c819d76dd7949858a9758f7b8f2a/src/kase/kase.c#L142

2.2.3 WIRE-P2-10: Outdated libvpx allowing remote DoS

Severity:	*MEDIUM*
CWE:	*730*

2.2.3.1 Description

The libvpx video codec currently used is version 1.6.0, which includes a DoS vulnerability that allows for remote DoS on Android Mediaserver (CVE-2017-0393). Version 1.6.1 is not vulnerable, and integrates the following patch: `https://android.googlesource.com/platform/external/libvpx/+/6886e8e0a9db2dbad723dc37a548233e004b33bc`

Potential impact on Wire is also a remote Denial-of-Service (DoS) as well.

The library is included in repository *`wire-audio-video-signalling`* as submodule. When building the application the fixed version 1.6.1 of libvpx is automatically fetched and used.

2.2.3.2 Solution Advice

Update to libvpx 1.6.1.

3 About

Kudelski Security
route de Genève, 22-24
1033 Cheseaux-sur-Lausanne
Switzerland

Kudelski Security is an innovative, independent Swiss provider of tailored cyber and media security solutions to enterprises and public sector institutions. Our team of security experts delivers end-to-end consulting, technology, managed services, and threat intelligence to help organizations build and run successful security programs. Our global reach and cyber solutions focus is reinforced by key international partnerships. Kudelski Security is a division of Kudelski Group.

For more information, please visit `https://www.kudelskisecurity.com`.

X41 D-Sec GmbH
Dennewartstr. 25-27
D-52068 Aachen
Germany

X41 D-Sec is an expert provider for application security services. Having extensive industry experience and expertise in the area of information security, a strong core security team of world class security experts enables **X41 D-Sec** to perform premium security services.

Fields of expertise in the area of application security are security centered code reviews, binary reverse engineering and vulnerability discovery. Custom research and a IT security consulting and support services are core competencies of **X41 D-Sec**.

For more information, please visit `https://www.x41-dsec.de`.