Fine penetration tests for fine websites

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

Pentest-Report Enpass Windows Client, API & Server 06.2022

Cure53, Dr.-Ing. M. Heiderich, F. Grunert, N. Boecking, S. Schirra, BSc. F. Heiderich

Index

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

Introduction

"Enpass not only takes care of your passwords, but also your credit cards, driving licenses, passports, and all the personal files you need to keep secure and handy."

From https://www.enpass.io/features/

This report - entitled ENP-01 - details the scope, results, and conclusory summaries of a penetration test and source code audit against the Enpass Windows client and UI, backend API endpoints, plus underlying backend and server. The work was requested by Enpass Technologies Inc. in May 2022 and initiated by Cure53 in May and June 2022, namely between CW22 and CW24. A total of twenty days were invested to reach the coverage expected for this project.

The testing conducted for ENP-01 was divided into three separate work packages (WPs) for execution efficiency, as follows:

- **WP1**: White-box pentests and code audits against Enpass Windows client and UI
- **WP2**: White-box pentests and code audits against Enpass backend API
- **WP3**: Gray-box pentests and assessments against Enpass backend and server

Cure53 was provided with a binary, sources, pertinent documentation, URLs, as well as any alternative means of access required to complete the audit. For these purposes, the methodology chosen was white-box for the first two WPs and gray-box for the third as requested. A team of five senior testers was assigned to this project's preparation, execution, and finalization. All preparatory actions were completed in May 2022, namely in CW21, to ensure that the testing phase could proceed without hindrance or delay.

Communications were facilitated via a dedicated, shared Slack channel deployed to combine the workspaces of Enpass and Cure53, thereby allowing an optimal collaborative working environment to flourish. All participatory personnel from both parties were invited to partake throughout the test preparations and discussions. One can denote that communications proceeded smoothly on the whole. The scope was well-prepared and clear, no noteworthy roadblocks were encountered throughout testing, and cross-team queries were kept to a minimum as a result. Enpass delivered excellent test preparation and assisted the Cure53 team in every respect to procure maximum coverage and depth levels for this exercise.

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 offered by Cure53 and subsequently conducted via the aforementioned Slack channel. Regarding the findings in particular, the Cure53 team

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

achieved comprehensive coverage over the WP1 through WP3 scope items, identifying a total of eleven. Eight of these findings were categorized as security vulnerabilities, whilst the remaining three were deemed general weaknesses with lower exploitation potential.

Generally speaking, the overall yield of findings is relatively moderate for a scope of this magnitude and complexity. This would typically constitute a positive indication regarding the Enpass Windows client's perceived security posture. However, this positive impression is somewhat jeopardized by the sole *High* severity-rated issue detected during this audit.

The *High* severity issue - which specifically pertains to a license activation bypass that can facilitate an activation of the desktop application to Lite, Premium, or Business licenses without registration or payment - should be addressed and mitigated with utmost priority at the earliest possible convenience to minimize any associated risk to the client's payment structure. Further guidance related to this finding is documented in ticket ENP-01-009. Nevertheless, no additional significant attack surfaces or threats were unveiled during this test bar the aforementioned vulnerability.

However, the testing team would like to underline that the majority of all findings were discovered during the pentests and assessments against the Enpass backend and underlying server under WP3. This indicates that these components represent a priority area for targeted hardening and would greatly benefit from improvement in order to elevate the security level in general. All in all, Cure53 can conclude that the Enpass Windows client has already incorporated a solid security framework prior to this Cure53 audit, though the plethora of findings identified provides ample evidence of the necessity for security improvement to reach a first-rate security posture.

The report will now shed more light on the scope and testing setup as well as provide a comprehensive breakdown of the available materials. This will be followed by a chapter outlining the test coverage for each work package, which serves to provide greater clarity on the techniques applied and coverage achieved throughout this audit. 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 Enpass Windows client and UI, backend API endpoints, as well as underlying backend and server, giving high-level hardening advice where applicable.

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

Scope

- **Penetration tests and source code audits against Enpass Windows client, API, and server**
  - **WP1**: White-box pentests and code audits against Enpass Windows client and UI
    - **Tested binary:**
      - https://dl.enpass.io/stable/windows/setup/6.8.1.1063/Enpass-setup.exe[1]
    - **Tested version:**
      - 6.8.1.1063
    - **Additional documentation:**
      - https://support.enpass.io/docs/security-whitepaper-enpass/index.html
    - All relevant sources and documentation were shared
  - **WP2**: White-box pentests and code audits against Enpass backend API
    - **URLs in scope:**
      - https://license.enpass.io
      - https://rest.enpass.io
      - https://console.enpass.io
    - **Additional documentation:**
      - https://support.enpass.io/business/console/getting_started_with_enpass_admin_console.htm
      - https://support.enpass.io/business/app/setup/setting_up_enpass_business.htm
    - All relevant sources and API documentation were shared
  - **WP3**: Gray-box pentests and assessments against Enpass backend and server
    - In scope were all subdomains from the *enpass.io* domain
      - Note: As requested by Enpass onJun 2, 2022via Slack, the subdomain https://btlicense.enpass.io/ was excluded from the scope.
- **Test-users utilized**
  - U: consoletestuser@acmebizness.com
  - U: apple_user@acmebizness.com
  - U: apple_user@acmebizness.com
- **Test-supporting material was shared with Cure53**
- **All relevant sources were shared with Cure53**

[1] sha256: 823dca8f74169cedfa5047d30a220f3635e7d66599d0509a49a60fe994cd8e22

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

Test Methodology

The primary objective of this report's Test Methodology section is to elaborate on the Cure53 team's comprehensive testing process, giving context and transparency towards the actions performed, the vulnerability classes confirmed, and the exploitation attempts negated. Since the overall testing process was divided into client-side security checks on the client software for Windows; numerous server-side checks on the backend API; and the investigation of the servers themselves, the following three sections separately detail the security audit methods for those areas.

Test Coverage for WP1: Enpass client software for Windows

- The application's source code was reviewed to determine any usage of insecure vulnerable functions, such as *strcpy*, *strcat*, *memcpy*, *sprintf*, and *snprintf*. As a result, two locations were identified that suffer from an off-by-one heap buffer overflow possibility (see ENP-01-007 and ENP-01-008).
- The application supports connections to alternate cloud storage providers. The uploader functionality for all providers was statically reviewed for weaknesses in the memory management and API calls. Here, testing confirmed that all objects are removed from the memory, hence no overflows could be identified. Furthermore, all API calls are correct and could not be manipulated. Positively, no issues in this area were identified.
- The secure memory plus secure string implementation and usage was statically reviewed by assessing the source code in order to ensure that all sensitive information is removed from the memory after usage. Additionally, the application was assessed by using dynamic binary instrumentation via frida[2] and WinDBG[3]. Testing confirmed that all data is removed from memory after usage, therefore no issues were identified in this area.
- The Windows application's vault implementation was reviewed to determine the presence of any weak or erroneous behaviors, though positively no issues were identified in this regard.
- The application's crypto implementation was also statically reviewed. Here, the confirmation was made that the application only utilizes secure ciphers and block modes. However, several hard-coded keys were located in the source code, though these keys belong to an older version of the application and are only included for compatibility reasons. No further issues were identified otherwise.
- The implementation and the usage of the Pseudo Random Number Generator (PRNG) were statically reviewed. Here, testing confirmed that the OpenSSL implementation is utilized in a correct and secure manner, therefore no associated issues were identified.

[2] https://frida.re
[3] https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/debugger-download-tools

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

- The application was also dynamically reviewed for DLL hijacking vulnerabilities by using the Process Monitor from the *sysinternals* suite[4]. Here, the observation was made that all DLLs are loaded from safe areas, therefore no associated issues were detected.

Test Coverage for WP2: Enpass admin and backend API

- Testing was initiated to determine the presence of security-relevant HTTP headers within all involved applications, as well as correct configuration and secure implementation. No weaknesses were detected in this regard.
- The API calls were checked for authentication flaws that could allow unauthenticated or low-privileged users to perform authorized actions. No issues were found in this domain.
- The API endpoints and the Admin Console were tested for injection flaws such as SQL injection, Cross-Site-scripting or code injections. Positively, no security flaws were found.
- Attempts to intercept communication without a trusted server certificate and blocking communication in order to achieve unexpected behavior from the application were executed. Here, one major weakness was located that allows potential attackers to bypass the license activation of the Enpass application (see ENP-01-009).
- During the analysis, the application design was reviewed to determine the presence of any potential security risks. Two observations were filed regarding the use of the UUIDv1 algorithm (see ENP-01-006) and the storage of sensitive information in the session storage (see ENP-01-003).
- The API interfaces were assessed for any indication that functions may be exploitable via automation attacks. Positively, no indications were found and sufficient rate limiting was observed.
- The API endpoints were also analyzed for serialization and deserialization issues, though no insecure deserialization procedures were located.

[4] https://docs.microsoft.com/en-us/sysinternals/

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

Test Coverage for WP3: Enpass servers and infrastructure

- The security configuration of the web servers hosting the Admin Console and the API interfaces were deep-dive assessed. One low-risk observation was made regarding the disclosure of technical environment information, as documented in ticket ENP-01-001.
- Port scans were conducted upon all systems in scope with the aim of identifying open ports and additional services that may be susceptible to abuse. Positively, no security-related issues were discovered here.
- The systems in scope were analyzed for outdated or deprecated software versions affected by publicly-known vulnerabilities. Here, one system was identified that utilized an outdated nginx web-server version (see ENP-01-005). References to outdated JavaScript libraries were also located in the applied Content Security Policy (see ENP-01-004).
- Additionally, a security analysis of the network communication to and from the Enpass application was executed, as well as an analysis of the communication towards the Admin console and the API interfaces. Here, one *Medium* risk observation was made regarding the security of the implemented transport encryption, as detailed in ticket ENP-01-002.

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

Identified Vulnerabilities

The following sections list all vulnerabilities and implementation issues identified throughout the testing period. Please note that findings are listed in chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Furthermore, each vulnerability is given a unique identifier (e.g., *ENP-01-001*) to facilitate any future follow-up correspondence.

ENP-01-001 WP3: Leakage of precise nginx version via *license.enpass.io* *(Low)*

***Note**: This issue was fixed by the Enpass team and the fix was verified by Cure53, the problem no longer exists.*

A disclosure of technical information was identified in the server response header of the API interface *license.enpass.io*.

Specifically, the following technical information was disclosed:

- `Software: nginx/1.20.0`
- `Host: license.enpass.io`
- `Location: HTTP Response Header`

Received response header:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://console.enpass.io
Allow: POST, OPTIONS
Content-Security-Policy: [...]
Content-Type: application/json
Date: Tue, 31 May 2022 12:00:27 GMT
Server: nginx/1.20.0
Vary: Cookie, Origin
X-Frame-Options: SAMEORIGIN
Content-Length: 75
Connection: Close
```

Steps to reproduce:

1. Log into the Admin Console application (*https://console.enpass.io*).
2. Trigger any action in the application and intercept a server response from an outgoing request to the API (*license.enpass.io*) using common browser developer tools or an intercepting proxy, such as Burp Suite[5].
3. Inspect the server response HTTP header for the header value *Server:*.

[5] https://portswigger.net/burp

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

Disclosure of technical product data provides any would-be attacker with sensitive information concerning the internal system structure, meaning that specific vulnerabilities for the components and versions in-use can be targeted. If new security vulnerabilities (zero-day security vulnerabilities) are published for the disclosed technologies, the affected systems may become a prioritized attack target.

Generally speaking, product and version information relating to deployed components should not be disclosed. Cure53 recommends adjusting the system configuration of the server components to suppress the disclosure of server banners.

ENP-01-002 WP3: Outdated TLS version in multiple domains *(Medium)*

***Note**: This issue was fixed by the Enpass team and the fix was verified by Cure53, the problem no longer exists.*

Testing confirmed the presence of a web API utilizing outdated TLS versions. The supported protocol versions are no longer considered best practice from a security viewpoint and can lead to insecure communication if not due diligently configured. As a result, the communication between the client and the server cannot be considered comprehensively protected.

A result excerpt of the tool *sslscan*[6] for the domain *rest.enpass.io* is offered below:

```
SSL/TLS Protocols:
SSLv2  disabled
SSLv3  disabled
TLSv1.0      enabled
TLSv1.1      enabled
TLSv1.2      enabled
TLSv1.3      disabled
```

Affected domains:

- *rest.enpass.io*
- *license.enpass.io*

The BSI (Bundesamt für Sicherheit in der Informationstechnik / German Federal Office for Information Security) advises against usage of the SSLv2, SSLv3, TLS v1.0 and TLS v1.1 protocols (since April 2020)[7]. The BSI describes the requirements for TLS in the document "Mindeststandard des BSI zur Verwendung von Transport Layer Security". According to the DSGVO (GDPR - General Data Protection Regulation), the technical

[6] https://github.com/rbsec/sslscan
[7] https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Mindeststst...df?__blob=publicationFile&v=4

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

measures implemented to achieve the protection goals must comply with current standards. Consequently, the protocol versions no longer recommended by the BSI do not meet this requirement.

The results can be reproduced by performing an SSL security scan on the domains *rest.enpass.io* and *license.enpass.io* (Port: *443\\tcp*) using either *sslscan* or *sslyze*[8] .

To mitigate this issue, Cure53 advises disabling support for TLSv1.0 and TLSv1.1 on the server and only leveraging secure versions such as TLSv1.2 or higher. Notably, configuration alterations to the TLS versions can have an impact on compatibility with older client systems, therefore one should carefully consider the potential implications of such changes before they are deployed.

ENP-01-005 WP3: Outdated nginx version on *license.enpass.io* (Medium)

***Note**: This issue was mitigated by the Enpass team and the fix was verified by Cure53, the problem no longer exists.*

Testing confirmed that the web service under the domain *license.enpass.io* operates with an nginx web server in version 1.20.0. According to the vendor and public vulnerability databases, this software version is affected by a security vulnerability, which is listed as CVE-2021-23017[9] and is related to a potential Denial of Service attack (DoS).

The detection is based on the displayed version number, which is transmitted by the server in the HTTP response header.

Response header:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://console.enpass.io
Allow: POST, OPTIONS
Content-Security-Policy: [...]
Content-Type: application/json
Date: Tue, 31 May 2022 12:00:27 GMT
Server: nginx/1.20.0
Vary: Cookie, Origin
X-Frame-Options: SAMEORIGIN
Content-Length: 75
Connection: Close
```

[8] https://github.com/nabla-c0d3/sslyze
[9] https://nvd.nist.gov/vuln/detail/CVE-2021-23017

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

Steps to reproduce:

1. Log into the Admin Console application (*https://console.enpass.io*).
2. Trigger any action in the application and intercept a server response from an outgoing request to the API (*license.enpass.io*) using common browser developer tools or an intercepting proxy, such as Burp Suite[10].
3. Inspect the server response HTTP header for the header value *Server:*.

This vulnerability may allow an attacker that is able to forge UDP packets from a DNS server to instigate a memory overwrite, resulting in a DoS of the server and - in the worst case scenario - a remote code execution (RCE), which could lead to a compromised server. Nevertheless, the vulnerability can only be exploited if a resolver has been added to *nginx.conf*. If no entry for a resolver in the config file exists (as in the following excerpt), the vulnerability cannot be exploited.

```
location / {
      resolver x.x.x.x;
      proxy_pass    http://example.com;
  }
```

Whilst the official CVS score for the application is rated as greater than 8, the associated risk of this issue is considered *Medium* due to the low attack likelihood. Nevertheless, the nginx software should be updated to the latest version to address the aforementioned vulnerability.

ENP-01-006 WP2: Weak UUIDv1 algorithm usage for team IDs *(Low)*

***Note**: This issue was fixed by the Enpass team and the fix was verified by Cure53, the problem no longer exists.*

The observation was made that UUIDv1 tokens are utilized as unique identifiers. In order to request information regarding a team, a team ID is used in requests towards the application API. A sample request is offered below.

Sample request:

```
POST /api/v1/policy/team/ HTTP/1.1
Host: license.enpass.io
[…]

{"team":"f597428e-b977-11ec-869f-0242ac110002"}
```

[10] https://portswigger.net/burp

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

Team ID token example:
F597428e-b977-11ec-869f-0242ac110002

The used identifier structure is equal to the structure of UUID tokens; the structure of UUIDv1 tokens constitutes the following[11]:

- Timestamp: 60 Bit (TimeLow 4 Bytes, TimeMid 2 Bytes, 12 Bits of remaining UTC time).

Time conversion for the used token (time when token was created):

```
Hex Value: 1ecb977f597428e
Decimal Value: 138689613414220430
Epoch: (138689613414220430- 122192928000000000) / 10000 = 16.496.685.414.220,43
Date: GMT: Monday, 11. April 2022 09:15:41.422
```

- NodeID: 48 Bit MAC address of the system generating the token
  - In the example: 02:42:ac:11:00:02
- Clock Sequence: 14 bit
- UUID version: 6 Bit

UUIDv1 token usage discloses information concerning the timestamp, clock sequence, and MAC address of the system issuing the UUID token as demonstrated in the previous example. The randomness of UUIDv1 tokens is also limited since one can predict them with greater ease when previous tokens are known.

Several security issues related to the randomness of UUID tokens have been published in the past. Most of the vulnerabilities were fixed in later versions such as version 4. The security of generated UUIDv4 tokens can vary depending on the implementation used. RFC4122[12] describes the structure and implementation of UUID tokens, though lacks information regarding the cryptography to be used. Therefore, RFC-compliant UUIDv4 implementations were developed using weak random number generators such as *math.random()*.

The RFC in paragraph 6 states: "*Do not assume that UUIDs are hard to guess; they should not be used as security capabilities (identifiers whose mere possession grants access)*". Some UUIDv4 implementations are considered secure and are also recommended by manufacturers for security-critical functions, such as the NodeJS "crypto" library. The security of the tokens ultimately depends on the respective implementation.

[11] https://www.ietf.org/rfc/rfc4122.txt
[12] https://www.ietf.org/rfc/rfc4122.txt

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

While one can generally recommend ensuring unique IDs are unpredictable, testing was initiated to determine whether sufficient authorization is enforced and even if foreign team IDs are known, access without permission is denied. This vulnerability's associated severity rating was assigned based on the exposure of system information and the weakness of the issued tokens' randomization.

To reproduce the issue, log in to the Admin Console application and inspect the server response header when viewing a team using the common Browser developer tools or an intercepting proxy.
The following guidance is recommended when handling security-related tokens:

- UUIDv1 to UUIDv3 are affected by various security issues and should not be utilized at all.
- UUIDv4 tokens can be secure, though this is dependent on the implementation.
- The security of the library implementation that creates UUIDv4 tokens should be assessed before usage.

ENP-01-007 WP1: Potential OBO heap buffer overflow in *file_upload_cb* (Low)

***Note**: This issue was fixed by the Enpass team and the fix was verified by Cure53, the problem no longer exists.*

Testing confirmed that the HTTP server's *file_upload_cb* function in the *httpserver.cpp* file suffers from an off-by-one heap buffer overflow. Specifically, the function *file_upload_cb* utilizes *strlen* to retrieve the length of two strings in order to calculate the length of a target buffer used to allocate a buffer in the heap, as highlighted below:

Affected file:
httpserver.cpp

Affected code (line 66 f):

```
const char *resource_uri_path=server->_resource_uri.data();
char *resource_path = (char *)malloc(strlen(resource_uri_path) +
strlen(filename)+1);
```

The length of the buffer is calculated as follows:

```
length resource_uri_path + length filename + 1
```

This length is sufficient for the strings and the trailing null byte. However, it is used to concatenate the *resource_uri_path*, the *filename* and a path separator (\), as shown below.

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

Affected code (line 70 ff):

```
memset(resource_path,0,sizeof(resource_path));
strcat(resource_path,resource_uri_path);
strcat(resource_path,"\\");
strcat(resource_path,filename);
```

This means that the size of the buffer is sufficient for the strings itself. However, the trailing null bytes added by *strcat* exceed the allocated memory and is written to the byte right after the buffer *resource_path* overwriting memory located behind the *resource_path* buffer, which can facilitate unpredictable behavior.

To mitigate this issue, Cure53 recommends increasing the buffer for the string by 1 in order to fit the trailing null byte, as shown in the following excerpt:

```
char *resource_path = (char *)malloc(strlen(resource_uri_path) +
strlen(filename)+1 + 1 /*additional byte for the trailing null byte*/)
```

ENP-01-008 WP1: Potential OBO heap buffer overflow in *callback_http* (Low)

***Note**: This issue was fixed by the Enpass team and the fix was verified by Cure53, the problem no longer exists.*

Testing confirmed that the HTTP server's *callback_http* function within the file *httpserver.cpp* suffers from an off-by-one heap buffer overflow. The function *file_upload_cb* utilizes the *strlen* to retrieve the length of two strings in order to calculate the length of a target buffer used to allocate a buffer in the heap, as shown below:

Affected file:
httpserver.cpp

Affected code (line 117 ff):

```
const char *resource_uri_path=server->_resource_uri.data();
char *resource_path;

// allocate enough memory for the resource path
resource_path = (char *)malloc(strlen(resource_uri_path) +
strlen(requested_uri));
```

The length of the buffer is calculated as follows:

```
length resource_uri_path + length requested_uri
```

This length is sufficient for the strings, though not for the trailing null byte.

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

Affected code (line 124):

```
sprintf(resource_path, "%s%s", resource_uri_path, requested_uri);
```

This means that the size of the buffer is sufficient for the strings itself. However, the trailing null bytes added by *sprintf* exceed the allocated memory and are written to the byte right after the buffer *resource_path* overwriting memory located behind the *resource_path* buffer, which can lead to unpredictable behavior.

To mitigate this issue, Cure53 advises increasing the buffer for the string by 1 in order to fit the trailing null byte, as shown in the following excerpt:

```
resource_path = (char *)malloc(strlen(resource_uri_path) + strlen(requested_uri)
+ 1 /*additional byte for the trailing null byte*/);
```

ENP-01-009 WP2: License-activation flaws facilitate bypass *(High)*

The discovery was made that the license activation can be bypassed, allowing attackers to activate the desktop application without registration or payment for Lite, Premium, or Business licenses.

Upon first use of the Enpass desktop application, a user is requested to activate the application. The user can then select an existing account (with a certain license attached such as Premium) or register a new user for free to receive a "Lite User" license. The activation process is realized via HTTP requests to the API interface *license.enpass.io*.

The regular activation process constitutes the following steps:

1. User clicks on *Activate* in the desktop application.
2. User enters the email address and confirms.
3. A sign-in API call to endpoint: */api/v1/user/signin* is triggered.
4. An OTP is sent to the email address provided in Step 2.
5. User enters the OTP from the email and confirms.
6. A verification OTP API call to endpoint */api/v1/user/verify/otp/* is triggered.
7. An *access_token* is assigned to the user.
8. A subscription info API call to endpoint */api/v1/subscription/me/* is triggered (including *access_token*).
9. The application receives information regarding the user status, subscription (Lite/Premium) and affected policies (Business)

By activating the proxy server option, one could intercept all outgoing and incoming application network traffic via an intercepting proxy.

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

By modifying the incoming server responses, the registration workflow can mostly be bypassed prior to being passed to the application via the following steps:

1. An attacker provides a random email address in the activation form. An OTP is sent to the provided address but is not used in the following steps.
2. The attacker uses a random OTP and triggers the API call to */api/v1/user/verify/otp/*. The server response containing an error is modified to a success and passed to the application. Original and edited responses are listed as examples below:

   Original response from API endpoint */api/v1/user/verify/otp/* when using wrong OTP:

```
HTTP/1.1 200 OK
Allow: POST, OPTIONS
[cropped HTTP header]
Content-Length: 78
Connection: Close

{"error":true,"code":"otp_invalid","description":"OTP is invalid or
expired."}
```

   Modified response from API endpoint */api/v1/user/verify/otp/* assigning random *access_token*:

```
HTTP/1.1 200 OK
Allow: POST, OPTIONS
[cropped HTTP header]
Content-Length: 145
Connection: Close

{"error":false,"code":"login_success","description":"You have
successfully logged
in.","access_token":"6a6957ab22any22random22token222569ead13f"}
```

3. The API call to the endpoint */api/v1/subscription/me/* is then triggered. The server response containing an error (owing to the wrong access token) is modified to a success and passed to the application. The response data from a legit call to the API endpoint includes license, policy, and identity information regarding the subscriber. This data is altered from license value *lite* to *premium*. Original and edited responses are listed as examples below:

   Original response from API endpoint */api/v1/subscription/me/* when using wrong *access_token*:

```
HTTP/1.1 401 Unauthorized
Allow: POST, OPTIONS
```

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

```
[cropped HTTP header]
Content-Length: 27
Connection: Close

{"detail":"Invalid token."}
```

Modified response from API endpoint */api/v1/subscription/me/* assigning premium license:

```
HTTP/1.1 200 OK
Allow: POST, OPTIONS
[cropped HTTP header]
Content-Length: 1814
Connection: Close

{
       "error": false,
       "code": "success",
       "description": "",


       "profile": {
       "name": "RL Test Proxy Premium Activation ",
       "email": "nonreg@recurity-labs.com"
       },

       "license": "premium",
       "status": "active",
       "migrated":false,

       "duration": {
       "start_date": "1653982745",
       "end_date": "1656574745"
       },

       "provider": {
       "type": "organization",
       "name": "nonreg Any Business",
       "partner": "enpass",
       "team_id": "e80f0744-e660-11ec-8fea-0242ac120002",
       "team_state": "active"
       },
       "info": {
       "store": "team",
       "id": "123",
       "userid": "1234",
        "transaction_id":"",
       "purchase_type": "team",
```

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


```
        "logo": ""
        },
        "limits": {
        "items": {
              "desktop": "-1",
              "mobile": "-1"
        }
        },

        "client_policies" : {"mp": {"min_length": 20, "min_strength": 1},
"mp2": {"min_length": "test", "min_strength": 2}, "security":
{"clear_clip778881board": "123", "clear_clip124board": true,
"clear_clip123board": true, "clear_clipboard_interval": 30,
"desktop_inactivity_interval": 1, "desktop_inactivity_type": 1,
"mobile_inactivity_interval": 60, "mobile_lock_on_exit": true,
"hide_sensitive": true}, "sharing": {"psk_mandatory": true,
"psk_min_strength": 3}, "team_policy": {"team": {"domains":
[ "ZnJlZXBhc3NkYXNkYXNkc2Q.pentest", "ZGFzZGFzZHNk"],
"primary_vault_forced": false, "allowed_backup": 0, "allowed_export": 0,
"allowed_sharing": 1, "can_change_data_location": true,
"allow_copy_item_outside_team_account": false, "name": "Any Nonreg",
"icon": "", "team_slug": "", "cloud": "onedrive-team"}}, "advanced":
{"ios_universal_clipboard": true}},

        "offer_available" : false
}
```


4. The application is now activated with the license type Enpass Premium/Business/Pro.

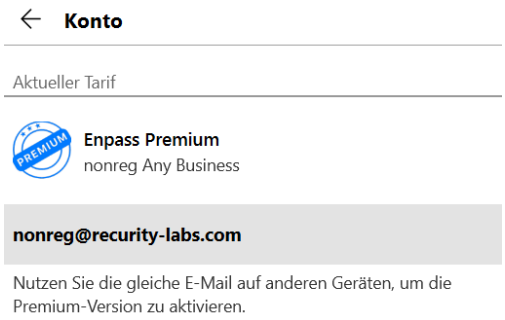

Fig.: License activation as Premium user.

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

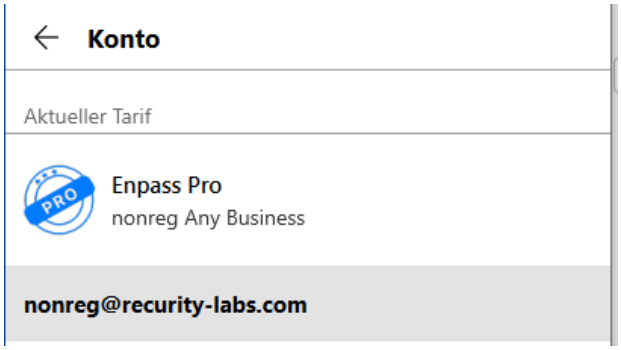

Fig.: License activation as Pro user.

This risk-laden behavior could facilitate bypass of feature and usage restrictions via alternating license types. This includes current limitations as well as future changes to limitations. The weakness may also incur financial loss should information concerning the described bypass be released to public access. Furthermore, attackers may establish their own activation server that responds similarly to the described example. This activation server may then be used to activate business features such as policy assignments for a complete environment of accounts and systems.

To mitigate this issue, Cure53 recommends reviewing the security of the activation process. During this process, one should ensure that the destination server constitutes a valid Enpass licensing server and that the integrity of the message contents is retained. A multitude of implementation concepts such as public key pinning, key exchanges, and checksums can provide assistance toward achieving this. Lastly, the impact of this bypass upon Enpass mobile applications should also be subject to review.

ENP-01-010 WP3: Insufficient input validation for minimum password length *(Low)*

***Note**: This issue was fixed by the Enpass team and the fix was verified by Cure53, the problem no longer exists.*

Testing confirmed that user input during the policy configuration is not sufficiently validated and may cause the Enpass client application to malfunction. The minimal length of a master password can be defined in the team policies configuration option in the Admin Console. The provided value of type integer remains unlimited to a specific size, allowing attackers to set an excessively large minimal password length.

The following example demonstrates a modified API request to the endpoint *license.enpass.io/api/v1/policy/team/update/* successfully setting an excessive minimum password length.

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

```
POST /api/v1/policy/team/update/ HTTP/1.1
Host: license.enpass.io
[...]
Connection: close

{"team":"f597428e-b977-11ec-869f-0242ac110002","policy_json":"{\"mp\":
{\"min_length\":60000000000000000000000060000000000000000000000060000000000000
00000000000060000000000000000000000060000000000000000000000060000000000000000
00000600000000000000000000000060000000000000000000000060000000000000000000000
06000000000000000000000000060000000000000000000000060000000000000000000000,\"m
in_strength\":2},\"security\":
{\"clear_clipboard\":true,\"clear_clipboard_interval\":30,\"desktop_inactivity_i
nterval\":1,\"desktop_inactivity_type\":1,\"mobile_inactivity_interval\":60,\"mo
bile_lock_on_exit\":true,\"hide_sensitive\":true},\"sharing\":
{\"psk_mandatory\":true,\"psk_min_strength\":3},\"team_policy\":{\"team\":
{\"domains\":
[\"acmebizness.com\",\"cure53.de\"],\"primary_vault_forced\":false,\"allowed_bac
kup\":0,\"allowed_export\":0,\"allowed_sharing\":1,\"can_change_data_location\":
true,\"allow_copy_item_outside_team_account\":false}},\"advanced\":
{\"ios_universal_clipboard\":true}}"}
```

The assigned password minimum length would be accepted and is applied to the team policy.

Fig.: Applied minimum password length.

Whilst assessing the impact on the clients regarding this configuration, the observation was made that the client activation in the desktop application fails as soon as the policy with the large minimal password length is fetched. This indicates that client activations are no longer possible if the minimum password length is set too high by an attacker or by misconfiguration. In order to reproduce the observation, an excessively large number can be directly typed into the minimum password length form field. Alternatively, the according numeric value in the outgoing request can be altered with an intercepting proxy. To mitigate this issue, Cure53 advises limiting the minimum password length that can be assigned by a user. The input validation could be performed on the client side via JavaScript to improve the user experience, but should also be implemented on the server-side to validate input from direct API calls.

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 lead to an exploit but might 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 way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

ENP-01-003 WP2: Admin portal stores auth tokens in session storage *(Info)*

***Note**: This issue was fixed by the Enpass team and the fix was verified by Cure53, the problem no longer exists.*

Testing confirmed that the application utilizes an insecure method to store the authorization token entitled *token*. When authorization tokens are passed to the client after a login, the location wherein the token is stored in the browser is specified. Typically, this can constitute the local storage, session storage, or cookie storage. Tokens stored as cookies are advantageous since they can be equipped with additional protection mechanisms. The cookie flag *httponly*, for example, prevents attempts to read cookies by JavaScript. This cookie flag can prevent the cookie from being hijacked by an attacker through various attack scenarios such as Cross-Site Scripting.

In its CheatSheetSeries, the OWASP Project advises against storing authorization tokens in the local storage[13]. This is justified by the lack of any kind of *httponly* directive, which also applies to the session storage. The following image highlights the *token* authorization token within the Firefox browser's session storage:

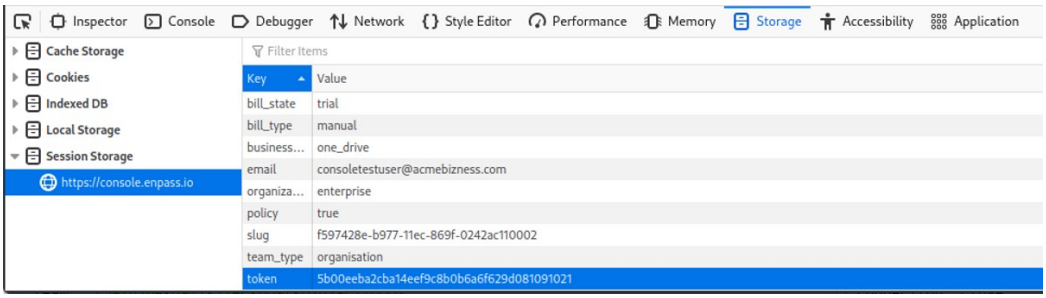

Fig.: Session storage storing a sensitive authorization token.

Steps to reproduce:

1. Log in to the Console-Admin application (*https://console.enpass.io*).
2. View the session storage using the browser developer tools.

[13] https://cheatsheetseries.owasp.org/[...]/HTML5_Security_Cheat_Sheet.html#local-storage

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

3. Observe that the session storage contains locally-saved data, including the *token* authorization token.
4. Confirm that the token can be accessed via JavaScript using the function-call *sessionStorage.getItem("token")*.

Nevertheless, this specific lack of protection could not be exploited during the time frame of this audit. However, if future website features reveal vulnerabilities that allow token reading via JavaScript, for example, any attacker attempts to hijack user sessions will be rendered significantly easier to achieve.

To mitigate this issue, Cure53 advises passing the authorization token as a cookie to benefit from its inherent protection capabilities during storage.

ENP-01-004 WP3: References to outdated JavaScript libraries in CSP *(Info)*

***Note**: This issue was fixed by the Enpass team and the fix was verified by Cure53, the problem no longer exists.*

Whilst deep-dive assessing the Admin Console web application, the discovery was made that references to deprecated JavaScript libraries persist. The API *license.enpass.io* uses a so-called Content Security Policy header (CSP), which can be leveraged to specify which scripts are allowed to be loaded within the web application, for example. However, deprecated libraries and libraries affected by publicly-known vulnerabilities are specified as trusted in the CSP, specifically:

- *https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.8.0/Chart.bundle.min.js*
  - Public CVE:
    - CVE-2020-7746[14]
- *https://code.jquery.com/jquery-3.4.1.min.js*
  - Public CVEs:
    - CVE-2020-11022[15]
    - CVE-2020-11023[16]

The following CSP is sent in the API's response:

```
Content-Security-Policy: img-src 'self' license-enpass-io.s3.amazonaws.com
favicon.enpass.io; style-src 'self' fonts.googleapis.com
cdnjs.cloudflare.com/ajax/libs/Chart.js/2.8.0/Chart.min.css
cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css license-enpass-
io.s3.amazonaws.com 'unsafe-inline'; font-src 'self' fonts.gstatic.com license-
enpass-io.s3.amazonaws.com; connect-src 'self'; default-src 'none'; script-src
```

[14] https://nvd.nist.gov/vuln/detail/CVE-2020-7746
[15] https://nvd.nist.gov/vuln/detail/cve-2020-11022
[16] https://nvd.nist.gov/vuln/detail/cve-2020-11023

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

```
'self' ajax.googleapis.com
cdnjs.cloudflare.com/ajax/libs/Chart.js/2.8.0/Chart.bundle.min.js
cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.min.js
code.jquery.com/jquery-3.4.1.min.js license-enpass-io.s3.amazonaws.com 'unsafe-
inline'
```

Nevertheless, testing could not alleviate evidence of the inclusion of these libraries in the web application. As a result, this issue was merely assigned an *Info* severity rating.

Steps to reproduce:

1. Log into the Admin Console application (*https://console.enpass.io*).
2. Inspect the server response header from a request outgoing to the API (*license.enpass.io*) using common browser developer tools or an intercepting proxy.

To mitigate this issue, Cure53 advises solely utilizing releases of JavaScript libraries that do not contain any known security vulnerabilities. The Enpass team should also assess whether the affected libraries are currently used or planned for future use.

ENP-01-011 WP1: Hard-coded encryption material detected in source code *(Info)*

An analysis of the source code revealed several hard-coded keys and initialization vectors (IV). Nevertheless, it was communicated that all keys solely exist for compatibility reasons. Consequently, this ticket is merely listed for completeness reasons. The following excerpts highlight the identified hard-coded encryption material.

Affected file:
plugincrypto5.cpp

Affected code (line 68 ff):

```
PluginCrypto5::PluginCrypto5() {
	char iv_str[17] = "iqHBpS3qbu6u7qui";

	unsigned char* iv;
	iv = (unsigned char*) malloc(16);
	memcpy(iv, iv_str, 16);

	// AES-128 or AES-256 will be used depending upon size of key generated
using PBKDF2
	unsigned char key[] = "2TjfWW2jbey5ppmi";
	aes_init(key, iv, &mEn, &mDe);
}
```

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

Affected file:
sharehelper.cpp

Affected code (line 972 ff):

```
std::unique_ptr<Item> decryptOldShareLink(const std::string& link, const
std::string &vaultUUID, const std::string &teamID){
	try{
	Url url(link);
	std::string encryptedCardString = url.query("data");
	std::string decodedString;
	Base64::Decode(encryptedCardString,&decodedString);

	auto data = std::make_unique<SecureMemory::ByteArray>((const
uint8_t*)decodedString.c_str(), decodedString.size());
	if(data->size() < 32){
		return nullptr;
	}
	auto iv = SecureMemory::ByteArray::subDataWithRange(data, 0, 16);
	auto salt = SecureMemory::ByteArray::subDataWithRange(data, 16, 16);
	auto itemData = SecureMemory::ByteArray::subDataWithRange(data, 32, data-
>size() - 32);

	auto key = make_secure_string("I4^O$rA9;YNtF(85Dc2_>+zk3gj1B4#u");
	auto crypto5 = std::make_unique<Crypto5>(key,salt,iv,5);
	auto plainData = crypto5->decrypt(itemData);
	if (plainData) {
		//logDebugT("ShareHelper") << __FUNCTION__ << plainData->c_str();;
		return
Keychain2Vault::readShareCardToItem(plainData,vaultUUID,teamID);
	}else
		return nullptr;
	}catch(std::exception& e){
	return nullptr;
	}
}
```

To mitigate this issue, Cure53 recommends removing any hard-coded key material as soon as it is deemed surplus to requirement, particularly if only utilized for compatibility reasons as mentioned in discussion with the maintainer team.

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 CW22 through CW24 testing against the Enpass Windows client and UI, backend API endpoints, plus underlying backend and server 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, though both security strengths and deficiencies were observed across the framework in scope.

Five senior members of the Cure53 team completed the project over the course of twenty working days from late May into mid-June 2022, achieving comprehensive coverage over the vast majority of components and areas in scope. These components were assessed using an extensive array of testing techniques: the first two work packages comprising pentests and code audits against the Enpass Windows client, UI, and backend API endpoints were tested using a white-box methodology, whilst the third work package comprising pentests and assessments against the Enpass backend and server was approached using a gray-box methodology.

Communication was achieved via a shared Slack channel, cross-team queries regarding certain findings and functionality were promptly answered, and the engineering team provided immediate assistance to the testing team when required. The Enpass team also facilitated a trouble-free testing phase by providing a binary, sources, documentation, and URLs prior to the audit. This was particularly welcome in situations whereby application flows or technical issues were initially difficult to understand, since the Enpass team was able to verify that the consultants had obtained a correct understanding of the target system.

Generally speaking, Cure53 gained a positive impression regarding the security posture of the Enpass client, primarily owing to the fact that only a moderate volume of eleven findings were detected. Specifically, eight of these findings were categorized as security vulnerabilities, whilst three were deemed hardening and best-practice recommendations.

Furthermore, the testing team is pleased to confirm the complete lack of *Critical* severity-rated findings and only one *High* severity vulnerability. This corroborates the viewpoint that the Enpass team has already instilled a solid security foundation for the Enpass client, UI, and corresponding API endpoints.

Regarding the findings specifically, the deep-dive assessment against the Admin Console and associated API showcased that an average level of security had been achieved within these areas, since a plethora of findings considered *Medium* and *Low* in nature were identified here.

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

Toward this, the testing team observed that user input often remained unsanitized or checked for invalid values. Nevertheless, Cross-Site Scripting payloads and other injection attacks could not be exploited due to sufficient client- and server-side protections. One exception to this was detected regarding a policy whereby a minimum length for a required password can be set to an excessively high numeric value, as documented in ticket ENP-01-010.

Elsewhere, the API interfaces exhibited strong authorization checks that are strictly performed for each requested entity. However, an authorization configuration vulnerability was identified pertaining to the storage location of a security-critical authorization token (see ENP-01-003). Cure53 recommends adopting a more secure method of token storage to mitigate this issue.

Furthermore, several mostly minor weaknesses in the general security configuration were identified - including usage of a deprecated and vulnerable web-server software - as detailed in ticket ENP-01-005. Additionally, a selection of misconfigured HTTP headers were observed that should be hardened to sufficiently obfuscate any sensitive information relating to the technical environment. Supplementary testing in this area also revealed a weak Universally Unique Identifier (UUID) implementation (see ENP-01-006). Even though these findings should be considered relatively minor issues in isolation, they may enable an attacker to obtain system information or locate greater attack surfaces in the future.

Whilst performing additional assessments, the discovery was made that the license activation process can be bypassed for the desktop application to receive an arbitrary license (Pro or Premium) or abuse business functions such as the policy management. This bypass may also incur direct and significant financial loss upon the organization. Cure53 recommends reviewing the activation process to incorporate a stronger process from a security standpoint. Further information regarding this issue can be perused in ticket ENP-01-009.

The testing team also observed that the application utilizes a host of functions - including *strcpy*, *strcat*, *memcpy*, *sprintf*, and *snprintf* - that are considered insecure. Nevertheless, the vast majority of calls do not disclose any vulnerability and only two exposed locations were identified (see ENP-01-007 and ENP-01-008).

Positively, a multitude of assessment areas withheld to testing scrutiny admirably. The encryption implementation appeared to be properly implemented, since only strong ciphers and block modes are deployed. The same viewpoint applies to the PRNG implementation utilizing OpenSSL.

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

The self-implemented secure memory and secure string implementation also appeared strong whilst under testing duress, since no weaknesses in this area were identified. Furthermore, the usage of this implementation was deemed soundly composed with no associated risk to report. Additionally, the current vault implementation, as well as the uploader functionality offering uploads to several cloud storage providers, also garnered a strong impression on the whole with no weaknesses in these areas detected.

In conclusion, the Enpass Windows client made a solid security impression in general. However, the identified vulnerabilities and miscellaneous issues across all WPs outlined in this report underline the essential requirement for improvement in several areas of the solution. This conclusory outcome also highlights that the Enpass team handles sensitive customer information due diligently, since no *Critical* or even *High* severity-rated vulnerabilities in this regard were identified.

Moving forward, Cure53 recommends recurrent security assessments against the Enpass Windows client, ideally at least once a year and/or prior to the rollout of significant framework alterations. This proven approach will ensure that both existing vulnerabilities and issues are sufficiently addressed, as well as ensure that newly-introduced functionalities cannot incur fresh vulnerabilities and attack vectors.

Cure53 would like to thank Ankur Gupta, Harsh Valecha, Vinod Kumar, Vivek Singh, and Yogesh Kumar from the Enpass Technologies Inc. team for their excellent project coordination, support, and assistance, both before and during this assignment.