Audit-Report Bitbeans StreamCryptor 04.2015

Cure53, Dr.-Ing. Mario Heiderich, Dipl.-Math. Franz Antesberger, Jann Horn

Index

Introduction

“You can use StreamCryptor to encrypt and decrypt files without size limit and the need to load every file completely into memory. StreamCryptor uses FileStream to read and write files in chunks, there is also an asynchronous implementations for progress reporting available.“

From https://github.com/bitbeans/StreamCryptor

This report describes the Source Code Audit (SCA) against the StreamCryptor library[1] maintained by Christian Hermann. The SCA was carried out by three testers of the Cure53 team in late April 2015 and took two days to complete. The scope of the test essentially comprised the latest Github-stored snapshot of the library, with an inclusion of all files and documentation offered. The audit yielded two security vulnerabilities and eight general weaknesses.

Evidently, the overall count of issues indicates a positive result. No major flaws were spotted in the cryptographic implementation and primitives, while the only serious flaw resulted from improper validation of file paths leading to a potential directory traversal and arbitrary file-write attack. This bug should be trivial to address and fix.

[1] https://github.com/bitbeans/StreamCryptor/

Scope

- **Bitbeans StreamCryptor Source Code**
  - https://github.com/bitbeans/StreamCryptor

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in a chronological order rather than by their degree of severity and impact, which is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier *(*e.g. *BSC-01-00X)* for the purpose of facilitating any future follow-up correspondence.

BSC-01-004 Directory Traversal Vulnerability via *outputFolder* Argument *(High)*

The variants of the method `DecryptFileWithStreamAsync()` receive an `outputFolder` argument and decrypt data into a given folder are affected by a path traversal vulnerability. The methods pass an unfiltered path-name contained in the encrypted data into `Path.Combine()`. Next, they use the result as a path to which the received file should be moved by using `File.Move()`. This allows an attacker to place files in almost arbitrary file system locations to which the user has write-access. Thus this can potentially lead to remote code execution as quoted below.[2]

> *"If path2 does not include a root (for example, if path2 does not start with a separator character or a drive specification), the result is a concatenation of the two paths, with an intervening separator character. If path2 includes a root, path2 is returned."*

A partial mitigation possibly takes place if the parameter `overWrite` is set to false, which is its default value.

From observations made on the code it transpires that the use of path-names is not intended to be permitted at this location at all. It is therefore recommended to either blacklist all characters returned by `Path.GetInvalidFileNameChars()`, or only whitelist known-safe characters. However, if it is important to support file names in distinct world languages (for instance Japanese), a blacklist might be more appropriate. Keep in mind that the latter solution, i.e. the blacklist, needs to be handled with care.

Further note that it is not necessary to filter the filename for special filenames like `PRN` or `COM1` because `File.Move()` calls `MoveFileEx(..., MOVEFILE_COPY_ALLOWED)`[3], which will not write into "existing files".

Note: The issue has been addressed by the maintainer of the StreamCryptor Library and was verified as fixed by Cure53.

[2] https://msdn.microsoft.com/en-US/en-en/library/fyy7a5kt(v=vs.110).aspx
[3] https://msdn.microsoft.com/en-us/library/windows/desktop/aa365240%28v=vs.85%29.aspx

BSC-01-006 Unauthenticated FilenameNonce allows Information Leakage *(Low)*

The `FilenameNonce` member of the `EncryptedFileHeader` class is not protected against tampering in any way. In addition, `Filename` is solely protected by a NaCl Secretbox[4] using `ephemeralKey`.

If a Man-in-the-Middle (MitM) attacker[5] copies the ciphertext of the last chunk (which has to be very short) into `Filename` and the nonce of the chunk into `FilenameNonce`, respectively, then the filename of the decrypted file will in fact be the same as the contents of the plaintext chunk. It is therefore recommended to include at least `FilenameNonce`, and ideally also `Filename`, in the header checksum.

Note: The issue has been addressed by the maintainer of the StreamCryptor Library and was verified as fixed by Cure53.

BSC-01-007 Same Key is used for MAC and SecretBox *(Medium)*

StreamCryptor uses `ephemeralKey` for both message authentication (using `Sodium.GenericHash.Hash()`) and encryption (using `SecretBox`).

It is strongly recommended to make `ephemeralKey` larger and split its use. One half of it can be employed as a key for message authentication, while the other half as a key for encryption. Otherwise a class of cryptanalytic attacks on the combination of *Blake2b* and *XSalsa20* could become possible.

Note: The issue has been addressed by the maintainer of the StreamCryptor Library and was verified as fixed by Cure53.

[4] http://nacl.cr.yp.to/secretbox.html
[5] http://en.wikipedia.org/wiki/Man-in-the-middle_attack

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid attackers in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

BSC-01-001 README.md listing an outdated, deprecated Library Location *(Info)*

The *protobuf-net* library was moved from Google Code to Github, [6] but is still listed under the old location in the project's *README.md* file.[7] It is recommended to update the document to reflect the new location. This would prevent users from adopting a deprecated and potentially vulnerable library.

Note: The issue has been addressed by the maintainer of the StreamCryptor Library and was verified as fixed by Cure53.

BSC-01-002 Redundant Header Data is being used *(Low)*

Chunks contain `ChunkNumber` and `ChunkNonce` members that encompass the number of the chunk and a derived nonce value.[8] These member variables are not required for correct parsing of the encrypted file. In all actuality, StreamCryptor ignores them while decrypting the file.

Nevertheless, the presence of these members in the encrypted data might lead future re-implementations of StreamCryptor towards the use of the chunk number or chunk nonce contained in the chunk header instead of counting chunks themselves. Such behavior would allow attackers to mix the order of encrypted chunks, and therefore also the plaintext contained within them. It is therefore recommended to remove `ChunkNumber` and `ChunkNonce` for the purpose of maintaining clarity and unambiguity.

Similarly, file footers contain unused members such as `ChunkCount` and `OverallChunkLength`. A removal of `ChunkCount` and `OverallChunkLength` as well as the nonces that protect them (`FooterNonceLength` and `FooterNonceCount`) from the footer is recommended. It is desirable not to encrypt their length and count prior to passing onto the MAC function in `SetFooterChecksum()` and `ValidateFooter-Checksum()`.

Note: The issue has been addressed by the maintainer of the StreamCryptor Library and was verified as fixed by Cure53.

[6] https://github.com/mgravell/protobuf-net
[7] https://github.com/bitbeans/StreamCryptor/blame/master/README.md#L8
[8] http://en.wikipedia.org/wiki/Cryptographic_nonce

BSC-01-003 Outdated libsodium Version is used in Build Process *(Info)*

The utilized implementation of the .NET library libsodium.net uses an outdated library core, namely the version libsodium 1.0.0. It is important to note that potentially security-critical fixes of bugs were implemented in the newer versions, especially when it comes to a leap made between the 1.0.0 and the 1.0.1 versions of libsodium. The documentation[9] states that:

> *"NaCl's donna_c64 implementation of curve25519 was reading an extra byte past the end of the buffer containing the base point. This has been fixed."*

The library's dependencies should be updated to address this properly.

Note: The issue has been addressed by the maintainer of the StreamCryptor Library and was verified as fixed by Cure53.

BSC-01-005 ChunkLength is not authenticated and not needed *(Low)*

The member variable `ChunkLength` of the class `EncryptedFileChunk` is not being authenticated properly, neither through the use of the `ChunkChecksum` (as most of the other important members are), nor in any other way. This means that the `overallChunkLength`, which is computed in `DecryptFileWithStreamAsync()` and later passed to the method `ValidateFooter- Checksum()`, is more or less useless from a security perspective.

Moreover, the value of `overallChunkLength` is also only hashed together with `chunkCount`, which would securely indicate tampering on its own, and thus makes `ChunkLength` apparently superfluous. It is recommended to either remove `ChunkLength` from the code or, if it is intended to be some sort of extra security or non-security measure, include its value in `ChunkChecksum`.

Note: The issue has been addressed by the maintainer of the StreamCryptor Library and was verified as fixed by Cure53.

BSC-01-008 MACs use no explicit Type Prefix *(Low)*

`HeaderChecksum`, the `ChunkChecksums` of different chunks and `FooterChecksum`, have no explicit type prefixes in the hashed data.

It is recommended to add explicit prefixes to the hashed data. This would ensure that all checksums are only valid in the places where they indeed belong to. It would additionally prevent an attacker from gaining the ability to tamper with data by copying one of the checksums over another one. For example, the tampering could mean that one could prepend the byte 0x00 to the hashed data for the `HeaderChecksum`, prepend the byte 0x01 to the hashed data for all `ChunkChecksums` and prepend the byte 0x02 to the hashed data for the `FooterChecksum`.

[9] https://github.com/jedisct1/libsodium/releases

Note that in the current design, it already seems impossible for an attacker to swap around checksums without being noticed. However, an explicit type prefix would make this more robust and future-proof in case changes are made to the protocol later.

Note: The issue has been addressed by the maintainer of the StreamCryptor Library and was verified as fixed by Cure53.

BSC-01-009 Unchecked Length Additions in ArrayHelpers *(Medium)*

Both exposed `ConcatArrays()` methods in the class *ArrayHelpers* add unknown array lengths without doing so within a `checked {}` block.

Presently both these cases are not exploitable as the first function first adds up the new numbers using `arrays.Sum()`, which performs computations in a `checked` block, and the second function only adds two numbers, then using the result as array length (note that negative numbers are rejected there).

Nevertheless, it is recommended to perform computations which prevent any potentially attacker-controlled `ints` in `checked` blocks. This way the attackers could no longer take advantage of exploiting integer overflows.

Note: The issue has been addressed by the maintainer of the StreamCryptor Library and was verified as fixed by Cure53.

BSC-01-010 GetRandomString is slightly skewed by an off-by-one Error *(Info)*

The method `Utils.GetRandomNumber()` falsely uses 1 instead of 0 as the lowest allowed number. It also employs the parameter as an exclusive upper-bound. This means that the call `GetRandomNumber(53)` in `Utils.GetRandomString()` returns a number in the range of 1 to 52 (inclusive), and the characters '0' and 'Z' can never appear in the generated strings.

It is recommended to let `GetRandomNumber()` return a random number in a range starting with zero instead of one and call it as `GetRandomNumber(53)` in `GetRandomString()`.

Note: The issue has been addressed by the maintainer of the StreamCryptor Library and was verified as fixed by Cure53.

Conclusion

The StreamCryptor library has proven itself strong and robust - a toolkit that truly did not yield any severe cryptographic issues in this audit. The only spotted attack scenario of high impact is essentially a path traversal caused by improper validation of filenames. This is an exceptionally positive outcome. Once the fixes for this one particular problem as well as the rest of the identified issues are remediated, nothing stands in the way of recommending StreamCryptor to be used in real world crypto scenarios.

Cure53 would like to thank Christian Hermann for this interesting project as well as continuously good support and assistance during this assignment.