Penetration Test Report

Picocrypt

V 1.0
Amsterdam, September 4th, 2024
Public

Document Properties

Client	Picocrypt
Title	Penetration Test Report
Target	Picocrypt encryption tool (7e403a2)
Version	1.0
Pentester	ROS Auditor
Authors	ROS auditor, Marcus Bointon
Reviewed by	Marcus Bointon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	August 26th, 2024	ROS auditor	Initial draft
0.2	August 30th, 2024	Marcus Bointon	Review
1.0	September 4th, 2024	Marcus Bointon	1.0

Contact

For more information about this document and its contents please contact Radically Open Security B.V.

Name	Melanie Rieback
Address	Science Park 608 1098 XH Amsterdam The Netherlands
Phone	+31 (0)20 2621 255
Email	info@radicallyopensecurity.com

Table of Contents

1 Executive Summary

1.1 Introduction

Between August 19, 2024 and August 28, 2024, Radically Open Security B.V. carried out a code audit for Picocrypt.

This report contains our findings as well as detailed explanations of exactly how ROS performed the penetration test.

1.2 Scope of work

The scope of the penetration test was limited to the following target:

- Picocrypt encryption tool (7e403a2)

The scoped services are broken down as follows:

- Code Audit (incl.PM/Review): 3 days
- **Total effort: 3 days**

1.3 Project objectives

ROS will perform a code audit of Picocrypt with support of the developer in order to assess its security. To do so ROS will analyse the design and architecture of the application, as well as review the implementation.

1.4 Timeline

The security audit took place between August 19, 2024 and August 28, 2024.

1.5 Results In A Nutshell

During this crystal-box penetration test we found 1 Moderate, 3 Low and 2 N/A-severity issues.

We did not find any major issues in the design and architecture of the encryption process. We made two observations regarding the design, they do however not impact the security or confidentiality of the application or data encrypted with Picocrypt.

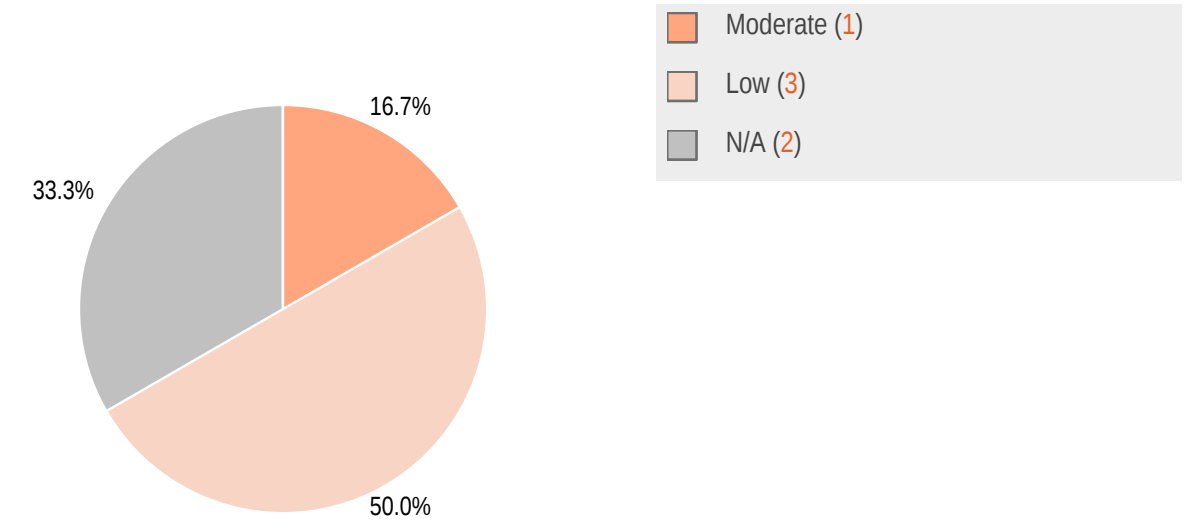
Missing checks for the RNG running out of entropy PCC-003 (page 9) has a moderate severity as it has the potential to compromise the confidentiality of encrypted data, but a low chance of occurring. Three low-severity issues include the Picocrypt file header missing authentication PCC-001 (page 10), the file header can contain a negative length value

PCC-002 (page 11), and observing that Picocrypt decrypts data before checking that it's not been modified PCC-004 (page 11).

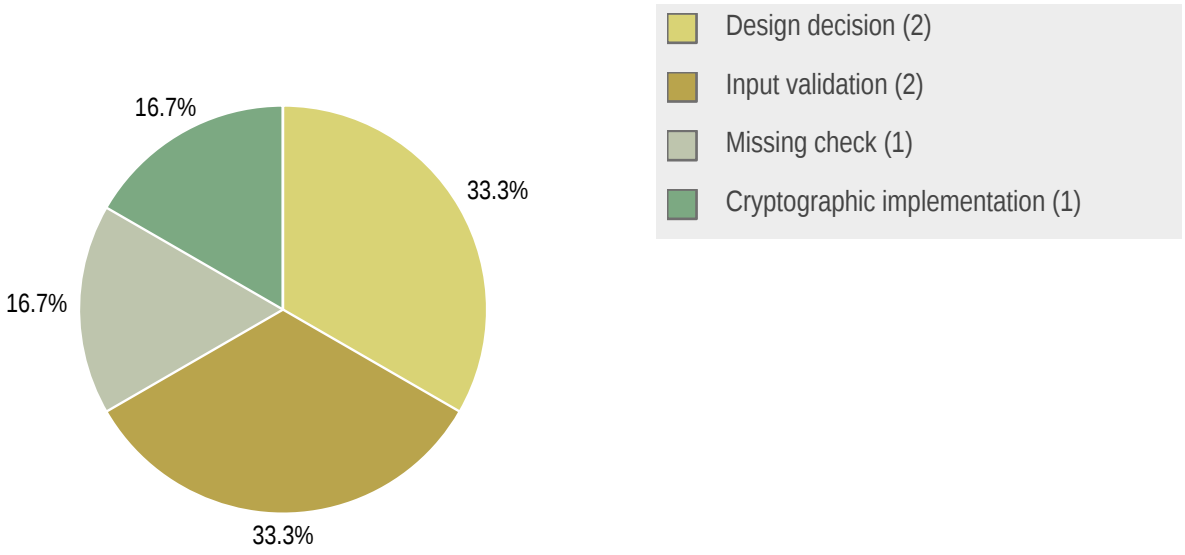
1.6 Summary of Findings

ID	Type	Description	Threat level
PCC-003	Missing check	The application correctly uses the random number generator, supplied by Go's crypto package. The Rand.Read() will also return an error if something is wrong. Unfortunately this error value is not used, allowing for improperly initialized cryptographic primitives.	Moderate
PCC-001	Design decision	Data encrypted with Picocrypt is stored in a custom file format, including a header which is unauthenticated. As a result, any changes made by an attacker would go undetected.	Low
PCC-002	Input validation	When parsing the length of the comment to an integer, it is not checked for negative values.	Low
PCC-004	Cryptographic implementation	Picocrypt offers both encryption and authentication. When a user wants to decrypt a volume, it starts with decrypting and only verifies the signature afterwards. As a result a user may unknowingly use their private key on attacker-controlled material.	Low
PCC-005	Input validation	When encrypting a file, the user can enter a comment to be stored in the volume. If this comment is longer than 99,999 characters, the volume becomes corrupted, and can no longer be decrypted.	N/A
PCC-006	Design decision	Picocrypt uses a memory-hard key derivation function to limit brute force attacks on the password. The output is then hashed using SHA3-512 and stored in the header, to verify the key before attempting to decrypt. As a result there are two algorithms that can be attacked individually which would both lead to the key.	N/A

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



1.7 Summary of Recommendations

ID	Type	Recommendation
PCC-003	Missing check	• Check the error value for each call to `rand.read()`, and cancel all operations if the RNG is not able to supply random numbers.
PCC-001	Design decision	• Authenticate data before processing. Since part of the header is required for key derivation, this is not possible. Adding authentication to the header would allow users to be informed if the header was tampered with (after key derivation, but before decryption). • Ensure that users are made aware that the comment field can be modified by an attacker mid-flight.
PCC-002	Input validation	• Sanitize and validate user-supplied input, and check bounds based on the context.
PCC-004	Cryptographic implementation	• Authenticate the ciphertext before decrypting it.
PCC-005	Input validation	• Check whether the comment exceeds the maximum supported length.
PCC-006	Design decision	• Do not store the hash of the key in the header. • Verify the signature before attempting to decrypt the volume. • Consider replacing the hash with a MAC of the header.

2 Methodology

2.1 Planning

Our general approach during penetration tests is as follows:

1. **Design Review**
   During this phase we analyse the design and architecture of the encryption process. The aim is to find any cryptographic oversight that could compromise the security or confidentiality of Picocrypt.
2. **Code Audit**
   After establishing that the design is sound, we will evaluate if the application correctly implements the design. This is done by auditing the source code provided to us by the developer.

2.2 Risk Classification

Throughout the report, vulnerabilities or risks are labeled and categorized according to the Penetration Testing Execution Standard (PTES). For more information, see: http://www.pentest-standard.org/index.php/Reporting

These categories are:

- **Extreme**
  Extreme risk of security controls being compromised with the possibility of catastrophic financial/reputational losses occurring as a result.
- **High**
  High risk of security controls being compromised with the potential for significant financial/reputational losses occurring as a result.
- **Elevated**
  Elevated risk of security controls being compromised with the potential for material financial/reputational losses occurring as a result.
- **Moderate**
  Moderate risk of security controls being compromised with the potential for limited financial/reputational losses occurring as a result.
- **Low**
  Low risk of security controls being compromised with measurable negative impacts as a result.

3 Findings

We have identified the following issues:

3.1 PCC-003 — Ignored error value when reading random value

Vulnerability ID: PCC-003

Vulnerability type: Missing check

Threat level: Moderate

Description:

The application correctly uses the random number generator, supplied by Go's `crypto` package. The `Rand.Read()` will also return an error if something is wrong. Unfortunately this error value is not used, allowing for improperly initialized cryptographic primitives.

Technical description:

Several cryptographic primitives used by Picocrypt require cryptographically secure random numbers (e.g. key generation, hash-based key derivation, the nonce for Chacha20). To generate these values, the random number generator (RNG) from the `crypto` package is used, which depends on the correct RNG depending on the platform.

The function will write the random output to the array supplied to the function call, and returns the number of bytes written and an error value. The application does not use this error value. In the unlikely event that the RNG is not able to supply the random numbers, the application would not be able to detect it.

Impact:

During key generation, a failed RNG would result in a trivial key (all 0's), potentially leading to loss of confidentiality. For the nonce used by chacha20, this could result in plaintext recovery if a nonce is reused with the same key/password. For key derivation, a fixed nonce of all 0's could allow the generation of a look-up table.

While this would have a large impact on the security, the chance of this happening are very small. As a result the overall impact is rated as Moderate.

Recommendation:

- Check the error value for each call to `rand.read()`, and cancel all operations if the RNG is not able to supply random numbers.

3.2 PCC-001 — Picocrypt volume header is unauthenticated

Vulnerability ID: PCC-001

Vulnerability type: Design decision

Threat level: Low

Description:

Data encrypted with Picocrypt is stored in a custom file format, including a header which is unauthenticated. As a result, any changes made by an attacker would go undetected.

Technical description:

The Picocrypt file's header contains parameters required for decryption, as well as a field for comments. While the encrypted data does have an authentication tag, the header does not. As a result an attacker can change parameters used for decryption or key derivation, as well as the comment field.

Impact:

The impact is quite limited, but extra care should be taken when writing software that parses potentially untrusted data. Decrypting data using attacker-controlled data is a prerequisite for certain attacks (e.g. side channel attacks).

Recommendation:

- Authenticate data before processing. Since part of the header is required for key derivation, this is not possible. Adding authentication to the header would allow users to be informed if the header was tampered with (after key derivation, but before decryption).
- Ensure that users are made aware that the comment field can be modified by an attacker mid-flight.

3.3 PCC-002 — Unchecked input while parsing the header

Vulnerability ID: PCC-002

Vulnerability type: Input validation

Threat level: Low

Description:

When parsing the length of the comment to an integer, it is not checked for negative values.

Technical description:

Picocrypt allows users to add a comment to an encrypted volume. The comment and its length are stored as a string in the header. This string stores the length of the comment in 5 bytes, as ASCII characters. Before trying to read the comment field, it tries to convert this string in to an integer using `strconv.Atoi()`.

The result of this conversion is not checked to be in a valid range, before allocating an array of that length. A modified value with a negative comment length, will result in the application trying to allocate an array with a negative length.

Impact:

Luckily the choice of programming language limits the impact, as Go does not implicitly cast signed to unsigned integers and checks bounds before allocating the array. As a result, the application panics when encountering a volume with a negative comment length.

Recommendation:

- Sanitize and validate user-supplied input, and check bounds based on the context.

3.4 PCC-004 — Decryption of unauthenticated data

Vulnerability ID: PCC-004

Vulnerability type: Cryptographic implementation

Threat level: Low

Description:

Picocrypt offers both encryption and authentication. When a user wants to decrypt a volume, it starts with decrypting and only verifies the signature afterwards. As a result a user may unknowingly use their private key on attacker-controlled material.

Technical description:

When decrypting a volume, Picocrypt will read the input file in blocks of 1 MiB. Each block is appended to the state of the Message Authentication Code (MAC) function, and then decrypted. Only after all blocks have been processed is the calculated MAC compared to the MAC stored in the volume.

Impact:

It is possible for an attacker to modify an existing volume, and present this to a user. When a user attempts to decrypt this with their password, they can process a large amount of attacker-controlled data using their secret keystream. Decrypting data using attacker controlled data is a prerequisite for certain attacks (e.g. side channel attacks).

For the user there would be no way of knowing that this attack is underway until after all data has been processed and Picocrypt informs them that “the input is modified or damaged”.

Recommendation:

- Authenticate the ciphertext before decrypting it.

3.5 PCC-005 — Comment input field does not check comment length

Vulnerability ID: PCC-005

Vulnerability type: Input validation

Threat level: N/A

Description:

When encrypting a file, the user can enter a comment to be stored in the volume. If this comment is longer than 99,999 characters, the volume becomes corrupted, and can no longer be decrypted.

Technical description:

The Picocrypt volume stores the length of the comment as a 5-character string. This means that the maximum length of the comment is limited to 99,999 characters. When entering a comment, this condition is not checked.

If the comment is longer than the limit, the application will still store the comment, but sets the length to 0. When decrypting, the application assumes that the comment length is indeed 0, and will confuse the comment with the encrypted data that follows it.

Impact:

This finding does not impact security, and only renders the encrypted volume unusable. Therefore, the impact is rated to be N/A.

Recommendation:

- Check whether the comment exceeds the maximum supported length.

3.6 PCC-006 — Hash of key stored in the header

Vulnerability ID: PCC-006

Vulnerability type: Design decision

Threat level: N/A

Description:

Picocrypt uses a memory-hard key derivation function to limit brute force attacks on the password. The output is then hashed using SHA3-512 and stored in the header, to verify the key before attempting to decrypt. As a result there are two algorithms that can be attacked individually which would both lead to the key.

Technical description:

To derive a key from the user-supplied password, Picocrypt uses Argon2id. This hash-based function is intentionally designed to be heavy on memory and does not greatly benefit from Application Specific Integrated Circuit (ASIC) acceleration. As a result it is a good choice for converting low-entropy passwords to high-entropy key material.

After derivation, it also stores the SHA3-512 hash of the 256-bit key in the header. SHA3 is not memory-hard, and can be accelerated easily using ASICs or GPUs. (N.B. 256-bit is widely considered to be secure as it is still far out of reach of even the most powerful machine).

As a result there are two paths to attacking the encryption (Argon2, and SHA3), expanding the attack surface. If ever a vulnerability or more efficient attack (than bruteforce) were to be discovered in either algorithm, it would compromise the security of data encrypted using Picocrypt.

Impact:

As both Argon2 and SHA3 are well studied and widely regarded as secure, this is a purely theoretical issue, so this finding is rated N/A.

Due to the claims that Picocrypt offers protection “even from three-letter agencies like the NSA”, we deemed it valuable to mention this risk, however small or unlikely it may be.

Recommendation:

While a purely theoretical issue, do not store the hash of the key in the header, especially since the added value of the stored hash is very limited.

To verify whether the correct key is used, the application should verify the signature *before* attempting to decrypt the volume (as recommended in PCC-004 (page 11)).

Another possibility would be to replace the hash with a Message Authentication Code (MAC) of the header. This would require verification of only 199 + C bytes, and would also resolve PCC-001 (page 10).

4 Future Work

- **Retest of findings**
  When mitigations for the vulnerabilities described in this report have been deployed, a repeat test should be performed to ensure that they are effective and have not introduced other security problems.
- **Regular security assessments**
  Security is a process, and this penetration test is just a single snapshot; security must be continuously evaluated and improved. Regular audits and ongoing improvements are essential in order to maintain control of your corporate information security.

5 Conclusion

We discovered 1 Moderate, 3 Low and 2 N/A-severity issues during this penetration test.

No major issues were found in the design and architecture of the encryption process. We made two observations regarding the design, but they do not impact the security or confidentiality of the application or data encrypted with Picocrypt.

During the code audit four issues were found in the implementation of the application. Three were of low impact. One finding as a moderate impact as it has the potential to compromise the confidentiality of encrypted data but a low chance of occurring.

We recommend resolving the implementation-related findings as they do not require major overhaul. We recommend adding additional authentication to the header to ensure that the header has not been tampered with, as well as verifying the correctness of the supplied key, allowing removing the hash of the private key.

Judging by the documentation, code, and comments, it seems that the developer is well aware of common cryptographic pitfalls. When encountering limitations of a specific primitive they chose simple and robust ways of dealing with them, instead of introducing complexity by trying to work around limitations. Overall the design and implementation have left a good impression.

We recommend fixing all of the issues found and then performing a retest in order to ensure that mitigations are effective and that no new vulnerabilities have been introduced.

Finally, we want to emphasize that security is a process – this penetration test is just a one-time snapshot. Security posture must be continuously evaluated and improved. Regular audits and ongoing improvements are essential in order to maintain control of your corporate information security. We hope that this pentest report (and the detailed explanations of our findings) will contribute meaningfully towards that end.

Please don't hesitate to let us know if you have any further questions, or need further clarification on anything in this report.

Appendix 1 Testing team

ROS Auditor	The auditor for this assignment is a senior security specialist specializing in embedded devices, chip security, and connected devices. This includes both design reviews, code reviews, and (hardware) penetration testing. During their career they have gained a thorough understanding of cryptography, extracted keys using side-channel analysis, and bypassed security measures by injecting faults into chips. They have also delivered training courses on these topics.
Melanie Rieback	Melanie Rieback is a former Asst. Prof. of Computer Science from the VU, who is also the co-founder/CEO of Radically Open Security.