CashFusion Security Audit

Final Report, 2020-07-29

FOR PUBLIC RELEASE

Contents

1 Summary

Saint Bitts, LLC have been working on CashFusion, an extension of the privacy protocol CashShuffle for Bitcoin Cash in which each input and output is validated by a random player, using a series of cryptographic commitments that allow an uncooperative participant to be identified and banned.

Saint Bitts, LLC hired Kudelski Security to perform a security assessment of CashFusion, including architecture review, implementation review and code review with a focus on the cryptographic components, to ensure that CashFusion can provide the highest level of privacy and security upon launch.

The repository concerned is:
https://github.com/Electron-Cash/Electron-Cash/tree/cashfusion/plugins/fusion
we specifically audited commit `238b4e0` of the "fusion" plugin, as used by commit `a391bd8` of the Electron-Cash "cashfusion" branch.

Our architecture review was based on the documentation available on:
https://github.com/cashshuffle/spec/blob/master/CASHFUSION.md.

This document reports the security issues identified and our mitigation recommendations, as well as some observations regarding the code base and general code safety. A "Status" section reports the feedback from Saint Bitts, LLC's developers, and includes a reference to the patches related to the reported issues. All changes have been reviewed by our team according to our usual audit methodology.

We report:

- 5 security issues of low severity
- 2 observations related to general code safety

The audit was performed jointly by Dr. Tommaso Gagliardoni – Cryptography Expert, and Yolan Romailler – Senior Cryptography Engineer, with support of Dr. Jean-Philippe Aumasson – VP of Technology.

2 Methodology and Overview

In this security audit, we performed five main tasks:

1. informal security analysis of the original protocol;
2. architecture review;
3. actual code review with code safety issues in mind;
4. assessment of the cryptographic primitives used;
5. compliance of the code with the technical documentation provided.

This was done in a static way and no dynamic analysis has been performed on the codebase. We discuss more in detail our methodology in the following sections.

2.1 Protocol Security & Architecture Review

We studied the protocol and architecture in view of the claimed goals and use cases, and we inspected the provided documentation, looking for possible attack scenarios. We focused on the following aspects:

- possible threat scenarios;
- necessary trust assumptions between involved parties;
- resistance to deanonimization attacks;
- effectiveness of the blame capabilities;
- edge cases and resistance to protocol misuse.

2.2 Code Safety

We analyzed the provided code, checking for issues related to:

- general code safety and susceptibility to known vulnerabilities;
- poor coding practices and unsafe behavior;
- leakage of secrets or other sensitive data through memory mismanagement;
- susceptibility to misuse and system errors;
- safety against malformed or malicious input from other network participants;
- error management and logging.

2.3 Cryptography

We analyzed the cryptographic primitives and components, as well as their implementation. We checked in particular:

- matching of the proper cryptographic primitives to the desired cryptographic functionality needed;
- security level of cryptographic primitives and of their respective parameters (key lengths, etc.);
- safety of the randomness generation in the general case and in case of failure;
- safety of key management;
- assessment of proper security definitions and compliance to the use cases;
- checking for known vulnerabilities in the primitives used.

2.4 Technical Specification Matching

We analyzed the provided documentation, and checked that the code matches its specification. We checked for things such as:

- proper implementation of the protocol phases;
- proper error handling;
- correct implementation of the blame phase;
- adherence to the protocol logical description.

2.5 Notes

It is important to notice that, although we did our best in our analysis, no code audit assessment is per se guarantee of absence of vulnerabilities. Our effort was constrained by resource and time limits, and in the scope of the agreement between Saint Bitts, LLC and Kudelski Security.

The specification of CashFusion is not formal and lots of the implementation details are not being explicitly covered by any reference document.

In assessing the severity of some of the findings we identified, we kept in mind both the ease of exploitability and the potential damage caused by an exploit.

2.6 Conclusions

Overall, we believe that CashFusion addresses an existing problem in managing anonymized transaction in Bitcoin Cash by adopting a reasonable security tradeoff. We observe that the security analysis of CashFusion considers many important aspects, although it can certainly be improved in the future.

The protocol is very complex, which limits the accessibility for a sound analysis and security evaluation. There are some limitations (for example the need to use a trusted server, the strict timing requirements for connecting clients, and limited security assurances regarding combinatorial attacks), but in general we believe that CashFusion offers a practical way to recombine fragmented anonymous transactions in a secure way without the server being able to steal the funds or deanonymize users.

We further believe that the CashFusion codebase provided by Saint Bitts, LLC, that we reviewed, is implementing the protocol as correctly as described and we did not find any evidence of malicious intent or potential backdoor in the codebase.

3 Architecture review

This section reports our observations regarding the general architecture of the Cashfusion system.

This assessment is based on the documentation provided by Saint Bitts, LLC.

Generally, the documents we reviewed focus on technical solutions, and do not document processes, best practices or policies, notably regarding:

- Key generation, handling, back-up and recovery processes. For example, threat model.
- Roles and responsibilities, segregation of duties. For example, between development and infrastructure teams for the server component.
- Management of credentials to access the server. For example, regarding use of 2FA, back-up, password updates.
- Internal IT and risk policies, for example regarding software and hardware management, personnel background check and rotation.
- Incident response processes.

CashFusion is meant to be used as a plugin for users that typically already own a Bitcoin Cash wallet, which explains why the first point above is not overly documented. We do however recommend documenting the 4 other items to ensure transparency and increase trust from CashFusion's user base.

3.1 Documentation completeness

Recommendation

The specification document (in the file CASHFUSION.md) describes each phase of the protocol with a certain level of detail, and discusses some of its security properties and design rationale. This documentation helps users and reviewers understand the general logic, but falls short from being a complete specification, which may for example be necessarily to design formal verification models.

Furthermore, a higher level of assurance could be achieved by providing a thorough security analysis of the protocol under realistic adversarial conditions. Specifically, we would recommend to add:

- A threat model, describing at least succinctly the classes of attackers (in terms of capabilities), and their goals (anonymity compromise, linking, DoS, etc.)
- Refined security analysis of the risks discussed, for example DoS, and how the blame mechanism could potentially be abused.
- More rigorous analysis of the amount linkage risks (the combinatorial arguments is a good start.)
- A description of other operational risks and underlying assumptions for the protocol to operate correctly and securely.

The exercise of working out such documents may in turn reveal overlooked design aspects or unforeseen optimizations.

Status

As the CashFusion project is maturing, Saint Bitts, LLC is planning to polish up the documentation significantly.

3.2 Anticipation of BCP/DRP concerns

The CashFusion server being a critical service to ensure the good operation of the CashFusion protocol, the technical components and processes must anticipate different types of disasters and be designed to facilitate continuity of operations, for

example thanks to preventive measures (the current monitoring and alerting setup) and reactive measures (automated software and services deployment).

The compromise or loss of cryptographic credentials (certificates, authentication tokens, SSH keys, etc.) should notably be considered as a possible scenario.

Recommendation

We recommend that Saint Bitts, LLC defines and tests recovery processes in various scenarios in order to empirically estimate the time to recover and identify overlooked operations. Such scenarios are for example the possible compromise of some access key, the unavailability of part of the infrastructure, the unavailability of critical staff members, and so on.

Depending on the business needs of the service, redundant hardware and software as well as failover service instances and load-balancing might for example be needed.

3.3 Vulnerability management

The cumbersome but necessary process of software and version inventory in the context of vulnerability management is of high importance for the CashFusion architecture, because different parties might reuse the same software version (such as OS and ElectronCash version), and because the server component requires particular trust.

Vulnerability management entails the inventory and ranking of software components impacting the risk posture of the system and processes to proactively monitor the existence of vulnerabilities—such as security audits—as well as processes to learn about public vulnerabilities, and as importantly, processes to remediate to potential security risks in a timely manner that minimizes impact on business operations.

Recommendation

Besides the current security audit, we recommend that Saint Bitts, LLC performs a red team testing exercise modeling insider threat, on a replica of the production system, especially their server component, as well as a penetration test of the said server component.

3.4 Timing enforcement possibly too strict

During switching from one round to another in the CashFusion protocol, in order to keep synchronization with all the clients, the server enforces a minimum time threshold of a few seconds, and bans those players who fail to enter the new round within the given time frame. For example, during connection establishment and switching from Round 1 to Round 2, the server signals to clients that their pool has been filled and they need to move on to the fusion process within 2 seconds. According to the documentation:

> Since clients will expect to receive messages by various times, the server needs to exclude any clients that move too slowly. In order that the various clients' TC are not too dispersed in time, the server should attempt to simultaneously send message 2 to each client at time TS, and kick any client that does not respond with a message 3 (see below) within 2 seconds.

However, keep in mind that most of these clients will connect to the server through Tor to preserve their anonymity, possibly from remote locations or behind layers of VPN, satellite connections, etc. The protocol is assuming a not too asynchronous network, which might be difficult to guarantee in practice for all locations. It might be the case that so few seconds is a too tight time window for the players to complete the switch.

Recommendation

We recommend testing this feature with a variable number of clients connecting from different sources, in different areas with different latencies and networking infrastructures, in order to find a suitable threshold that does not penalize too much a large population of clients.

Status

At each stage there are somewhat tight timing tolerances, which is necessary to make sure all the clients are synchronized so that timing deviations don't reveal too much information. Accordingly, the initial timing is intentionally a bit more tight just in order to wash out any high latency clients before the process gets too far underway.

Although Tor does often have very slow connection times, the latency on an established connection is fortunately not so high, so issues on this point have not been observed so far. That said, most users have not been using Tor for the primary connection at

this stage, Saint Bitts, LLC will keep an eye on this and do some more investigation on the reliability.

3.5 Unclear documentation

The documentation of CashFusion goes into the details of the cryptographic protocol with terminology that is at times hard to follow and without proper references to existing literature. We believe that writing it in a less informal way would substantially improve its readability. A few examples:

- At a first read, it is unclear that there are actually two different types of nonces, one sent by the server and one chosen by the client.
- Key life cycles should be better documented. For example, the role of communication keys is not clear until the blame phase step.
- Certain subprotocols are taken from known literature, but references are not always provided. Some of them are modified in slight (but not necessarily harmful) ways. These modifications, and the rationale behind them, are not always highlighted.

Recommendation

The description of the protocol should be expanded, by including references to existing literature and explaining the rationale behind deviations where necessary.

3.6 On the possibility of combinatorial linking attacks

In the “Avoiding Amount Linkages Through Combinatorics” section of the documentation, it is analyzed the possibility for a malicious observer to link sets of inputs and outputs based on the transaction amount. The authors of CashFusion argue that, for a reasonable amount of players and components, the sheer number of combinations makes finding these combinations impossible.

However, we notice that this is only true for a pure brute-forcing approach. The problem in question reminds vaguely of the problem of *bin packing* where heuristic algorithms much more efficient than brute-forcing are known.

Recommendation

It is not obvious how a direct reduction between the two problems can be found, but we recommend nevertheless checking carefully this security argument against state-of-the-art works found in combinatorics literature.

Status

Although it is expected that finding combinations is quite computationally difficult, CashFusion's security is not relying on this as a first line of defense, but instead aims to be in the regime where there are numerous valid partitions (i.e., even if a perfect algorithm discovered a solution, there would still be no evidence since many other solutions would be possible.)

That said, Saint Bitts, LLC will keep researching the pragmatic aspect of these algorithms, since computational difficulty may help as a defense in depth against linkage, when the number of valid solutions is (for whatever reason) not large enough for a particular transaction.

3.7 Vulnerability to server collusion

The server is a critical component that needs to be trusted not to collude with any player in order to attain the privacy guarantees that CashFusion is aiming for. Notice this is discussed in the Design Trade-Offs section of CashFusion documentation.

Since the linkages between players' commitments are kept by the server, it suffices to have one or more players that have somehow colluded with the server to gain some knowledge of the linkages.

Notice that it is the server who is assigning a new client to a given waiting pool and that will start the fusion when a pool is filled, which mean that a rogue server can easily add the players of its choosing to any pool, thus allowing them to trace transactions in all pools and all fusions by gathering both the linkage information provided to the rogue server and the verification data provided to the rogue players.

This is an inherent design decision and is a security trade-off made by CashFusion.

Recommendation

Do not use any random server to run CashFusion and have "trusted", tested servers available.

4 Findings

This section reports security issues found during the audit.

The “Status” section includes feedback from the developers received after delivering our draft report.

KS-SBCF-F-01: Encryption is misusing AES-CBC

Severity: Low

Description

In encrypt.py, we can see that the encrypted proofs are encrypted using AES-CBC with a zero padding and an all zeros IV:

```
iv = b'\0'*16
if AES:
        ciphertext = AES.new(key, AES.MODE_CBC, iv).encrypt(plaintext)
else:
        aes_cbc = pyaes.AESModeOfOperationCBC(key, iv=iv)
        aes = pyaes.Encrypter(aes_cbc, padding=pyaes.PADDING_NONE)
        ciphertext = aes.feed(plaintext) + aes.feed()
        ↪  # empty aes.feed() flushes buffer
mac = hmacdigest(key, ciphertext, 'sha256')[:16]
return nonce_pub + ciphertext + mac
```

Furthermore it uses the Encrypt-then-Mac paradigm, which usually requires unrelated keys for encryption and authentication.

Notice that the use of an all-zero IV is not a problem if you never reuse the same key, which is the case here since the encryption key is generated on the fly using a Diffie-Hellman shared secret derivation.

Also notice that the Encrypt-then-Mac is done using HMAC, which should be resistant to related key attacks, however this has not been formally proven yet.

These are however poor practices and are not necessarily future-proof.

Recommendation

Using different derived keys for encryption and authentication would be best, derivation could be done using HKDF typically. Furthermore, including the IV in the HMAC would be best, as well as using random nonces as IVs.

Notice that none of these recommendations are required per se to guarantee security in the current state of the codebase. However these would increase maintainability and avoid future misuses shall the code be changed or refactored.

Status

The all-zero IV is an explicit choice that will be better documented in the code.

The use of unrelated keys for the authentication and the encryption will be enforced in the next protocol update.

KS-SBCF-F-02: Default binding to all interfaces

Severity: low

Description

In `fusion/plugin.py:570` the default value for the `bindhost` variable is `0.0.0.0`

This can lead to issues such as CVE-2018-1281, and is not recommendable.

Status

Saint Bitts, LLC acknowledges the issue.

KS-SBCF-F-03: Default server is not using SSL

Severity: low

Description

It appears that the default server is not using SSL, which would be advisable in order to guarantee the highest level of trust:

```
def _get_default_server_list() -> List[Tuple[str, int, bool]]:
        """
        Maybe someday this can come from a file or something.  But can also
        always be hard-coded.

        Tuple fields: (hostname: str, port: int, ssl: bool)
        """
        return [
                # first one is the default
                CashFusionServer('cashfusion.electroncash.dk', 8787, False),
                CashFusionServer('server2.example.com', 3436, True),
        ]
```

Recommendation

Enabling SSL on the server should be done in order to guarantee a higher level of trust and to limit as much as possible the amount of data a passive attacker listening on the network can gather.

Status

This is a work in progress and will be fixed.

KS-SBCF-F-04: Presence of magic numbers that are not NUMS

Severity: Low

Description

Magic numbers are found through the codebase. For example, one can find the following in connection.py:

```
class Connection:
# Message length limit. Anything longer is considered to be a malicious server.
]
↪  # The all-initial-commitments and all-components messages can be big (~100 kB in large fusions).
        MAX_MSG_LENGTH = 200*1024
        magic = bytes.fromhex("765be8b4e4396dcf")
```

And the string `765be8b4e4396dcf` does not appear to be a "nothing-up-my-sleeve" value[1].

[1] https://en.wikipedia.org/wiki/Nothing-up-my-sleeve_number

Recommendation

This is not really a security concern in this specific case but we recommend using NUMS values as magic values, whenever these are required.

Status

Saint Bitts, LLC takes NUMS seriously as can be seen in the Pedersen commitment code on line 41, but here this occurrence is more like a file magic value to identify the protocol and as such is not a security issue as mentioned in the above recommendation.

KS-SBCF-F-05: Non-cryptographic Random is used

Severity: low

Description

In the following places in the codebase, the random package is used to generate randomness:

- in fusion/plugin.py.
- in fusion/covert.py.
- in fusion/server.py.

It appears that it is most notably used in order to shuffle different lists. While not clear from the protocol description, the said shuffling might be important to ensure privacy from an adversary that is tapping the network traffic.

The `random.shuffle` function can take as argument a 0-argument function returning a random float in $[0.0, 1.0[$, however using a float to sample a value below a given max is introducing a bias similar to the so-called "modulo bias", which is no good when used with Knuth Shuffle as it is the case in Python's `random` package. This is not the case in the current codebase, as it is instead relying on the default argument of the `random.shuffle` function, but this is also relying on a floating-point value in $[0.0, 1.0[$ if there is no `getrandbits()` implemented by the `Random` class.

Recommendation

Since Python's `random` is known for not being a cryptographically secure PRNG, we recommend reimplementing Knuth Shuffle using a secure PRNG such as `os.urandom()`.

Furthermore we recommend not importing the `random` package at all in order to avoid any future misuse.

Status

Saint Bitts, LLC acknowledges this and the design rule of not importing `random` will be kept in the future.

5 Observations

This section reports various observations that are not security issues to be fixed, such as improvement or defense-in-depth suggestions.

KS-SBCF-O-01: Using `assert` in production is not recommended

Throughout the codebase, `assert` statements are used to verify certain conditions, such as the fact that the amount is bigger than 0 and other such checks. However, according to the Python documentation, “assert statements are a convenient way to insert debugging assertions into a program”, and they are not run when optimization is requested.

This is not directly an issue, but it might be possible for example that a group of people wrongly request optimization by using a command line provided by a third party, thus disabling many required safety checks. We did not characterize the actual impact of running the codebase in optimized mode.

Notice this issue is inherent to the ElectronCash wallet - which in some cases specifically relies on the assert behavior

KS-SBCF-O-02: PEP 8 or another coding style is not enforced

The fusion plugin does not appear to enforce a specific coding style. In order to increase readability and thus maintainability and auditability, we recommend enforcing a coding style on the codebase.

6 About

Kudelski Security is an innovative, independent Swiss provider of tailored cyber and media security solutions to enterprises and public sector institutions. Our team of security experts delivers end-to-end consulting, technology, managed services, and threat intelligence to help organizations build and run successful security programs. Our global reach and cyber solutions focus is reinforced by key international partnerships.

Kudelski Security is a division of Kudelski Group. For more information, please visit https://www.kudelskisecurity.com or https://kudelski-blockchain.com/.

Kudelski Security
Route de Genève, 22-24
1033 Cheseaux-sur-Lausanne
Switzerland