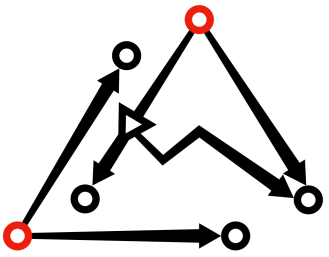
Onion Routed Cloud (ORC)
Final Security Audit Report

Dead Canaries

Report Version: 5 April 2019



Least Authority
PRIVACY MATTERS

Table of Contents

This audit makes no statements or warranties and is for discussion purposes only.

Overview

The Onion Routed Cloud Project has requested Least Authority perform a security audit of Onion Routed Cloud (ORC), a piece of software designed to support and protect journalists and activists by providing an anonymous, distributed, and censorship resistant cloud.

ORC is managed through a comprehensive sponsorship by Dead Canaries, a registered 501(c)(3) tax exempt nonprofit, volunteer-based organization.

The audit was performed from January 30 - February 15, 2019, by Dominic Tarr and Meejah. The initial report was issued on February 21, 2019. A final report has been issued following the discussion and verification phase on April 5, 2019.

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of ORC followed by issue reporting, along with mitigation and remediation instructions outlined in this report. The following code repositories are in scope:

Specifically, we examined the ORC Git revision:

`98e536dbbbc5412e67eeaacfa09143db6a042404`

We also examined the main dependency, Kadence, at the Git revision:

`a5160c51e62b246ace69228c222b6ca5a93d8ade`

All file references in this document use Unix-style paths relative to the project's root directory.

Areas of Concern

Our investigation focused on the following areas of concern:

- Primary Areas of Concern:
  - Identify any vulnerabilities that could expose the identity of the user, censor or destroy data, compromise secrets.
- Secondary Areas of Concern:
  - Identify any vulnerabilities or shortcomings in network architecture that could undermine the ability of the network to function properly, like DDoS, Sybil, etc.
  - Identify any issues in the way that ORC makes use of the Tor network.
- Anything else as identified during the initial analysis phase

Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

This audit makes no statements or warranties and is for discussion purposes only.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present, creating Issue entries, and for each we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Responsible Disclosure

Before our report or any details about our findings and suggested solutions are made public, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for resolution that balances the impact on the users and the needs of your project team. We take this agreed timeline into account before publishing any reports to avoid the necessity for full disclosure.

Findings

Code Quality

The code is well organized, and written in a very modular style. This is can be advantageous for maintenance of the code base, however, this can also be a disadvantage in that it increases indirection

This audit makes no statements or warranties and is for discussion purposes only.

and can result in obscure and uncertain behavior. Due to the code being spread across many plug ins, it is not always obvious what the behavior or intended functionality will be.

Issues

We list the issues we found in the code in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Issue A: DoS Vector via Scrypt in Kadence Eclipse Plugin	*Resolved*
Issue B: Peers That Requested Particular Data Can Sometimes Be Statistically Identified	*Unresolved*
Issue C: Revealing Available Bytes On Storage Nodes Could Deanonymize Trusted Storage Peers	*Resolved*
Issue D: Proof of Work Identities Is Only Weak Deterrent To Eclipse Attacks	*Unresolved*
Issue E: Eclipse Attack Much Easier Than Expected	*Resolved*
Suggestion 1: Give Explicit Deployment Instructions for One OS	*Unresolved*
Suggestion 2: Outline Deployment Decision Trade-offs	*Unresolved*
Suggestion 3: Use Unix Sockets	*Unresolved*

Issue A: DoS Vector via Scrypt in Kadence Eclipse Plugin

Synopsis

Kadence uses scrypt as a sybil mitigation, however, due to it being an symmetric work function, it introduces a DoS vector.

Impact

An instance of this would make easy an occurance of denial of service to a particular Kadence node. Although this does not currently affect ORC because it is not using the eclipse plugin, this also means that the existing implementation of ORC does not have resistance to eclipse attacks.

Preconditions

Attacker connects to node and sends many http requests with random identities.

Feasibility

High.

Technical Details

When a peer sends any message, the message includes their identity. A valid identity contains a proof of work (to make sybil attacks more difficult) and this proof of work is then checked by the receiving peer. Scrypt is a memory-hard symmetric work function. A single run of the scrypt algorithm takes a given amount of memory (which is set high enough to discourage sybil attacks), taking many runs of the

This audit makes no statements or warranties and is for discussion purposes only.

algorithm and a single iteration in order to generate a valid identity,, which requires a significant amount of resources (especially memory). Therefore, a DoS attack can be mounted by sending random invalid identities (merely random numbers, taking little to no effort to generate) but the receiver uses a lot more resources to perform a single run of the scrypt algorithm; thus, the attacker can use a small amount of local resources (sending fake messages) and the receiver must use significantly more to invalidate the messages.

Mitigation

Switch to a symmetric work function that has less validation resources - such as a sha256 partial collision. However, this would make a sybil attack easier, since such an algorithm is already used by bitcoin and second-hand mining rigs (which are no longer profitable for mining), giving a motivated attacker an easy advantage over an honest peer.

Remediation

Use an asymmetric work function that is designed for low validation resources. This is an area of active research, but equihash is a good candidate that is currently used by Zcash.

Status

An equihash proof that takes 30 seconds to a minute to generate can be verified in about 0-1 ms. An invalid proof is also verified in 0-1 ms, as a result, sending invalid proofs doesn't make a DoS attack any easier.

Verification

Resolved.

Issue B: Peers That Requested Particular Data Can Sometimes Be Statistically Identified

Synopsis

ORC uses a Distributed Hash Table (DHT) to index both pointers to shares and peers. Peers accept connections from any other peer and respond with a mix of data they collected because of the random position they occupy in the DHT in addition to data they have gathered from asking their own queries. Therefore, peers will respond with some data that can be statistically identified to be data they queried. As a result,practical queries can be mapped to particular identities.

Impact

Potentially reduce privacy.

Preconditions

Attacker must map the network, connecting to all peers, and asking each what data they have.

Feasibility

Medium. Reasonably complex.

Technical Details

In a DHT, peers are organized into a "hash ring", with each peer helping with a proportion of the data which has hashes similar to their identity. Peers connect to and request data from any other peer. When peers request data, they add it to their local cache; in turn,when data is requested from a peer, they respond with anything relevant in their cache. The position of a peer in the hash ring is determined by its identity, thus publicly known, so any data they respond with that is not the responsibility of their section of

This audit makes no statements or warranties and is for discussion purposes only.

the hash ring could also be information they have queried themselves. As a result, statistical evidence can be gathered that a particular peer is interested in particular data.

Mitigation

Instead of responding with cached data, peers could give responses only that fit their range of the DHT. This would mean that their response to queries would not reflect the data they themselves have queried. However, the scalability of DHTs requires data required for popular lookups to be spread throughout the network, otherwise popular content would be a burden on peers in that portion of the hash ring.

Remediation

Ideally, there would be a way to respond to all queries with statistically fixed properties that was unrelated to whether a hash is requested by the current node. One avenue could be to associate a global proof of work with each record andevery peer that requests a piece of data would try to improve the proof. As a result, popular data would end up with a strong proof and unpopular would end up with a weak proof. When a peer responds, they could take into account the proof strength. Since weakly proved data is unpopular, they would in-turn respond with such data only if it is in their portion of the hash ring. Responding with more popular data would also be acceptable if it's more strongly proven, since the proof shows global popularity, rather than something the responding node happens to have.

It is strongly advised to investigate the academic literature on DHT privacy problems.

Status

This issue has not been resolved due to a misunderstanding that was a reasonable result of ambiguous language in the initial report. It is understood that the ORC development team has started a radical redesign, which will likely avoid this issue entirely because the way peers interact will be very different.

Verification

Unresolved.

Further Clarification

When data is requested from a Kadence peer, it either responds with that data (if it has it) or with peers it knows that are closer to that data. In standard Kademlia, peers cache data they have requested and respond with it if asked. While ORC does not respond in this way, it does add new peers to the routing table while looking for data and also responds with those peers if asked for data near them. If a peer has looked for a given piece of data, they'll also know about peers near that data. If those peers are not near it's own bucket, that can be interpreted as a sign that they recently queried for that data. The routing table is not persisted, so this information is only retained while the same process is running.

Issue C: Revealing Available Bytes On Storage Nodes Could Deanonymize Trusted Storage Peers

Synopsis

ORC peers allow other peers marked as trusted to store data with them locally. But peers also advertise the number of bytes of storage they have offered to the network in addition to the number of bytes they have remaining available. Changes in the available amount can be correlated to see that a group of peers have stored portions of the same file and therefore may trust each other.

Impact

Potentially High. This could leak which peer trusts other peers, which should be considered high value information.

This audit makes no statements or warranties and is for discussion purposes only.

Preconditions

Active participation in network.

Feasibility

High. This would only require ,tracking times of changes in storage availability.

Mitigation

Storage availability does not need to be public information. Since peers only allow storage from other trusted peers, they could also provide the availability information only to other trusted peers, instead of to everyone.

Remediation

It is not necessary to know the exact amount of storage available, just whether there is enough to be used. Therefore, storage availability could be hidden. Peers asking to store would simply request and, in the event that the peer storage is full, they can try another peer. If knowledge of availability is useful, then availability information could also be obfuscated by showing only the number of gigabytes available, which would change infrequently. The mitigation could also be applied.

Status

The capacity feature was simply removed.

Verification

Resolved.

Issue D: Proof of Work Identities Is Only Weak Deterrent To Eclipse Attacks

Synopsis

ORC identities have a proof of work using the scrypt algorithm. This is intended to discourage attackers from inserting nodes into arbitrary positions in the DHT (known as an eclipse attack). However, it is not as much of a deterrent as it appears.

Impact

Sybil attacks are still possible, just requiring additional motivation.

Preconditions

An attacker wishing to insert a peer at a particular position in the DHT must generate a number of proofs equal (on average) to the number of buckets per peer.

Feasibility

Potentially high. (Particularly with some Amazon EC2 servers).

Technical Details

To generate a valid identity, a public key is generated followed by a scrypt being generated and checked to see if it meets the specified difficulty. If not, the process is repeated. The default settings are intended to take an average of 1 minute to generate a new valid identity. This identity will effectively be placed into a random bucket.

This audit makes no statements or warranties and is for discussion purposes only.

Mitigation

Unfortunately this is an open problem. One approach might be to continually improve a proof instead of just a one-off proof, such that an honest peer that has been a member of the network for a long time has built up a quite strong proof, which demonstrates it has been in the network a long time.

Status

This is an open problem that affects a number of protocols that use proof of work on identities.

Verification

Unresolved.

Issue E: Eclipse Attack Much Easier Than Expected

Synopsis

The Kadence eclipse plugin intends to use a proof of work to prevent eclipse attacks (creating nodes that appear near a targeted “eclipsed” node) but it appears that eclipse identities can be generated without much more effort than an honest node.

Impact

Continued vulnerability to eclipse attacks.

Preconditions

Attacker can write custom script to generate node identity.

Feasibility

High.

Technical Details

Kadence uses a public key (on secp256k1 curve) as the node identity. Using the bitcoin BIP32 standard, keys are generated at indexes followed by one iteration of the scrypt algorithm being calculated with the public key as an input. If the scrypt output is within the required difficulty setting, the identity is considered valid. If not, the key at the next index is generated and the scrypt is repeated.

Each identity is mapped into a bucket. The object of an eclipse attack is to create many nodes that are mapped into a particular bucket and to interfere with the operation of that bucket thereafter (e.g. suppress distribution of data assigned to that bucket).

When generating an identity there are two stages. First, generating the key and, second, checking the proof of work (scrypt). To generate an eclipse, it is desired that the attacker would have to perform the full process for many identities until they get one that falls into the bucket they want to target. However, the bucket of an identity depends only on the key and not the scrypt, therefore, an attacker can generate many keys and discard those not in the target bucket. As a result, generating an eclipse node is nearly as easy as generating an honest node.

If:

K is the effort to generate a key (quite low, can likely generate tens of thousands of keys per second);S is the effort to validate one scrypt (quite high, test used in this case was 20 per second); B is the number of buckets (on average, you need to generate B keys to find one that is in a particular bucket, by default 160);

This audit makes no statements or warranties and is for discussion purposes only.

and D is the number of scrypts that must be attempted on average before finding one meeting the required difficulty.

Then:

An honest node uses (K+S)*D effort to generate a single identity in a random bucket (generates a K, and a scrypt D times).

In the current kadence, the bucket depends only on the key. Therefore, an eclipse attacker can use (K*B + S)*D effort to generate a key in a target bucket (generate B keys with little effort, for D scrypts, nearly the same as an honest node).

Ideally, it should take (K+S)*B*D (generate a key and a scrypt for every bucket, D times).

Remediation

Either keep the PoW output around and determine the bucket from the PoW or calculate the PoW from the key and a nonce, and keep the nonce. The advantage of the latter suggestion is the identity would not get much bigger (an 8 byte nonce is likely sufficient).

Status

The eclipse plugin now does an equihash of the public key and a nonce, as a result, the identity (node position in the ring) is now the hash of the equihash proof. The equihash proof contains hashes of the public key so the identity is based on both the key and proof.

Verification

Resolved.

Suggestions

Suggestion 1: Give Explicit Deployment Instructions for One OS

Synopsis

It may be possible to “fingerprint” the OS of an ORC system via TCP techniques. An adversary could then narrow their search to locate ORC nodes containing target shards (for example, knowing a particular group of journalists favour a particular OS).

Mitigation

It would be best if all ORC nodes presented identical network behavior. In order to get as close to that as possible, giving explicit instructions for one operating system seems optimal. Furthermore, Debian/Ubuntu support “apt-transport-tor” as already mentioned in the ORC README but making this “mandatory” would help keep ORC nodes consistent. Specifying an update schedule should also be considered (the ORC protocol advertises its version, so an adversary might be able to identify when particular nodes get updated).

Status

The ORC development team has noted that this suggestion has been considered, but will not be incorporated due to the need for ORC to be accessible to everyone and, as a result, it cannot be reasonably expected that every journalist has the ability to run Debian/Ubuntu support on their machine.

This audit makes no statements or warranties and is for discussion purposes only.

Verification

Unresolved.

Suggestion 2: Outline Deployment Decision Trade-offs

Synopsis

ORC can be deployed on many different systems and in different ways. It would be useful to provide users with discussion about “recommended” ways of deploying ORC and the trade-offs.

Mitigation

As one example, an organization could deploy an ORC node and give each journalist access to it (i.e. shared passphrase) or they could deploy an ORC node per journalist. The latter would be more work, but might have benefits for “disaster recovery” if, for example, a journalist’s devices are seized and thus the passphrase + ORC node is considered compromised.

Status

The ORC development team has indicated that this suggestion will be incorporated in the future.

Verification

Unresolved.

Suggestion 3: Use Unix Sockets

Synopsis

ORC currently uses a local-only TCP listener to set up the Tor Onion service. Although this is suitable, it would be optimal to use Unix-domain sockets instead (where supported).

Mitigation

Using a unix-domain socket, it is impossible to misconfigure the socket in such a way that a remote attacker can learn about the listening ORC daemon. With a loopback-only TCP listener, a cross-protocol attack on any other TCP service on the machine may be able to confirm there is an ORC listener (or even establish two-way communications).

Status

The ORC development team has indicated that this suggestion will be incorporated in the future.

Verification

Unresolved.

Recommendations

We recommend that any of the *Issues* and *Suggestions* stated above that remain unresolved without a clear strategy for avoiding vulnerabilities are addressed as soon as possible and followed up with verification by the auditing team. We also recommend that additional security audits be conducted on future development releases to ensure that any potential issues and vulnerabilities are identified, addressed and verified.

This audit makes no statements or warranties and is for discussion purposes only.

Appendix 1: Comments on the Use of Tor

ORC depends on Tor for user anonymity and peer-to-peer connectivity. Tor is widely used and well studied, however, there are some documented weaknesses. In this report, our feedback is based on certain assumptions about Tor which were applied during our investigation. Broadly speaking, Tor provides a good layer of network location anonymity but it is not perfect.

The most immediately surprising weakness in Tor is that it is easily feasible for a service provider (either an ISP or also an institutional network, such as a university) to know which users on their service are connecting to Tor, given that there is only a small number (about 6000) of Tor relay nodes. If a network administrator checks whether a user is connecting to the IP address of a known Tor relay, this reveals that the user is connecting to Tor. It is possible to mitigate this by running secret bridge nodes, but this will be an exceedingly complicated processes for many users.

Sometimes a Tor user can be manually de-anomized, especially if the user has poor opsec practices. "Tor stinks" snowden leak "with manual analysis we can de-anonymize a very small fraction of Tor users" and sometimes a Tor user will have activity that creates artifacts that are linkable between activity protected by Tor and activity that is not protected (for example, correlating Tor with bitcoin transactions).

Since ORC activity has the potential to link together people communicating with each other, we have reported cases where activity from one peer can be linked to another peer.

Tor is also not sufficient protection against a "global passive adversary". That is an adversary who can see all traffic entering and leaving the Tor overlay network and can de-anonymize clients. Since ORC is using Onion services, this is partially mitigated as traffic to an Onion service does not "leave" the Tor network. However, refer to "The Sniper Attack" paper, which discusses using selective DoS to de-anonymize Onion services.

This audit makes no statements or warranties and is for discussion purposes only.