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 06.2022

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

Index

Disclaimer

Please note that this report only covers a *certain* part of the results of the penetration-tests and audits conducted by the audit team against the Enpass software compound.

This report was created & the results were curated based on a request of the Enpass team: It only showcases the **WP1** findings that cover the issues spotted in the Enpass Windows client. Other report documents that contain the findings for **WP2** and **WP3** will be published separately.

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-WP1** - details the scope, results, and conclusory summaries of a penetration test and source code audit against the **Enpass Windows client and UI**. 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.

Cure53 was provided with a binary, sources, pertinent documentation, as well as any alternative means of access required to complete the audit. For these purposes, the methodology chosen was white-box. A team of three 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 achieved comprehensive coverage, identifying a total of three. Two of these findings were categorized as security vulnerabilities, whilst the remaining one was deemed a general weakness with lower exploitation potential.

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.

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

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, 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**
  - **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
- **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.

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.

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

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-01-007 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.

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.

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

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.

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.

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

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*/);
```

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-01-011 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);
}
```

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{
```

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

```
        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** by the Cure53 team - will now be discussed. 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.

Three senior members of the Cure53 team completed the project over the course of several working days from late May into mid-June 2022, achieving comprehensive coverage over the vast majority of components and areas in scope.

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, and documentation 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 software scope. Furthermore, the testing team is pleased to confirm the complete lack of *Critical* or *High* severity-rated findings.

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.