Fine penetration tests for fine websites

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

Pentest-Report Delta Chat Mail-Server Template & Libraries 02.2023

Cure53, Dr.-Ing. M. Heiderich, Dipl.-Ing. A. Inführ, MSc. H. Moesl

Index

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

Introduction

"Starting in 2017, merlinux is facilitating and sponsoring the development of Delta.Chat, a unique server-less, end-to-end encrypting messenger that interoperates with the existing massively federated e-mail infrastructure."

From https://merlinux.eu/

This report details the scope, results, and conclusory summaries of a penetration test and source code audit against the merlinux-curated and mailcow-based mail-server template for Delta Chat usage, the related setup, and several connected Delta Chat Rust libraries. The security assessment was requested by merlinux GmbH in January 2023 and initiated by Cure53 in February 2023, namely in CW07. A total of ten days were allocated to fulfill this project's coverage expectations.

The testing conducted for this audit was divided into two distinct Work Packages (WPs) for execution efficiency, as follows:

- **WP1**: White-box tests & audits against merlinux mail-server template & setup
- **WP2**: White-box tests & audits against merlinux Rust libraries

Cure53 was granted access to all relevant Github repositories. The methodology selected was white-box and a team comprising three skillmatched senior testers was assigned to the project's preparation, execution, and finalization. All preparatory measures were completed in February 2023, namely in CW06, to ensure that the audit could proceed without hindrance or delay. Communications were facilitated via a dedicated Delta Chat channel, wherein all participatory personnel from both parties were invited to partake throughout the test preparations and discussions. In light of this, communications proceeded smoothly on the whole. The scope was well-prepared and transparent, no noteworthy roadblocks were encountered throughout testing, and cross-team queries remained minimal as a result.

Cure53 gave frequent status updates concerning the test and any related findings, whilst simultaneously offering prompt queries and receiving efficient, effective answers from the maintainers. Live reporting was requested and subsequently conducted for all tickets assigned a higher severity rating, specifically *Medium* in this case. Concerning the findings, the testing team achieved widespread coverage over the WP1 and WP2 scope items, identifying a total of eight. Four of the findings were categorized as security vulnerabilities, whilst the remaining four were deemed general weaknesses with lower exploitation potential. Considering the moderate volume of findings detected, the testing team garnered an exceedingly positive impression of the underlying codebase.

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

Leveraging Rust as the primary programming language was deemed an astute choice, as the underlying framework offers robust security paradigms by default. Supplementary integrations using *forbid(unsafe_code)* force developers into a path that offers far less attack surface to classic memory corruption issues. Nevertheless, Cure53 observed ample opportunities for hardening, as corroborated by the tickets outlined in this report. MER-01-004 demonstrates a classic example whereby different components of the setup employ alternate parsing mechanisms to check email addresses. This underlying mismatch thus allows for the creation of permanent accounts that cannot be deleted.

In addition, two path traversal issues were detected and documented in tickets MER-01-005 and MER-01-001 respectively. The former only allows directory traversal to almost arbitrary mailcow API endpoints, though the latter may have been exploitable via attack vectors on Python Pillow. However, the checks ensuring the referenced paths exist render this issue unexploitable at present. The remaining weaknesses mostly pertain to oversights in connected dependencies that should be kept up-to-date, as well as some erroneous behaviors in the underlying TLS configuration. Notably, these incur insignificant risk and minimal impact against the audited scope's resilient security posture on the whole.

The report will now shed more light on the scope and testing setup as well as provide a comprehensive breakdown of the available materials. Subsequently, the report will list all findings identified in chronological order, starting with the detected vulnerabilities and followed by the general weaknesses unearthed. Each finding will be accompanied by a technical description and Proof of Concepts (PoCs) where applicable, plus any relevant mitigatory or preventative advice to action.

In summation, the report will finalize with a conclusion in which the Cure53 team will elaborate on the impressions gained toward the general security posture of the merlinux-curated and mailcow-based mail-server template for Delta Chat usage, related setup, and several connected Delta Chat Rust libraries, giving high-level hardening advice where applicable.

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

Scope

- **Pentests & source code audits against merlinux-curated mail-server setup**
  - **WP1**: White-box tests & audits against merlinux mail-server template & setup
    - **Relevant source-code repositories**:
      - https://github.com/mailcow/mailcow-dockerized
      - https://github.com/deltachat/mailadm
    - **Test instance:**
      - http://afterca.re/
    - **SSH credentials:**
      - U: Audit
  - **WP2**: White-box tests & audits against related Delta Chat Rust libraries
    - **Relevant source-code repositories**:
      - **Delta Chat core:**
        - https://github.com/deltachat/deltachat-core-rust/tree/master/src
      - **async-smtp:**
        - https://github.com/async-email/async-smtp
      - **async-imap:**
        - https://github.com/async-email/async-imap
      - **async-native-TLS:**
        - https://github.com/async-email/async-native-tls
      - **Native-TLS:**
        - https://github.com/sfackler/rust-native-tls/
      - **fast-socks5:**
        - https://crates.io/crates/fast-socks5

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

Identified Vulnerabilities

The following section lists all vulnerabilities and implementation issues identified during the testing period. Notably, findings are cited in chronological order rather than by degree of impact, with the severity rank offered in brackets following the title heading for each vulnerability. Furthermore, all tickets are given a unique identifier (e.g., *MER-01-001*) to facilitate any future follow-up correspondence.

MER-01-003 WP2: Unencrypted SQLite database storage *(Low)*

The Delta Chat core library written in Rust underwent dynamic testing and a source code review, revealing that both the Delta Chat desktop and mobile apps utilize a SQLite database to store sensitive information such as email addresses, passwords, OAUTH2 tokens, and configuration data. Whilst the Android app implements encryption using a passphrase for this database, the desktop app stores the database unencrypted in the user's home directory. By leveraging access to the database, an attacker would gain all necessary information to assume full control over the used email account.

Given that the customer is already aware of this issue and that the assessment in question is not specifically focused on the Delta Chat desktop app, this issue's severity impact was appropriately downgraded to *Low*.

Steps to reproduce:

1. Open the database using the tool *sqlite3*, as follows:

```
sqlite3 <HOME>/.config/DeltaChat/accounts/<account_nr>/dc.db
```

2. Note that fetching all data from the *config* table yields the following result:

Shell excerpt:

```
sqlite> select * from config;
1|dbversion|96
2|addr|wmvnq@testrun.org
3|mail_pw|4/<redacted>

14|configured_mail_pw|4/<redacted>
[...]
20|configured_send_pw|4/<redacted>
21|configured_send_security|2

48|notify_about_wrong_pw|1
```

To mitigate this issue, Cure53 advises implementing a passphrase-protected SQLite database for the Delta Chat desktop app, as achieved for the Android app integration.

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

One optimal solution in this respect would be to utilize PBKDF2 or another similar mechanism to derive the passphrase from a master password, which the user would enter when launching the desktop app.

MER-01-004 WP1: Domain validation and creation mismatch *(Medium)*

During the assessment of the *mailadm* application, the observation was made that the specified address is modified before being passed to the internal mailcow API during the creation of a new user account. This behavior allows one to specify an address that passes all checks and is accepted by the mailcow API but is stored differently in the mailadm local database. This mismatch means that the account is never deleted, since mailadm stores a different account address than that sent to mailcow and the deletion process is solely based on locally stored addresses.

Initially, the mailadm simply checks whether the specified address string ends with the configured domain. As the domain setting does not contain the @ character, this check itself permits arbitrary domains. Additionally, as demonstrated below, the application subsequently divides the address and solely utilizes the first and second parts, therefore ignoring any additional @<*characters*> proportions present in the address string.

Example command:

```
/add-user anemail@afterca.re@randomstuff@ABCDafterca.re abcd1234 <yourToken>
```

Address sent to mailcow:
anemail@afterca.re

Affected file:
mailadm-master/src/mailadm/conn.py

Affected code:

```
def add_email_account(self, token_info, addr=None, password=None):
[...]
    else:
    if not addr.endswith(self.config.mail_domain):
    raise ValueError(
    "email {!r} is not on domain {!r}".format(addr, self.config.mail_domain)
)
[...]
# seems that everything is fine so far, so let's invoke mailcow:
self.get_mailcow_connection().add_user_mailcow(addr, password, token_info.name)
```

Affected file:
mailadm-master/src/mailadm/mailcow.py

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

Affected code:

```
def add_user_mailcow(self, addr, password, token, quota=0):
[...]
    url = self.mailcow_endpoint + "add/mailbox"
    payload = {
        "local_part": addr.split("@")[0],
        "domain": addr.split("@")[1],
        "quota": quota,
        "password": password,
        "password2": password,
        "active": True,
        "force_pw_update": False,
        "tls_enforce_in": False,
        "tls_enforce_out": False,
        "tags": ["mailadm:" + token],
    }
    result = r.post(url, json=payload, headers=self.auth, timeout=HTTP_TIMEOUT)
```

To mitigate this issue, Cure53 recommends strictly validating a specified email address before further processing, for which the Python language offers libraries that already implement this functionality. Additionally, the developer team should adapt the *endswith* call so that the @ character is included at the very least.

MER-01-005 WP1: Path traversal in mailcow API request *(Low)*

During the user account creation process, the specified address is passed to the mailcow API to verify whether it exists already. Here, the confirmation was made that the address is used in the HTTP path without any strict validation, which permits traversing the HTTP path up and specifying an arbitrary mailcow API endpoint. Since the response is not returned to a malicious user, nor do the exposed HTTP GET endpoints incur any specific security issue, this flaw's severity impact was appropriately downgraded to *Low*.

PoC commands:

```
/add-user goingtostay@afterca.re@/../../differentEndpoint#@afterca.re abcd1234
<yourToken>
```

Created URL:
http://mailcow.host/api/v1/get/goingtostay@afterca.re@/../../differentEndpoint#@afterca.re

Normalized URL:
http://mailcow.host/api/v1/differentEndppiont

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

Affected file:
mailadm-master/src/mailadm/mailcow.py

Affected code:
```
def get_user(self, addr):
    """HTTP Request to get a specific mailcow user (not only mailadm-generated
ones)."""
    url = self.mailcow_endpoint + "get/mailbox/" + addr
    result = r.get(url, headers=self.auth, timeout=HTTP_TIMEOUT)
    json = result.json()
```

To mitigate this issue, Cure53 advises ensuring that a specified email address is strictly validated before any further processing, as stipulated in ticket MER-01-004. Generally speaking, a benign email address must not contain characters such as question marks (*?*), hashes (*#*), dots (.), and forward slashes (*/*), which should therefore be rejected by the mailadm application.

MER-01-008 WP1: Shell command injection in mailcow *Sync Job* feature *(High)*

The mailcow UI, which is used by the administrator during the original setup phase, can also be utilized by standard users to configure certain account settings. During the assessment, the observation was made that the *Sync Job* feature - which can be made available to standard users - suffers from a shell command injection. A malicious user can abuse this vulnerability to obtain shell access to the Docker container running *dovecot*.

The *imapsync* Perl script implements all the necessary functionality for this feature, including the XOAUTH2 authentication mechanism. This code path creates a shell command to call *openssl*. However, since different parts of the specified user password are included without any validation, one can simply include and execute additional shell commands. Notably, the default ACL for a newly-created mailcow account does not include the necessary permission, though the testing team confirmed that specific instances exist whereby users assign this permission.

Affected file:
https://github.com/mailcow/mailcow-dockerized/blob/master/data/Dockerfiles/dovecot/imapsync

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

Affected code:

```
sub xoauth2
{
[...]
if( $imap->Password =~ /^(.*\.json)$/x )
{ [...] }
else
{
        # Get iss (service account address), keyfile name, and keypassword if
        necessary
        ( $iss, my $keyfile, my $keypass ) = $imap->Password =~ /([\-\d\w\@\.]+);
        ([a-zA-Z0-9 \_\-\.\/]+);?(.*)?/x ;

        # Assume key password is google default if not provided
        $keypass = 'notasecret' if not $keypass;

        $sync->{ debug } and myprint( "Service account: $iss\nKey file: $keyfile\
        nKey password: $keypass\n");

        # Get private key from p12 file (would be better in perl...)
        $key = `openssl pkcs12 -in "$keyfile" -nodes -nocerts -passin pass:$key-
        pass -nomacver`;
```

Steps to reproduce:

1. Log in at the mailcow GUI with a *Sync Job* permission user.
2. Click *Create a new job sync*.
3. Use the following information:
4. *Host: imap.google.com*
5. *Port: 993*
6. *Encryption: SSL*
7. *Username: test*
8. *Password: 123;abc;r ;$(id);*
9. *Custom parameters: --authmech1=XOAUTH*
10. Check the *Active* checkbox only and create the sync job.
11. Wait for the sync job to fail and open the associated links.
12. Observe that the injected ID command will be executed.

Log output:

```
directory:../crypto/bio/bss_file.c:69:fopen('abc','rb')
140026593842496:error:2006D080:BIO routines:BIO_new_file:no such file:../
crypto/bio/bss_file.c:76:
sh: 1: uid=65534(nobody): not found
sh: 1: -nomacver: not found
```

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

To mitigate this issue, Cure53 highly advises reporting this vulnerability to the official maintainers to ensure an adequate resolution can be rolled out. One temporary solution here would be to remove the XOAUTH2 function until this fix is correctly implemented. As a defense-in-depth mechanism, the documentation could be extended to include a warning that stipulates assigning additional permissions to mailcow users and the potential security impact.

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

Miscellaneous Issues

This section covers any and all noteworthy findings that did not incur an exploit but may assist an attacker in successfully achieving malicious objectives in the future. Most of these results are vulnerable code snippets that did not provide an easy method by which to be called. Conclusively, whilst a vulnerability is present, an exploit may not always be possible.

MER-01-001 WP1: Path traversal in QR code generation via token name *(Low)*

After an administrator creates a token, a QR code can be generated in tandem that can be leveraged by users to obtain valid credentials. During this process, the specified token name is used to craft a filename to store the QR code image in the local file system. Since the token name is not validated, an attacker can traverse the local file system and potentially overwrite other images.

However, this sink was deemed unexploitable because the utilized PIL library verifies that all specified folders in the path exist before applying any normalization, which breaks any successful traversal functionality.

PoC command:

```
/add-token /../mytesttokenv2 1d 1
```

Exception:

```
Exception [Errno 2] No such file or directory: 'docker-data/dcaccount-after-
ca.re-/../mytesttokenv2.png'
```

Affected file:

src/mailadm/commands.py

Affected code:

```
def qr_from_token(db, tokenname):
    with db.read_connection() as conn:
    token_info = conn.get_tokeninfo_by_name(tokenname)
    config = conn.config

    if token_info is None:
      return {"status": "error", "message": "token {!r} does not
exist".format(tokenname)}

    image = gen_qr(config, token_info)
    fn = "docker-data/dcaccount-%s-%s.png" % (config.mail_domain,
      token_info.name)
    image.save(fn)
    return {"status": "success", "filename": fn}
```

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

To mitigate this issue, Cure53 advises verifying that a token's name does not specify potential malicious character sequences allowing it to alter the local file path. An alternative approach in this respect would be to generate a random filename for each token during creation and use this property when a local file is created or read.

MER-01-002 WP2: Outdated and vulnerable dependencies *(Info)*

Whilst reviewing the source code of the *deltachat-core-rust* repository, the observation was made that the library utilizes a plethora of external libraries, some of which are outdated and incur known vulnerabilities. Publicly available tools such as *snyk*[1] and the *cargo-audit*[2] crate (for the Rust language specifically) can provide assistance for development teams in identifying and mitigating known vulnerabilities in used dependencies and can be directly integrated into the build process.

The software dependencies (Rust crates) that remain vulnerable at the time of testing represent the following:

- libsqlite3-sys, version 0.24.2, Severity HIGH (CVSS 7.5)[3]
- time, version 0.1.44, Severity MEDIUM (CVSS 6.2)[4]

To mitigate this issue, Cure53 recommends scanning for vulnerable dependencies on a regular basis, which can be achieved by integrating automated vulnerability scanners into the software's Continuous Integration *(*CI*)* lifecycle. Even in the event the used dependencies do not persist any known weaknesses, all dependencies should be retained up-to-date in adherence with best practice.

[1] https://snyk.io/docs/using-snyk
[2] https://crates.io/crates/cargo-audit
[3] https://rustsec.org/advisories/RUSTSEC-2022-0090
[4] https://rustsec.org/advisories/RUSTSEC-2020-0071

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

MER-01-006 WP2: Lack of TLS minimum version set *(Low)*

During the source code review of the *deltachat-core-rust* library, the observation was made that the *async_native_tls* crate is utilized to establish TLS sessions to an SMTP/IMAP server. Specifically, *async_native_tls* uses *Schannel*[5] on Windows, *Security.framework*[6] on iOS, and *OpenSSL*[7] as SSL/TLS libraries.

Testing confirmed that no minimum TLS version is configured via the function *min_protocol_versio* during initialization of the Rust struct *TlsConnector*. According to the *TlsConnector* documentation, a default value of TLS 1.0 is supplied in that case[8].

Affected file:
deltachat-core-rust/src/login_param.rs

Affected code:

```
pub fn build_tls(strict_tls: bool) -> async_native_tls::TlsConnector {
     let tls_builder =
         async_native_tls::TlsConnector::new().add_root_certificate(...);

     if strict_tls {
         tls_builder
     } else {
         tls_builder
             .danger_accept_invalid_hostnames(true)
             .danger_accept_invalid_certs(true)
     }
}
```

Using an insecure version of the TLS protocol can incur issues on certain operating systems, particularly those that do not set TLS 1.2 as the default value. This is relevant for Windows versions earlier than Windows 8, iOS versions prior to version 8, and Android earlier than version 5.

Furthermore, the Delta Chat app for Android sets the minimum SDK version to 16, meaning that the earliest version of Android supported by the app constitutes 4.1. Consequently, when using TLS connections on this particular Android version, the default version of TLS would be TLS 1.1, which is considered deprecated according to IETF[9].

[5] https://learn.microsoft.com/en-us/windows-server/security/tls/tls-ssl-schannel-ssp-overview
[6] https://developer.apple.com/documentation/security
[7] https://www.openssl.org/
[8] https://docs.rs/async-native-tls/latest/async_native_tls/struct.TlsConnector.html
[9] https://tools.ietf.org/id/draft-moriarty-tls-oldversions-diediedie-00.html

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

Affected file:
https://github.com/deltachat/deltachat-android/blob/master/build.gradle

Affected code:

```
android {
    flavorDimensions "none"
    compileSdkVersion 32
    useLibrary 'org.apache.http.legacy'

    defaultConfig {
 [...]
        minSdkVersion 16
        targetSdkVersion 32
    }
[...]
}
```

To mitigate this issue, Cure53 recommends using *TlsConnector::min_protocol_version* to set the default TLS protocol version to TLS 1.2, since both TLS 1.0 and TLS 1.1 are insecure and no longer considered state-of-the-art. Additionally, one can advise setting the minimum SDK version for the Delta Chat Android app to 22 or higher, which would ensure that Android 5.0 is the minimum supported version for Delta Chat.

MER-01-007 WP2: Providers incur potentially insecure settings *(Info)*

Whilst reviewing the *deltachat-core-rust* library source code, the observation was made that the library contains a hardcoded email provider database, which includes domains and related settings such as SMTP/IMAP ports. This database is utilized to recognize the email servers to which messages are sent.

One can pertinently note that this issue has been discussed with the client during the course of this security assessment, who confirmed that a developer manually reviews the database and creates it using the Python script *deltachat-core-rust/src/provider/update.py*. However, this process could facilitate risk if any email provider entries have been tampered with and remain unnoticed during this process.

To mitigate this issue, Cure53 advises implementing an automated process for generating the email provider database. Additionally, the email provider entries should undergo integrity checks to eliminate any potential for manipulation.

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

Conclusions

The impressions gained during this report - which details and extrapolates on all findings identified during the CW07 testing against the merlinux-curated and mailcow-based mail-server template for Delta Chat usage, the related setup, and several connected Delta Chat Rust libraries by the Cure53 team - will now be discussed at length. To summarize, the confirmation can be made that the components under scrutiny have garnered a positive impression; sufficient security safeguards have been established for the features in scope, though numerous opportunities for hardening were observed and documented.

Firstly, Cure53 would like to comment on all WP1-related findings, which pertained to the merlinux-curated server guide template for Delta Chat, mailcow, and utilized components such as Nginx, Dovecot, Postfix, and the mailadm application. The design assurance that the rolled-out setup ensures an adequately TLS-encrypted and secured connection with the Delta Chat application was subjected to due diligence validation by the testing team. The Postfix and Dovecot configurations also underwent stringent examination for any incorrect authentication settings that may allow credential leakage in either a plaintext session or Man-in-the-Middle (MitM) scenario.

Despite the fact that Postfix exposes SMTP plaintext on port 25, authentication is unavailable without the STARTTLS command. Elsewhere, testing confirmed that the standard TLS/SSL settings were correctly adapted and adhere to typical guidance concerning supplementary hardening procedures, including the exclusion of potentially insecure ciphers.

Generally speaking, both applications are sufficiently configured and the deployed settings take advantage of the security features offered for TLS/SSL encryption. The aforementioned mailadm application and exposed HTTP endpoint - as well as its interaction via the Delta Chat bot component - represent another primary focus area for this assessment. Mailadm is written in Python and leverages the Flask framework to expose an HTTP Post endpoint. The testing team positively noted concise and legible code composition in this respect.

The externally exposed HTTP endpoint did not incur any security weaknesses since only one parameter is parsed, which is handled safely internally, including within any database queries. This viewpoint applies to database statements in addition, due to the fact that parameterized queries are utilized throughout the code. The interaction between Flask and the Nginx reverse proxy was also reviewed for common HTTP-related threats, such as HTTP request smuggling. Positively, these efforts confirmed a lack of associated vulnerabilities.

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

However, the Delta Chat bot component unveiled some weaknesses concerning validation of user-controlled parameters. As documented in tickets MER-01-004 and MER-01-005, a path traversal and an incorrect email validation were detected in the available bot commands. Notably, this communication channel is only available to administrators and no subsequent issues were observed in the bot's logic that would otherwise have allowed non-administrators to execute commands. Lastly, the mailcow GUI setup aspect was evaluated to determine the presence of any persistent security flaws.

In this respect, testing confirmed a shell injection vulnerability in the application's *Sync Job* feature, as stipulated in ticket MER-01-008. Nevertheless, the necessary permission must be assigned to an account; subsequently, the attached severity marker was appropriately downgraded. Despite the minimal impact, this behavior provides a perfect example of the effect any third-party application can incur upon the overall security foundation. In light of this, Cure53 highly recommends ensuring all deployed software are regularly updated to the latest available version.

Next, Cure53 would like to offer conclusory commentary on all WP2-related findings pertaining to the Delta Chat core library and other libraries utilized in this context. The *deltachat-core* library was developed using the Rust programming language with a code base that was easy to comprehend and understand. Generally, the code reviewed favorably during this pentest, though one must underline that some potential security risks were identified despite its evident quality. Delta Chat leverages third-party libraries for specific operations, and as such remains entirely dependent on the security resilience of said libraries. In this respect, Cure53 recommends enforcing a strict update regiment for all dependencies to negate any security issues persisted in a timely fashion.

As part of this review, a thorough scan of dependencies was performed, revealing that two crate dependencies are both outdated and susceptible to vulnerabilities, as outlined in ticket MER-01-002. The library's code has been fortified by applying *forbid(unsafe_code)*, which helps to prevent usage of any potentially insecure code segments. This practice facilitates a solid codebase resistant to a host of common attack vectors. Furthermore, the developers have implemented *clippy* for static code analysis, enabling the detection and prevention of commonly-found coding errors. SQLite constitutes the chosen database for the core library. A scan for potential SQL injection vulnerabilities was performed here, for which the testing team confirmed sufficient protection via prepared statements. The SQLite database was deemed unencrypted for the desktop app, storing all passwords in plain (see MER-01-003).

This absence of password protection and SQLite database encryption can expose all data contained to various security risks, including unauthorized access to sensitive information, such as passwords.

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

A primary focus was placed on async-native-tls, a Rust crate used for establishing TLS connections, with the following observations garnered for this feature: async-native-tls uses *SChannel* internally for Windows, *Security.framework* for iOS, and *OpenSSL* for all other platforms. Testing confirmed that the core library does not establish a minimum TLS version, as stipulated in ticket MER-01-006. This may result in compatibility and security issues with older operating systems, such as those preceding Windows 8, Android 4.1, and iOS 8.

The connection handling itself was audited, though no associated weaknesses were detected in this regard. The SMTP module was also scrutinized for STARTTLS downgrade attacks and was subsequently deemed error-resistant. The SOCKS5 connection handling was examined, which raised the absence of a socket connection timeout. However, this behavior was considered unexploitable and does not induce susceptibility to risk, thus no noteworthy findings were reported. In addition, the email provider handling was subjected to rigorous analysis, which resulted in the detection of an insecure workflow regarding email provider database creation (see MER-01-007). Finally, the HTTP and NET modules that establish HTTP clients and TCP streams were thoroughly audited, though this area yielded a lack of results.

In conclusion, the in-scope components garnered a positive impression following the completion of this review, particularly relating to strong SSL/TLS encryption, the mail server, and client library, which have all made excellent progress toward offering a first-rate framework from a security perspective. However, the mailcow backend was deemed suboptimal, for which the discovered issue should be addressed by the developer team at the earliest possible convenience. Moving forward, Cure53 recommends conducting another comprehensive security assessment against the mailcow code, which will ensure that existing vulnerabilities and miscellaneous issues are sufficiently addressed, that newly-introduced functionalities cannot incur fresh vulnerabilities and attack vectors, and that any emerging issues can be proactively negated as early as possible in the software development process.

Cure53 would like to thank Holger Krekel, Björn Petersen, Nami, and all other participatory personnel from the merlinux GmbH and maintainer teams for their excellent project coordination, support, and assistance, both before and during this assignment.