Security Analysis of Monero's Peer-to-Peer System

Jeffrey Hu
Israel Macias
Marcin Jachymiak
Yao E. Siabi

1. Introduction

Every cryptocurrency has an underlying peer-to-peer networking system that provides other pieces of the software with as fair and complete of a view of the network as possible. Nodes in the network need to receive all relevant messages (transactions and new blocks) in a timely manner in order to prevent potential attacks from malicious nodes. Prior work [2, 3] has demonstrated one such attack called an *eclipse attack* on Bitcoin and Ethereum. They were able to take over all incoming and outgoing connections of victim nodes with relatively little computation resources. Cryptocurrencies must defend against this by taking precautions in how they form their peer-to-peer networks.

In this work, we explore the potential for eclipse attacks on the cryptocurrency Monero. Monero is a open-source [1] cryptocurrency that places a greater emphasis on anonymity than most other cryptocurrencies. Implementing the Cryptonote protocol, it uses ring signatures and stealth addresses to obfuscate the sender, recipient, and amount of every transaction by default. Like all cryptocurrencies, it relies on a P2P network to distribute information about its blockchain. We chose Monero specifically because its design has many similarities to Bitcoin, but unlike several other cryptocurrencies, it is not a direct copy of Bitcoin. In this paper, we first present prior work on eclipse attacks and their countermeasures for Bitcoin and Ethereum. Then, we describe Monero's P2P network in detail. Lastly, we highlight the strengths of Monero's system design and propose potential weak spots and countermeasures.

Our work was done by analyzing the Monero source code found on Github with the commit hash `4b728d7dd48584987f53995a141baac4f886f017`.

2. Prior Work

Heilman et al[2] first proposed the eclipse attack and possible countermeasures in 2015. Since then, successful eclipse attacks have been demonstrated against two of the largest cryptocurrencies: Bitcoin and Ethereum. The underlying strategy in these attacks was to first isolate the victim by controlling all outgoing and incoming communication. Then, by filtering the information from the rest of the network, the attacker could distort the victim's view of the blockchain and subvert its computing power for further attacks.

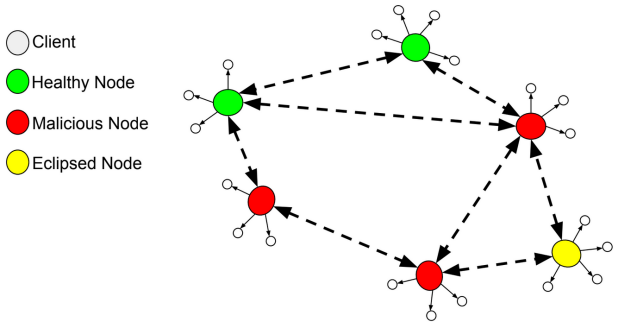

Figure 1. P2P network after an eclipse attack. Victim node's view of the network is provided by malicious nodes.

2.1. Eclipse Attacks on Bitcoin

In Bitcoin, nodes are identified by their IP addresses. Each node randomly selects 8 longterm peers for outgoing connections and accepts up to 117 peers from unsolicited incoming connections. When the attack was first proposed, nodes selected their outgoing connections from a "tried" table and were biased towards newer timestamps. They also published their "tried" table to all outgoing and incoming connections. Bitcoin also stores information about peers it has heard about, but has not connected to, in a "new" table.

The attack was to connect with as many of the victim's 117 available incoming connections as possible and overload it with IP addresses of even more attacker-controlled nodes. This gradually filled all of the victim's incoming connections and their "tried" table. Then, when the victim node eventually restarted, they chose, with high probability, 8 attacker-controlled addresses as outgoing connections. Now controlling all outgoing and incoming connections, the attacker successfully eclipsed the victim node.

To counter this, Heilman et al [2] proposed many solutions, three of which were implemented. In the first implemented solution, he recommended random selection of peers from the "tried" table. This made it more diffi-

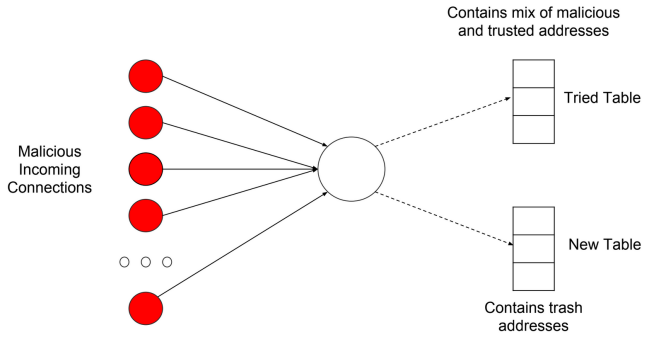

Figure 2. Eclipse attack on Bitcoin, where peer tables are populated with both malicious and trash addresses from malicious incoming connections.

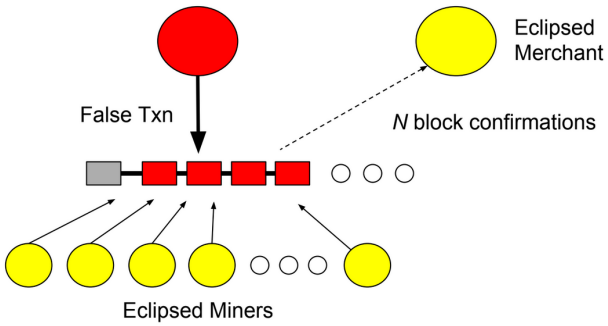

Figure 3. *N*-confirmation double spend. Eclipsed merchant and miners are presented with a fork created by eclipsed miners, containing a transaction they will believe is confirmed. On the "real" chain, the adversary can double-spend the merchant without him knowing.

cult for attackers guarantee their nodes would be selected upon restart. Second, he recommended deterministic random eviction for the tried table. This prevented attackers from completely taking over the tried table by providing IP addresses with the freshest timestamps. Lastly, he recommended simply increasing the size of the peer tables.

2.2. Eclipse Attacks on Ethereum

Since the design of Ethereum's P2P system differs greatly from Bitcoin's, many assumed it to be more secure against eclipse attacks. For starters, Ethereum required thirteen outgoing connections as opposed to Bitcoin's eight. It also used a cryptographically encrypted P2P message scheme instead of direct IP-to-IP communication. This made it more secure against man-in-the-middle attacks and attacks on the internet's routing protocol.

Still though, Marcus et al [3] found Ethereum's P2P system vulnerable to eclipse attacks. Ethereum used node IDs instead of IP addresses to identify nodes. This allowed one machine to run multiple nodes and enabled attackers to mount eclipse attacks on Ethereum with far fewer resources than were required on Bitcoin. Ethereum also made no distinction between outgoing and incoming connections. This allowed attackers to eclipse other nodes exclusively with unsolicited incoming connections.

Implemented countermeasures included modifications to the node identification structure and communication protocol.

2.3. Implications of Eclipse Attacks

The following section describes further attacks that can be performed once nodes have been eclipsed, as shown by Heilman, et al [2].

Selfish-mining By filtering all blocks from the network except their own, an attacker can force eclipsed miners to work on only their version of the blockchain. This increases the attacker's mining power and makes it more likely that their chain becomes the main chain.

51% attacks If the attacker manages to not just eclipse a few miners but fully partition the network, they can guarantee the two partitions never work on each other's chains. Then, the attacker with less than 50% of the compute power only has to out-compete the partitions individually to rewrite the blockchain.

N-confirmation double spend Merchants typically wait for N blocks before considering the transaction as real. If the attacker can eclipsed enough miners to build N blocks on a fork, they can get an eclipsed merchant release goods while double-spending on the real network.

3. Monero's Peer-To-Peer System

In this section we describe how the Monero source code manages its interaction with peers on the network.

Monero's P2P system primarily focuses on managing one type of data-structures: peer-lists.The system maintains a gray-list, a white-list, and an anchor-peer-list. It also uses other system parameters (e.g. the maximum number of incoming connections) that determine some of its behavior.

3.1. Dropping a connection

Periodically, the Monero P2P system may notice unwanted behavior from its peers. This includes: not maintaining a stable connection, not fulfilling a request, sending invalid transactions, sending invalid blocks, or breaking some other consensus rule. To deal with these peers, Monero implements a strict security policy. The system drop connections and can potentially takes further action.

If the peer's failure breaks a consensus rule (e.g. it sends transactions with invalid signatures) then it is assumed that other data sent from the peer may also be invalid. In some cases, the system counts failures for the hostname associ-

ated with that peer and bans the hostname after a certain threshold. In other cases, like a block validation failure, the system deletes the rest of the blocks it has received from that peer and has not checked yet. These measures work to prevent denial-of-service attacks. For an example attack prevented by these mechanisms, consider a malicious node can send many invalid transactions and blocks to its peers with the goal of causing its peers to waste time trying to validate incorrect information.

If the peer's failure seems to take the form of a liveness/availability issue, then the system often gives that node the benefit of the doubt and simply disconnects from the peer without banning it or deleting the data received. One example of such a failure is a RPC request that takes too long to complete.

3.2. Peer-lists

There are two main peer lists that Monero references throughout its source code, very similar to Bitcoin's *tried* and *new* tables that were described earlier. These are the gray and white peer lists.

The gray-list maintains a list of peers who are known, but have never benn connected to. Peers are identified by their IP address and by a `peer_id` which is randomly chosen at start-up. The maximum size of the gray-list is 5000 peers by default.

The white-list maintains a list of online peers. A peer is added to the white-list whenever a successful handshake and connection is established to a non-anchor peer during `gray_peerlist_housekeeping()`.

Finally, Monero also maintains a list of anchor peers. This list contains all the peers to which the node has an outgoing connection to. Connections to these peers are prioritized in the event that the node restarts.

3.2.1 Connecting to a peer

Upon successfully making a connection and completing a handshake with a peer, the P2P system merges the peer's white-list with its gray-list. If the peer was not previously in the node's white-list (or was in the gray-list), then it is added to the white-list.

3.3. Initialization and Startup

Upon initialization the P2P system first loads a hard-coded set of hostnames associated with trusted seed nodes. The system then resolves the hostnames to IP addresses using DNS. If the number of successfully resolved hostnames is too low (less than 12) then a small number of seeds with hard-coded IP addresses are also used as seeds.

(These nodes are presumably run by some trusted third party. Their role is to bootstrap new Monero nodes with list of peers that they can connect to.)

Then the system either loads an existing configuration, or the default configuration.

After choosing an initial set of IP addresses and a configuration, the system starts up 3 threads. One thread simply counts the number of incoming and outgoing connections and maintains those values. Then the system starts an `idle_worker` thread and an `on_idle` thread.

3.4. Handling idle connections

Once a second, the `on_idle` thread drops idle connections. To do so, it iterates through all open connections and calculates the time since it has last received a message from that connection. If the time elapsed is past a certain threshold, it is dropped.

3.5. The `idle_worker`

The `idle_worker` performs the following tasks once every second.

- `peer_sync_idle_maker` sends requests to peers that facilitate the syncing between nodes. In these messages, nodes send a copy of their peer-list, their current blockheight (the number of blocks in their stored blockchain), their current cumulative difficulty (the "amount" of work they see on their chain), and their current version (used to indicate forks for upgrades). This information can be used by the node to request further information from their peer. For example, if its peer's blockheight is greater, a node will request that the peer send it those new blocks. The mechanism for handling peerlists is relevant to the remainder of this project and is described in detail in a section on `gray_peerlist_housekeeping`.

- `store_config` persists all the information kept by the p2p system to a configuration file. This file is used to restore a node's configuration and known peers if it has to restart. When a node restarts, it first rebuilds its peer-lists from this file and then attempts to reestablish its outgoing connections. Once this is finished, the node can function as normal.

- `connections_maker` makes sure that a good number of connections are being made to the network. The connections maker will (in order) attempt to connect to the following types of peers: exclusive peers, seed nodes, priority peers. Exclusive peers and priority peers are not set by default so we ignore them in our analysis. Furthermore we note that there is, by default, a maximum of 8 outgoing connections and no incoming connections. Both of these commands can be tweaked by the node.

  After this step, if the maximum number of outgoing connections is not reached, then the P2P system makes

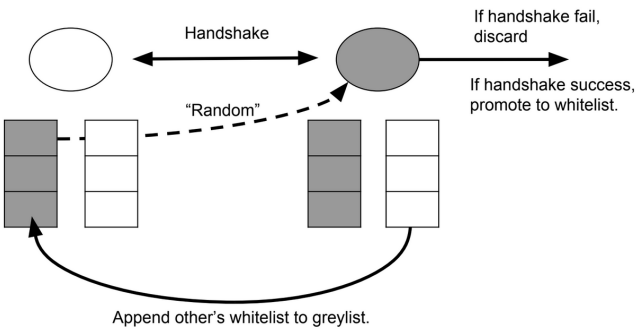

Figure 4. `gray_peerlist_housekeeping` procedure invoked to promote a gray peer to a white peer.

more connections in the following way. First, it checks what fraction of its current outgoing connections are to nodes in its white-list. If it is under an expected fraction (default is 70 percent), then it attempts to connect to anchor peers, then white-listed peers if necessary, and finally gray-listed peers if necessary. Otherwise, if the fraction of outgoing connections with white-list peers is large enough, then the node will try connecting to gray-listed peers, and then white-listed peers. This helps encourage openness in the network and facilitates connections with newer nodes. If at the end of this round of the process, the node has not reached the maximum number of connections, it will try to connect to more seed nodes instead.

- `gray_peerlist_housekeeping` provides a mechanism for a node to grow its white-list by gaining new successful connections. It *randomly* selects a peer from the gray list, and attempts to establish a connection and handshake with the gray list peer. If successful, that peer is promoted to the whitelist. We note that during a successful handshake, the gray peer's white-list is appended to the node's white-list. If the handshake is not successful, the gray list peer is discarded.

  If the either peerlist is full, the entries in it are sorted in order of timestamp of their latest message and peers farthest back in time are removed first.

  While analyzing Monero's source code, we noticed some interesting details regarding peer-list maintenance that could potentially facilitate an eclipse attack. In particular, we investigate the timestamp-based eviction policy and the method of random peer selection for maintenance in our analysis.

4. Potential Vulnerabilities

4.1. Timestamp-based Eviction

We noted that the attacks on Bitcoin's network exploited several weaknesses in its P2P system, one of which was the *bitcoin eviction* mechanism. [2]. When a Bitcoin node's *tried* table was full, 4 random addresses were chosen, and the oldest was evicted. This added bias towards fresher addresses and someone could estimate the probability with which addresses were going to be to be evicted.

In Monero's eviction policy, peers with the oldest timestamps are evicted. This suggests the following:

1. Multiple peers handshaking with the host node can fill the gray peer-list with their own crafted white-lists.
2. Since establishing a connection to a host node adds the peer to the host's white-list, a host's white-list can be populated with nodes that simply form an incoming connection, without the host's permission.
3. By ensuring that a remote node has successfully sent a message a host/public node recently, we can be more certain that their address is contained within the host node's white-list.

4.2. Biased Peerlist-Entry Selection

In `connections_maker` under Section 3.5 above, we describe how the P2P system chooses a peer-list to look for new connections in. Now we describe how it chooses individual entries from a peer-list when it makes a connection. In the case of the gray-list and anchor-peerlist, a index is chosen uniformly at random in the range of indices for that list.

The pseudo-code for that selection looks like the following:

```
crypto::rand<size_t>() % peers_count
```

We simulated this function with a Python script, and as expected the results give a uniform distribution across indices in the given range.

In the case of the white-list however, Monero's code employs a slightly different and unexplained method for choosing a peer. It uses a function `get_random_index_with_fixed_probability` to generate an index. It then sorts the white-list by timestamp, and chooses the value at the "random" index.

The following is pseudo-code for this random-index function:

```
x = rand()%(max_index+1);
rand_index = (x*x*x)/(max_index*max_index);
```

We implemented a Monte-Carlo simulation of this function in Python and show a distribution of the results in figure: 5. We see that the first index is heavily skewed, meaning that a node is much more likely to pick the peer from

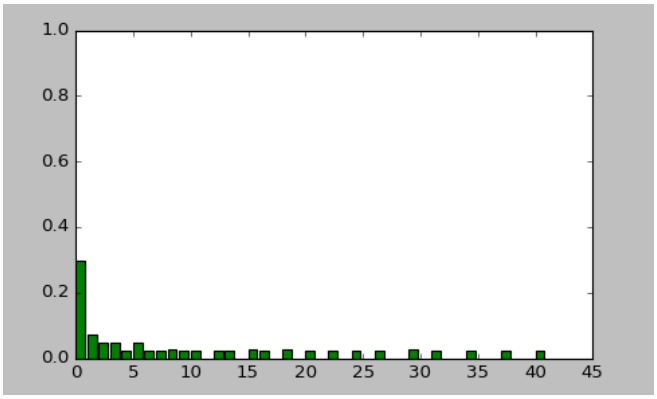

Figure 5. Probability distribution of indices returned with max_index set to 45, after 100,000 trials

its white-list which has most recently sent a message to it successfully.

4.3. Are these vulnerabilities?

Although removing the two features described above was suggested as a countermeasure for the Bitcoin eclipse-attack vulnerability, we suspect the addition of other security mechanisms in Monero decreases the risk created by them. In the next section we describe these security mechanisms, and end with our final analysis of the P2P system.

5. Security Mechanisms

In this section, we describe certain security mechanisms we found implemented in the Monero source-code, many of which were suggested as countermeasures against eclipse attacks by Heilman, et al. [2].

5.1. Feeler Connections

Feeler connections are a eclipse-attack countermeasure which Monero implements, in which the gray-list is periodically cleared of "trash addresses." The P2P system periodically attempts to connect and handshake with a peer on the gray-list. If this process fails, then the peer is removed from the list. This is described in the section on gray-list housekeeping.

This feature decreases the proportion of bad addresses stored in the gray-list, and therefore increases the probability the node will connect to non-malicious peers.

5.2. Anchor Connections

Anchor connections persist upon node reboot. These connections are stored in a separate anchor peerlist where upon restart, a node establishes 2 outgoing connections to nodes in this list. This is the default value in Monero's source code.

Anchor connections reduce an attacker's probability of successfully hijacking all of a victim's outgoing connections as the attacker must be an anchor connection.

This countermeasure is crucial. As long as a node maintains a connection to at least one honest peer, and therefore keeps it as an anchor peer, it will not be eclipsed.

5.3. Limiting Connections by IP and hostname

By default, Monero limits the number of incoming connections from any IP address to 1. This prevents a Sybil-attack, in which an adversary could simply launch many connections from the same IP in hopes of overcoming the victim. The Monero P2P system also stores a list of hostnames it is has banned, and refuses to make new connections with peers with that hostname.

5.4. Limiting Number of Incoming Connections

By default, Bitcoin nodes are limited to 117 incoming connections, which any peer can connect to. This was a critical vulnerability that allows malicious nodes to fill a Bitcoin node's tables with trash addresses. A Monero node, on the other hand, will by default accept *no* incoming connections. For this reason it is very difficult to eclipse non-public nodes as without many open incoming connection slots, it is hard to populate a victim node's peerlists with trash addresses.

6. Results

6.1. Suggested Future Work

Although we provide a detailed description and analysis of Monero's P2P system, there is much we could not do in the timespan of our class project that could have proved useful to anyone interested in the security of this system.

For one, we suggest an experiment to see the potential negative ramifications of timestamp-based eviction. In the Bitcoin eclipse-attack, timestamp-based eviction is abused because attackers can spam a node with many new incoming connections, each sending many trash addresses. In Monero, the grey-list is continually being cleared of trash, and attackers are naturally limited by a default maximum of 8 incoming connections. We suspect it would be useful to run experiments in which we calculate exactly the rate at which peers are cleared from the gray-list, and the rate at which an adversary could flood the gray peer-list with new entries.

We do not believe the biased index generator in section 4.2 is a vulnerability, but we suspect that selecting an element uniformly-at-random may increase the security of the system slightly, with little downside. It removes the possibility of an attacker cleverly spamming a node with messages, so that one of the attacker's addresses is picked with higher probability. Since the probability is really only much

greater for the very first index (sorted by timestamp), this attack is not very effective.

6.2. Conclusion

In this project, we did a thorough analysis of the Monero code base. We analyzed the P2P layer of Monero, collected our findings, and documented the security policies of the system. Then, we used the prior work done on attacks on Bitcoin[2] and Ethereum[3] to look for common vulnerabilities and countermeasures. We largely found that Monero adequately implements several countermeasures to eclipse-attacks.

References

[1] Monero github source code. `https://github.com/monero-project/`.

[2] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg. Eclipse attacks on bitcoin's peer-to-peer network, 2015.

[3] Y. Marcus, E. Heilman, and S. Goldberg. Low-resource eclipse attacks on ethereum's peer-to-peer network. Cryptology ePrint Archive, Report 2018/236, 2018. `https://eprint.iacr.org/2018/236`.