Eclipse Attacks on Monero's Peer-to-Peer Network

Ruisheng Shi*¶, Zhiyuan Peng*, Lina Lan*¶, Yulian Ge*, Peng Liu†, Qin Wang‡, Juan Wang§
* *Beijing University of Posts and Telecommunications, Beijing, China*
† *Pennsylvania State University, Pennsylvania, US*
‡ *CSIRO Data61, Sydney, Australia*
§ *Wuhan University, Wuhan, China*
¶ Corresponding authors: shiruisheng@bupt.edu.cn, lanlina@bupt.edu.cn

***Abstract*—Eclipse attack is a major threat to the blockchain network layer, wherein an attacker isolates a target node by monopolizing all its connections, cutting it off from the rest of the network. Despite the attack's demonstrated effectiveness in Bitcoin (Usenix'15, SP'20, Usenix'21, CCS'21, SP'23) and partially in Ethereum (NDSS'23, SP'23), its applicability to a wider range of blockchain systems remains uncertain.**

In this paper, we investigate eclipse attacks against Monero, a blockchain system known for its strong anonymity and pioneering the use of Dandelion++ (the state-of-the-art blockchain network layer protocol for transaction privacy protection). Our analysis of Monero's connection management mechanism reveals that existing eclipse attacks are surprisingly ineffective against Monero. We accordingly introduce the first practical eclipse attack against Monero by proposing a connection reset approach, which forces the target node to drop all benign connections and reconnect with malicious nodes. Specifically, we outline two methods for executing such an attack. The first one exploits the private transaction mechanisms, while the second method leverages the differences in propagation between stem transactions and fluff transactions under Dandelion++. Our attack is not only applicable to Monero but to all blockchain systems utilizing Dandelion++ and similar connection management strategies.

We conduct experiments on the Monero mainnet. Evaluation results confirm the feasibility of our attack. Unlike existing eclipse attacks, our connection reset-based approach does not require restarting the target node, significantly accelerating the attack process and making it more controllable. We also provide countermeasures to mitigate the proposed eclipse attack while minimizing the impact on Monero. In addition, we have ethically reported our investigation to Monero official team.

I. Introduction

The peer-to-peer (P2P) network forms the foundation of the entire cryptocurrency/blockchain system. It serves as the medium for transmitting information (e.g., protocol messages, transaction data, and block data) between nodes, and ensures the state consistency. The line of research covers a wide range of topics, including but not limited to network-level attacks (e.g., eclipse attacks, network partition) [1][2][3][4][5][6][7], network topology measurement [8][9][10][11][12][13][14], and network layer deanonymization [15][16][17][18][19].

Network and Distributed System Security (NDSS) Symposium 2025
24-28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.240095
www.ndss-symposium.org

Eclipse attack [1] is a method of compromising a node's connections. Through the attack, an attacker can control all the network connections of a target node, gaining complete control over all its incoming and outgoing information. Building upon eclipse attacks, an attacker can engage in more malicious activities, such as identifying the origin of transactions from the target node [15][16][17][18][19][20], disrupting its consensus state [7], and conducting selfish mining [21][22][23].

Previous research on eclipse attacks can be summarized into two categories: outgoing connection occupation methods based on *peerlist* attacks and incoming connection occupation methods based on connection eviction mechanisms. The basic idea behind occupying outgoing connections is to execute a *peerlist* populating attack and wait for the target node to restart. This causes the target node to actively establish connections with the malicious peers in the *peerlist*. Although researchers [1][2] have proposed various *peerlist* populating attack methods, these approaches share common drawbacks: high attack time costs (ranging from hours to weeks) and uncontrollable attack processes (which can only be triggered by restarting the node). The occupation of incoming connections primarily relies on the node's connection eviction mechanism. For instance, the default maximum number of incoming connections in Bitcoin for each node is 114 [24]. When a node reaches this limit and receives a new incoming connection request, it decides whether to evict an existing connection and accept the new one based on Bitcoin's connection eviction strategy. Using this mechanism, an attacker can gradually occupy the incoming connections of the target node by continuously attempting to establish new incoming connections with it [25][26].

However, many blockchain systems (e.g., Ethereum, Monero) lack a connection eviction mechanism similar to Bitcoin's, rendering the previous methods of incoming connection occupation ineffective. In Ethereum, the default maximum number of incoming connections for a node is 34 [7]. Once this limit is reached, all new incoming connection requests are rejected. Thus, the attacker's connection request cannot disrupt the benign connections already established by the node. The attacker has to wait for the original benign connections to disconnect, then occupies the vacant connection slots to gradually occupy all incoming connections [5][7]. In Monero, incoming connections are set to unlimited by default — a deliberate strategy to counter eclipse attacks. Consequently, such nodes unconditionally accept new incoming connection

requests, and the attacker's connection attempts do not impact the existing benign connections. For blockchain systems without a connection eviction mechanism and without constraints on the number of incoming connections, devising an effective eclipse attack remains an unresolved issue.

In this paper, we propose the concept of *connection reset attack*, a new attacking vector overlooked before. Based on this, we present the first practical eclipse attack against Monero. The connection reset attack not only achieves the occupation of incoming connections, but also significantly improves the speed of occupying outgoing connections (shortened to several minutes) and the controllability of the attack process.

The essence of our connection reset attack is to drop the current connections between the target node and all its benign peers, then force the target node to restart the selection process for outgoing connections. We provide two constructions.

The first construction for such an attack (see Sec.III-C1) is to create a specific type of incident that triggers Monero's connection dropping mechanism and to control these incidents occurring at the target node, resulting in its disconnection. The method involves three essential elements: (i) drop the connection, (ii) design triggering incidents, and (iii) control the location where these incidents occur. We use double-spending transaction conflicts as the triggering incidents for the connection dropping. While it is relatively straightforward for an attacker to generate these incidents, they are still ineffective if incidents do not occur at the target node. The challenge thus lies in *how to ensure the incidents occur at a target node*. Our paper represents **the investigation into controlling the location of the triggering incident**.

In the Bitcoin P2P network, the occurrence of a double-spending incident is unpredictable because Bitcoin nodes propagate the transaction to all of their connected neighbors. Double-spending transactions cannot consistently reach the nodes at the times or in the conditions where conflicts are anticipated[1]. However, the adoption of state-of-the-art protocols (e.g., Dandelion++ [27][28]) and new features (e.g., private transactions[2]) in Monero network enables attackers to control the location where double-spending conflicts occur. For instance, private transactions can precisely control the occurrence of double-spending conflicts on the target node's connection because private transactions are only received by the specified target node. By utilizing double-spending conflicts based on private transactions, attackers can achieve a connection reset attack on the target node.

However, a limitation arises from the fact that not all nodes are willing to open the RPC port to offer private transaction services. It is necessary to find more universal solutions.

Fortunately, we observe that every Monero node is obligated to comply with the Dandelion++ protocol. We accordingly devise a second construction (cf. Sec.III-C2) by utilizing the transaction propagation characteristics of the Dandelion++ protocol to **enable the attack to target any node**. The working principle is that **stem transactions only propagate to one outgoing connection** and stem transactions propagate much slower than fluff transactions (deferred to Sec.II-E). The competitive relationship between a pair of double-spending transactions allows the fast-propagating fluff transaction to confine the dissemination of the slow-propagating stem transaction within the range of the target node and a few other nodes. This facilitates the occurrence of double-spending transaction conflicts between the target node and a majority of its neighboring nodes, resulting in a disconnection from these neighboring nodes. Therefore, attackers can control the occurrence of double-spending transaction conflicts at specific locations with a very high probability.

We note that the connection dropping mechanism and transaction privacy protection mechanism are essential components of security mechanisms in all cryptocurrencies. However, these mechanisms are a double-edged sword that can be exploited by attackers to carry out our connection reset attacks.

We summarise our contributions as follows:

- We present the first eclipse attack against Monero. Our attack comprises three major components (Sec.III): the graylist attack, the whitelist attack, and the connection reset attack. The connection reset introduces a new attack vector, for which we provide two practical strategies (i.e., separately via private transactions and Dandelion++):
  - ▷ *The connection reset attack based on private transactions.* We reveal that public service nodes providing RPC services to wallets are vulnerable to eclipse attacks. Since these nodes are responsible for serving a considerable number of wallet users, being subjected to eclipse attacks could pose a significant risk to the transaction security and privacy of a substantial user base.
  - ▷ *The connection reset attack based on Dandelion++.* We have identified a security threat in the form of an eclipse attack that has existed in Monero network since 2020, coinciding with the adoption of the Dandelion++ protocol. None of the nodes within the Monero network is immune to this type of attack.
- We implement and evaluate our eclipse attack on Monero mainnet. Experimental results prove its effectiveness.
- We provide countermeasures that can mitigate our eclipse attack and examine their potential negative effects.

Responsible disclosure: For ethical reasons, we have reported our detected vulnerabilities to Monero official team via Hackerone with the report #2590695 and have received confirmation of the vulnerabilities from Monero official team (pull request #9218).

II. Monero P2P Network

We discuss the network layer behavior of Monero nodes. Our investigation is based on Monero source code, version

[1]The InvBlock attack proposed in the TxProbe (FC '19) paper[10] can achieve similar effects to private transaction, i.e. only one node in the Bitcoin network stores transaction details. However, this vulnerability has been fixed in February 2019 (https://github.com/bitcoin/bitcoin/pull/14897)

[2]Enabling the transaction initiator to restrict the transaction to a single node, thereby preventing the node from forwarding the transaction.

v0.18.3.1 [29] (released on October 11, 2023).

A. Monero Node

A Monero node is uniquely identified by its *(IP: Port)* combination. It establishes TCP connections with other peers using their *(IP: Port)* for communication. By default, a Monero node can initiate outgoing connections with up to 12 different peers. Nodes with a public IP address can receive an unlimited number of incoming connections[3]. Nodes behind NAT are unable to receive incoming connections.

Monero nodes also support Tor and I2P networks. However, as we focus on Monero public network, Tor and I2P are excluded from this analysis.

B. Monero Peer-to-Peer Messages

We introduce several types of messages [30] used in Monero's P2P protocol that are closely relevant to our work.

1) Handshake Message: **The handshake message is used to establish connections between peers.** When a node attempts to establish an outgoing connection with a remote peer, it first sets up a TCP connection with the remote peer using its *(IP: Port)* record and then sends a handshake request. Upon receiving this request, the remote peer responds with a handshake response message, which includes a peerlist field containing up to 250 peer records randomly selected from the remote peer's whitelist. Once the node receives this response, the connection between the node and the remote peer is successfully established (see peerlist creation in Appendix A).

2) PING&PONG Message: **The PING message is used to detect whether the remote peer is able to accept incoming connection, while the PONG message is the response.** When a node receives a handshake request from a remote peer, it will asynchronously send a PING message to the *(IP: Port)* provided by the remote peer in its request. The process of sending a PING message is similar to sending a handshake request. It is worth noting that the node will establish a new TCP connection with the *(IP: Port)* specified by the remote peer and send the PING request through this new TCP channel. When the remote peer receives the PING message, it will respond with a PONG message, confirming that it can receive incoming connections and indicating that the *(IP: Port)* provided in its handshake request is able to communicate.

3) Timed Sync Message: **The timed sync message is used for timed sync process between the node and its neighbors**. The timed sync process is similar to the handshake process. The response to a timed sync request also includes a peerlist field, making it another important method for Monero nodes to obtain new node records.

A node sends timed sync requests to all its neighbors every 60 seconds. The timeout for a timed sync response is set to 120 seconds. If peers do not respond within this timeout period, the node will actively drop the connections with them.

[3]This can be restricted by setting the *in_peers* parameter.

C. Peerlist Management

Peerlist[4] is an important structure for a Monero node to store other peers' *(IP: Port)* records. Each Monero node maintains three types of peerlists: *anchorlist*, *whitelist* and *graylist*.

1) Anchorlist: Anchorlist is an important part of the node's *peerlist* that is primarily used during the startup phase. We exclude the detail here as it does not significantly affect our attack (while more descriptions can still refer to Appendix B.

2) Whitelist: The peer record in the whitelist consists of the peer's *(IP: Port)* and a field named *last_seen*, which represents the timestamp of the node's last communication with that peer. Peer records are typically sorted in descending order by their *last_seen* value. The *last_seen* value reflects the frequency of communication between the node and the peer. Outgoing peers consistently maintain a high ranking in the node's whitelist.

Add records to whitelist. Whitelist records peers that the node has successfully established connections with. When the node successfully establishes an incoming or outgoing connection with a peer and there is no record of the peer in the node's whitelist, the peer's record will be added to whitelist, with the *last_seen* field set to the time when the connection was established. The handshake process is an important way to add new peer records to whitelist.

It should be noted that peers that cannot send PONG messages cannot be added to the node's whitelist. In addition, multiple peer records with the same IP address are not allowed in the whitelist simultaneously. Any subsequent records will replace the previous ones in the whitelist.

A Monero node executes the *gray_peerlist_housekeeping()* function every 60 seconds. The function randomly removes a peer record from the node's graylist and tests its connectability. If the peer is connectable, the peer record is added to the node's whitelist with *last_seen* set to the current time.

Update records in whitelist. For peers that already exist in whitelist, if the node establishes outgoing connections with them, the *last_seen* value for these peers will be set to the latest. The timed sync process can also update out-peers'[5] *last_seen* value. The node updates the *last_seen* value every minute for outgoing connections that successfully respond to its timed sync requests. Because of this mechanism, peer records with the latest *last_seen* values in whitelist always correspond to the node's current outgoing connections.

Monero nodes do not update the records of in-peers in whitelist. Allowing in-peers to update their *last_seen* field would enable malicious peers to continuously refresh this value by repeatedly establishing connections with the node, keeping the malicious peers' records at the top of the node's whitelist. Since Monero nodes are more likely to choose peers at the head of their whitelist to establish outgoing connections, this ability would increase the probability of malicious peers being selected by nodes as outgoing connections.

[4]*Peerlist* in this context is different from the peerlist used in node communications during the handshake or timed synchronization processes.

[5]The term "in-peer" refers to a peer associated with incoming connections, while "out-peer" refers to a peer associated with outgoing connections.

Remove records from whitelist. The whitelist can store up to 1,000 peer records by default. When reaching the capacity, the peer record with the smallest *last_seen* value will be removed.

3) Graylist: The field of peer record in graylist is similar to that of whitelist, which consists of peer's *(IP: Port)* and *last_seen* field. Unlike whitelist, the *last_seen* field of peer records in graylist are always set to 0. The records in graylist are sorted according to the order in which they are added to graylist. graylist is like a queue structure; the earliest peer record is at the tail of the queue.

Add records to graylist. graylist stores peer records in the peerlist shared by other peers. When a node receives the peerlist carried in the handshake response or timed sync response from other peers, it sequentially checks whether the peer records in the peerlist already exist in graylist or whitelist. Existing records will be ignored, and non-existing records will be added to the head of graylist. Nodes do not check whether the peer records are connectable during the process. When a peer record is added to graylist, the corresponding *last_seen* is set to 0 to prevent attackers from populating the graylist by setting the *last_seen* value of the peer records in the peerlist.

Remove records from graylist. graylist can store up to 5,000 node records by default. When graylist exceeds the maximum capacity, the earliest node records added will be deleted. Nodes also randomly delete peer records in graylist through periodic *gray_peerlist_housekeeping()* calls.

D. Connections Management

We summarize the connection management from two aspects: *connection establishment* and *connection dropping*.

1) Incoming connections establishment: By default, the Monero node with a public IP is able to receive an unlimited number of incoming connections, but the node can limit the maximum number by setting its *in_peers* parameter. Since the establishment of incoming connections is passive, the node checks every handshake request from remote peers. If the number of incoming connections reaches the upper limit, new handshake requests will be rejected. Additionally, the node only accepts one incoming connection from a single IP address. Subsequent incoming connection requests from the same IP will be rejected[6].

2) Outgoing connections establishment: A Monero node is able to establish up to 12 outgoing connections by default. The process of establishing outgoing connections depends on the current number of outgoing connections and the status of the node's *peerlist*. We briefly describe the entire outgoing connection establishment process starting from the startup of a new node. We ignore the situations where a node is configured with the parameters *priority_node* and *exclusive_node*.

When a node starts, it will first try to establish connections with peers in *anchorlist*. When establishing a connection fails or there are no available peers in *anchorlist*, the node will instead select out-peers in *whitlist* and graylist. Generally, whitelist has higher priority when selecting out-peers. Only when the number of outgoing connections of the node reaches a certain upper limit[7], or the node fails to establish a connection with peers in whitelist, the node will select out-peers in graylist. When selecting out-peers in whitelist, the node will give priority to the top 20 node records that do not belong to the same /16 IP address domain[8] as the current peers and have a larger *last_seen* value (see Appendix C for the details of establishing outgoing connections).

3) Connections dropping: Various factors may lead to connections dropping. Physical factors, such as network problems of peers, can cause the connection to be interrupted. Generally, connections drop due to inactive behaviors between peers, such as sending blocks or transactions that cannot pass verification (e.g., double-spending transactions).

Additionally, the function *update_sync_search()* periodically drops outgoing connections. Every 101 seconds, the function checks the block synchronization status of the node's outgoing peers. If the node's current number of out-peers reaches the maximum value and the number of out-peers in the synchronization state is less than 2, the node will randomly disconnect one of its non-anchor[9] out-peers.

E. Transaction Propagation

Since version v16.0.0 (released on May 29, 2020) Monero has adopted the Dandelion++ protocol [27][32] to obscure the origin of transactions [16][27][28][32][33]. In Dandelion++, the transaction propagation process can be divided into two stages: *the stem stage* and *the fluff stage*.

In *the stem stage*, transactions are propagated along a secret path in the network. This means transactions in stem stage can only be forwarded to at most one node at a time. This feature limits the propagation range of transactions in the network during stem stage and hides the propagation path of transactions. In *the fluff stage*, each node forwards the transaction to all its neighbors after receiving the transaction. This is similar to Bitcoin's diffusion mechanism. Since transactions in stem stage need to go through several hops before they can turn into fluff stage, their propagation speed in the network is slower than transactions that were initially in the fluff stage.

Similar to how transactions are forwarded, how a node operates depends on which phase the node is in. Each node is governed by two phases: *the stem phase* and *the fluff phase*. Each phase of the node lasts for one *epoch* (about 10 minutes). When an *epoch* ends, the node will enter a new *epoch* and be in *the stem phase* and *the fluff phase* with a probability of 80% and 20%, respectively. When the node is in the fluff phase, it will unconditionally convert any received transaction into the fluff stage and forward it. When the node is in the stem phase, it will forward stem transactions in stem mode and forward fluff transactions in fluff mode. Notably, no matter what phase

[6]see *handle_handshake()*, line 2512 in [31] for details on accepting incoming connection requests

[7]The upper limit is equal to *expected_white_connections*, which is associated with the node's block pruning. See Appendix C for more details.

[8]If the first 16 digits of two IP addresses are the same, they are said to share the same /16 IP address domain.

[9]Anchor peers refer to the out-peers established from the *anchorlist*.

the node is currently in, the node will forward its own original transaction in stem mode.

At each epoch onset, nodes randomly select two out-peers as proxy nodes to relay transactions during the stem phase. Specifically, nodes map each source node to a proxy node, directing all stem phase transactions from a given source to its assigned proxy.

Nodes in the block synchronization stage will not forward the transactions they have received, which affects the transaction spread to a certain extent.

III. Our Eclipse Attack

We introduce our eclipse attack. We describe the three sub-attacks (Sec.III-A, III-B, III-C), and explain the complete process of the eclipse attack (Sec.III-D, Fig.1). We also present a method to detect whether the attack is completed (Sec.III-E).

Threat model. The attack goal of our eclipse attack is to occupy all outgoing connections and drop all benign incoming connections of the target node. The target node is a Monero full node with a public IP and accepts incoming connections from other Monero nodes. The attack resources required by the attacker are 1020 public IPs, of which 1000 are used for whitelist attack and occupation of outgoing connections, and the remaining 20 are used for the graylist attack.

A. Sub-Attack-①: *The Graylist Attack*

The graylist attack aims to populate a large number of trash peer records into the target node's graylist. According to Monero's P2P protocol, the graylist content comes from the peer records contained in the peerlist of handshake/timed sync responses. Therefore, responding to their handshake/timed sync requests is the only and most effective way to populate the target node's graylist.

We populate the target node's graylist by carrying trash node records in the peerlist of the timed sync response. A peer list contains at most 250 peer records, and the default maximum capacity of the graylist is 5,000. If an attacker establishes 20 incoming connections to the target node simultaneously, a single timed sync process is sufficient to send 5,000 trash node records to the target node, effectively filling the graylist.

Given that the graylist operates on a FIFO (first-in, first-out) management mechanism, the attacker must carefully time their responses to the target node's timed sync requests. To maximize the compression of space for benign node records in the target's graylist, the peerlist carrying trash records should be processed after the benign peerlist whenever possible. Thus, we set a delay when responding to timed sync requests.

B. Sub-Attack-②: *The Whitelist Attack*

The whitelist attack aims to populate the target node's whitelist with malicious node records controlled by the attacker. This attack not only requires ensuring that the malicious node records are added to the target node's whitelist, but also necessitates the ability to update the *last_seen* value of these records to increase the likelihood of the malicious nodes being selected for outgoing connections.

To achieve this, we populate the target node's whitelist by establishing incoming connections using malicious nodes. This method is the only controllable way for the attacker to inject malicious node records into the target node's whitelist. For an incoming connection record to be inserted into the target's whitelist and have its *last_seen* timestamp updated to the latest, the corresponding IP address must not exist in the target's whitelist before the connection is established.

If the attacker possesses a sufficient number of controllable malicious nodes with different public IPs, they can orchestrate these nodes to establish incoming connections with the target node in a fixed sequence. If the initial malicious node records are removed from the target's whitelist as subsequent malicious node records are added, these initial records can be re-added to the whitelist, with their *last_seen* value reset to the latest timestamp. When the number of malicious nodes exceed the maximum capacity of the whitelist, the attacker can update all malicious node records in the target node's whitelist, thereby gaining control over its content.

C. Sub-Attack-③: *The Connection Reset Attack*

The connection reset attack exploits Monero's anti-DOS mechanism and privacy protection to trigger connection dropping with double spending transaction conflicts, which forces the target node to establish connections with the malicious nodes injected by the attacker in the whitelist and evict the target node's existing benign incoming connections. According to Monero's outgoing connection selection strategy, a benign node is unlikely to re-establish a connection with the same node after disconnection. Consequently, the attacker can occupy all connections of the target node. We propose two connection reset attack methods. Both attack methods are based on the property that double-spending transactions will cause the connection drops (recall from Sec.II-D3)

1) Based on the Private Transaction: In Monero network, nodes with open RPC services (*closed* by default) mainly refer to those with the *–public-node* parameter set at startup. These nodes provide remote services to Monero wallets, allowing users to create transactions in real-time through RPC interfaces without needing to synchronize the complete blockchain locally. The RPC service also provides developers with an interface to interact with Monero nodes, facilitating data queries and node control [34][35].

The RPC service node provides a method call called *send_raw_transaction()*, which allows users to send the raw transaction generated by the wallet to the node's transaction pool. This method accepts a parameter called *do_not_relay*. When its value is true, the transaction sent to the target node through this call will remain in the target node's transaction pool without being forwarded until the transaction is packaged and chained by the target node. The parameter is a privacy safeguard offered by Monero, initially designed to allow users to add transactions to the blockchain without broadcasting them across the network.

In summary, the connection reset attack based on private transactions operates in four steps:

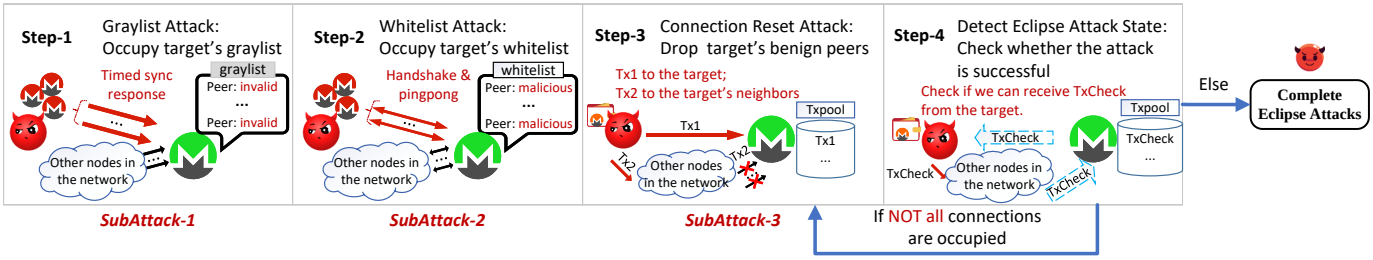


Fig. 1: Our Eclipse Attack against Monero

- Create two offline transactions *tx1* and *tx2* using the same unspent output (i.e., UTXO).
- Call the public RPC method *send_raw_transaction()* to send *tx1* to the target node with the parameter *do_not_relay=True*.
- Broadcast *tx2* to the Monero network.
- Wait for *tx2* to be forwarded to the target node by its neighbors, dropping the target node's benign connections.

Since the target node cannot forward *tx1*, the transaction *tx2* can propagate smoothly throughout the network. After receiving *tx2*, the neighbors of the target node will naturally forward *tx2* to the target node, causing double-spending conflicts between the target and its neighbors.

2) Based on the Dandelion++ protocol: The connection reset attack based on private transaction requires the target node to expose its RPC service, making it not applicable to all nodes. Based on the characteristics of transaction propagation under Dandelion++, we designed another connection reset attack that can be applied to all nodes in Monero network.

Monero uses the Dandelion++ protocol to protect the privacy of transactions during network propagation. In the Dandelion++ protocol, transactions in the stem stage are usually confined to a small area in the network and only a few nodes can receive them. When the transaction turns to the fluff stage, it can be quickly propagated to the entire network[10].

Our attack exploits the difference in propagation speed between stem transactions and fluff transactions. To make the presentation more clear, we first present a basic version of the connection reset attack based on Dandelion++:

- Our proxy node simultaneously sends stem *tx1* to the target node and broadcasts fluff *tx2* to the network.
- The target node first receives *tx1* and then forwards it to one of its outgoing connections, where *tx1* continues to propagate throughout the network according to the Dandelion++ protocol.
- Subsequently, some neighbors of the target node receive *tx2* and forward it to the target node, causing the target node to disconnect from these neighbors.

The basic version of connection reset attack works due to the following reason: A node only relays either *tx1* or *tx2*, whichever arrives first, and is considered *taken over* by that transaction. Among its neighbors, the target node will drop connections with those taken over by *tx2* while maintaining connections with those taken over by *tx1*.

We use the *success_rate* to measure the effectiveness of the connection reset attack:

$$success\_rate = 1 - \frac{benign\_neighbors\_after\_attack}{benign\_neighbors\_before\_attack} \quad (1)$$

where the parameters, *benign_neighbors_before_attack* and *benign_neighbors_after_attack*, represent the numbers of benign neighbors connected to the target node before and after the execution of the connection reset attack, respectively.

Improve the success rate of the basic version. We propose three ways to enhance the success rate.

① **Check whether the target node is in the fluff phase.** If a node is in the fluff phase, any *tx1* we send to it will immediately propagate to its neighbors. These neighbors will then prioritize *tx1* and refuse to accept *tx2*, causing the attack to fail. Detecting whether the target node is in the fluff phase allows the attacker to decide when to initiate a connection reset attack. If the target node is in the fluff phase, the attacker must wait for the next epoch, when the node switches back to the stem phase, before attempting the attack.

An attacker can utilize two proxy nodes, *A* and *B*, both of which are directly connected to the target node. Node *A* sends a stem transaction *tx* to the target node and then checks whether node *B* immediately receives the fluff transaction *tx* from the target node. If node *B* receives the transaction right away, it indicates that the target node is currently in the fluff phase, as a node in the fluff phase will immediately broadcast any received stem transaction in fluff mode.

② **Slightly delay sending the stem transaction *tx1*.** The goal is to allow the fluff transaction *tx2* to propagate to as many neighbors of the target node as possible, while ensuring that *tx1* reaches the target node before *tx2*.

To determine the **appropriate** delay, the attacker follows a specific strategy. Nodes broadcast fluff transactions to all neighboring nodes immediately upon receipt. The attacker can take advantage of this behavior by establishing an incoming connection with the target node. This allows the attacker to measure the propagation time of fluff transactions from the entry node to the target node. By analyzing the collected data, the attacker can accurately calculate the optimal delay time.

③ **Intercept the stem *tx1* through malicious nodes controlled by the attacker** to suppress its propagation. When the attacker's malicious nodes receive the stem *tx1*, they

[10]Stem transactions are forwarded to only one of the 12 outgoing connections, while fluff transactions are forwarded to all neighbor nodes (Sec.II-E).

deliberately refrain from forwarding it. This tactic disrupts the dissemination of *tx1* and increases the effectiveness of the connection reset attack.

The interception becomes even more impactful after completing the first round of the connection reset attack. By then, the attacker has already gained control of some of the target node's outbound connections. During a second round of the attack, if the stem transaction *tx1* is intercepted through these controlled outbound connections (which is highly probable), the success rate is significantly enhanced. If the whitelist takeover rate reaches 100%, the second round will take over all outbound connections.

The final attack we evaluate in the mainnet is the version that implements ALL the above three improvements.

D. Put it All Together: The Complete Eclipse Attack

The first two sub-attacks can commence simultaneously, with each getting re-executed every minute throughout the entire eclipse attack process, triggered by receiving timed synchronization messages. The connection reset attack must wait for the completion of the first execution of the other two sub-attacks, which lasts less than 60 seconds.

We start with an example to describe our eclipse attack and show how the interplay between anti-DoS defense and privacy protection enables our eclipse attack.

An example attack scenario. Victim Alice operates node *A* and initiates her Monero transactions through this node. Given Monero's strong anonymity, Alice assumes that it is impossible for anyone to identify which transactions she has initiated.

Mallory, an attacker, has two objectives: (i) to determine which transactions were initiated by Alice using the eclipse attack, and (ii) to intercept and block Alice's transactions from being confirmed on the blockchain. Mallory has control over 1,020 nodes, consisting of 1,000 nodes with public IPs that can accept incoming connections and 20 nodes that do not require incoming connections. To achieve her goals, Mallory executes the following actions:

Step-① Mallory uses graylist attack and whitelist attack to occupy node *A*'s graylist and whitelist, respectively. Graylist attack and whitelist attack ensure that Mallory occupies the majority of the target node's *peerlist*. These two attacks will get re-executed every minute to maintain Mallory's occupation of the node *A*'s *peerlist*.

After occupying the node *A*'s *peerlist*, the next step in executing a successful eclipse attack is to trigger node *A* to drop the connections with its neighbors.

Step-② Mallory exploits Monero's anti-DOS mechanism to trigger connection dropping with double spending transaction conflicts. Monero relies on an anti-DOS mechanism: Monero disconnects from nodes that send potentially malicious invalid transactions. This serves as a protective measure against the partitioning attack [7] on target nodes.

However, this anti-DOS mechanism can be manipulated by an attacker using double-spending transaction conflicts as a means to trigger connection drops. Although it is relatively easy to generate double-spending transaction conflicts within the Monero network, the challenge lies in precisely controlling where these conflicts occur. Mallory's objective is to force node *A* to drop its connections with existing neighbors, making the primary challenge the ability to control the location of the double-spending transaction conflicts accurately.

Step-③ Mallory exploits Monero's transaction privacy protection mechanism to control where double spending transaction conflicts occur. If Mallory can ensure that only Node *A* has transaction *tx1*, while the rest of the network holds transaction *tx2*, the double-spending conflict can be isolated to Node *A* alone. This requires Node *A* to have its RPC services enabled and support private transactions. In this scenario, Mallory can employ a connection reset attack using private transactions to achieve this isolation.

If Node *A* does not have an accessible RPC interface, Mallory can alternatively use a connection reset attack based on the Dandelion++ protocol to control where the double-spending conflict occurs (see technical details in Sec.III-C).

Step-④ Mallory takes over all of node *A*'s connections. If a single round of connection reset attack fails to take over all of node *A*'s connections, Mallory can launch additional rounds of connection reset attack until they take over all of node *A*'s connections. Since the malicious nodes do not forward transaction *tx2* to node *A* after receiving it, the connections taken over by the attacker will not be disrupted during the additional rounds of connection reset attacks.

E. Detect the Completion of the Eclipse Attack

To detect whether the eclipse attack is completed, a fluff transaction can be sent to the network. If the target node does not send this transaction to the malicious node connected to it, it means that the connection of the target node has been completely occupied and the eclipse attack is completed.

IV. Evaluation of Sub-Attacks

In this section, we focus on evaluating the three sub-attacks (evaluation of the complete end-to-end eclipse attack is deferred to Sec.V). We conducted all our experiments on the Monero mainnet.

Experiment settings. Conducting our attack in a real-world environment requires substantial IP resources (at least 1,000 unique IP addresses). To simulate a large number of nodes while working with limited IP resources, we modified the source code of the target node to relax certain restrictions related to connection establishment and whitelist management[11]:

- we lifted the restriction on *selecting outgoing connections*, allowing the target node to establish outgoing connections with multiple nodes with the same IP address.
- we lifted the restriction on *establishing incoming connections*, permitting the target node to accept multiple connections from the same IP address.

[11]We have conducted a series of preliminary tests on unmodified nodes and obtained experimental results consistent with those on our modified node.

- we lifted the restriction that *nodes with the same IP address cannot coexist in the whitelist*, allowing different node records with the same IP address to exist simultaneously in the target node's whitelist.

We ran the modified node on a server with a public IP and simultaneously ran 1,000 nodes on another server, also with a public IP. Each node used the same IP but different P2P ports. From the perspective of the modified target node, these 1,000 nodes are indistinguishable from peers in Monero network, with each belonging to a different /16 IP address domain. Due to resource constraints, instead of running 1,000 real Monero nodes, we ran 1,000 simulated nodes built on top of the open-source project *py-levin* [36]. Our simulated nodes implement the Monero network protocol and can seamlessly interact with real Monero nodes. Compared to real Monero nodes, simulated nodes consume fewer resources and are manageable.

Configurations. We set up two server nodes with public network IPs, referred to as Server A and Server B, respectively. Both servers have the same physical configuration, which includes an Intel(R) Xeon(R) Platinum 8255C CPU @ 2.50GHz 4C4T, 8GB of memory, a network bandwidth of 12Mbps, and running Ubuntu Server 20.04 LTS 64-bit. We ran our modified Monero node on Server A as the target node and ran 1,000 simulated Monero nodes on Server B as the malicious node resources for the eclipse attack. Additionally, we deployed 20 simulated Monero nodes to launch the graylist attack.

A. Evaluating the Graylist Attack

We set up 20 graylist attack nodes. These nodes established incoming connections with the target node and populated the target's graylist by responding to the target node's timed sync requests with a delay of 4s. The state of the target node before the graylist attack was illustrated as Table I.*Column-I*.

TABLE I: The status of the target node

	before graylist attack	before whitelist attack
Out-peers	12 / 12	12 / 12
In-peers	22	17
Whitelist	1000 / 1000	1000 / 1000
Graylist	5000 / 5000	5000 / 5000
	Column-I	*Column-II*

We conducted a 128-minute attack on the target node and record the contents of the graylist during the attack. The experiment result of the experiment is illustrated as Fig.2a.

As graylist attack began, the number of trash node records in graylist of the target node increased rapidly, while the number of benign node records was squeezed by the trash records and decreased rapidly. Throughout the attack, the number of trash records always remains at a high level. According to statistics, during the attack, the average proportion of the number of trash records in target node's graylist was approximately 80.0%, and the highest proportion was 99.6%, which means almost a complete occupation of the slots in the target's *graylsit*.

The proportion of trash records in target's graylist reflects the filling effect of graylist attack, but the ultimate goal of graylist attack is to prevent the target node from obtaining available benign node records from graylist. In order to evaluate the actual impact caused by graylist attack, we analyzed the target node's behaviors of selecting outgoing connections from graylist and the results of nodes executing *gray_peerlist_housekeeping()* during the graylist attack.

Selecting out-peers from graylist. During the attack, the target node made a total of 184 outgoing connection selections from its graylist. Among them, target node selected trash records 155 times (84.2%), while selected benign node records 29 times (15.8%). It can be seen that graylist attack significantly prevented the target node from obtaining avaliable out-peers from graylist.

Results on *gray_peerlist_housekeeping()*. The target node executed a total of 102 *gray_peerlist_housekeeping()* calls, 28 (27.5%) calls of them selected benign node records in graylist, while the remaining 74 (72.5%) calls selected trash records. Graylist attack also makes it more difficult for nodes to obtain available node records from graylist through *gray_peerlist_housekeeping()* mechanism.

Factors affecting the graylist attack. The number of trash records and benign node records in the target node's graylist fluctuated within a certain range. This variation occurred because, besides the malicious nodes, the target node's benign neighbors also sent peerlists containing benign peer records, which occupied space in graylist and pushed out trash records.

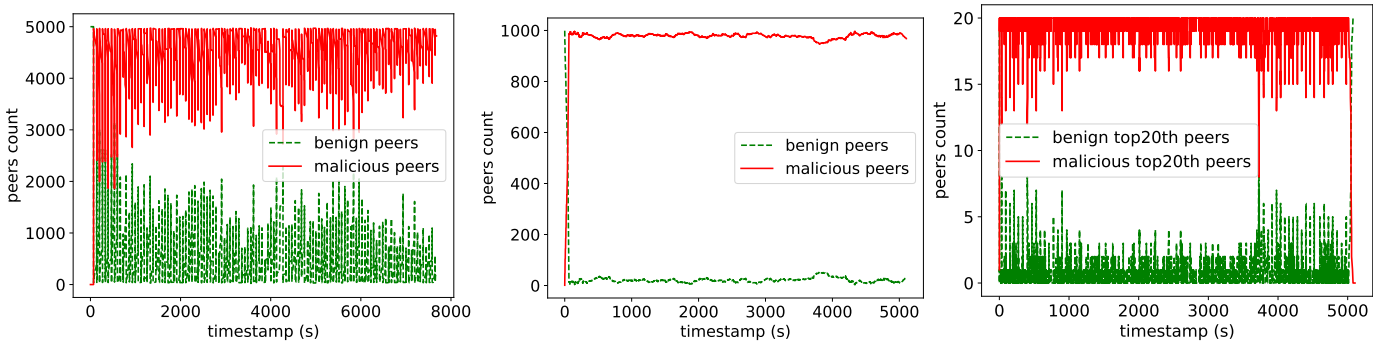
Our observations revealed that most benign neighbors responded promptly to the target node's periodic sync requests. This rapid response was often accompanied by an influx of benign node records, leading to a significant decrease in the number of trash records in the target node's graylist. However, a few nodes responded slowly to the sync requests, causing their peerlists to reach the target node after the attacker's malicious peerlists. This delay partially mitigated the impact of the graylist attack, as the malicious peerlists had already taken effect by the time the slower benign responses arrived.

B. Evaluating the Whitelist Attack

We use another 1,000 malicious nodes to perform the whitelist attack. The state of the target node before the whitelist attack is shown in Table I.*Column-II*.

We performed an 85-minute attack on the target node and recorded the content of the target node's whitelist during the attack. The results are presented in Fig.2b&2c.

Similar to the evaluation method used for the graylist attack, the effectiveness of the whitelist attack is directly assessed by counting the number of malicious node records in the target node's whitelist. As in Fig.2b, the number of malicious node records in the target node's whitelist increases rapidly. The attacker quickly completes the occupation of all the slots of the target node's whitelist. However, a slight decrease in the number of malicious records was observed during the attack. This reduction occurred because the target node completed a timed sync process with its outbound peers, leading to the re-insertion of outbound peer records into the target's whitelist.



(a) Progression of the graylist attack (b) Progression of the whitelist attack (c) Status change of whitelist top-20 slots

Fig. 2: Progression of the graylist/whitelist attack over time

The results presented in Fig.2c support our conclusion. The number of benign node records in the first 20 slots of the whitelist exhibited periodic fluctuations. An increase in the count of benign nodes indicates the completion of the target node's timed sync process, which in turn leads to a reduction in the number of malicious node records. It can be seen that the timed sync process of the target node is an important factor affecting the effect of whitelist attack.

Another factor impacting the attack's success is the establishment of new incoming connections by benign peers. These benign node records, like those of the malicious nodes, do not initially appear in the target node's whitelist. Consequently, they are added to the whitelist in the same way as malicious node records. Analyzing the target node's logs revealed that a total of 238 new incoming connections were established during the whitelist attack. This influx had a slight impact on the overall success of the whitelist filling process.

C. Evaluating the Connection Reset Attack

We evaluate the connection reset attack from two aspects:

- The success rate of connection reset attack;
- The completion time of the attack.

Success rate. It is calculated as the ratio of successfully reset connections to the total number of attempted connections (see Equation 1). During our experiments, we observed that some nodes were inactive, unable to relay transactions or blocks due to insufficient block height or being in a synchronization state. These inactive nodes were excluded from our analysis since they did not affect the effectiveness of our attack.

Completion time. The completion time of a connection reset attack is a key metric for assessing its effectiveness. We define this as the duration from the moment the first benign connection of the target node is reset to the moment the last benign connection is reset. A shorter completion time suggests that the connection reset attack is more timely and efficient.

1) Attack based on private transactions: **The number of nodes in Monero mainnet with open RPC services** (Table II) determines the scope for employing the connection reset attack using private transactions. We utilized a node probing program to identify Monero mainnet nodes with open RPC ports. This probing was conducted from March 12, 2024, to March 25, 2024. Over the course of these two weeks, we detected a total of 1,184 nodes with open RPC ports. On March 25, 2024, we assessed the availability of the RPC services for these nodes. Out of the 1,184 nodes, 735 (62.1%) were found to offer publicly accessible RPC services. Of the remaining 449 nodes, 210 were online but their RPC services were not publicly accessible. The RPC services of the final 239 nodes could not be verified, as they were offline during our testing.

TABLE II: RPC Status in Monero mainnet

The number of nodes with opened RPC ports		
1184 / 100%		
RPC service publicly available	**RPC service unavailable**	**Offline**
735 (62.1%)	210 (17.7%)	239 (20.2%)

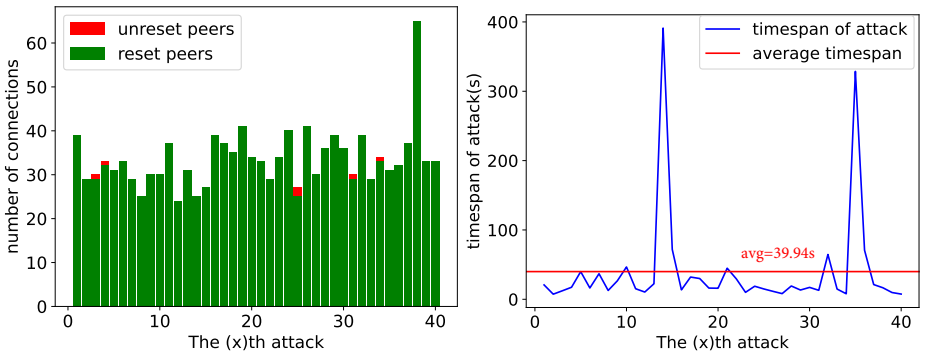
Success rate (Fig.3a). We conducted 40 times of connection reset attack based on private transaction on the target node. Among the 40 connection reset attacks, 35 (87.5%) attacks completed the reset of all connections of the target. The average connection reset rate reaches 99.55%. More importantly, even if an attack does not reset all connections, the connections that are not reset will be reset in subsequent attacks.

Completion time (Fig.3b). We calculated the time span on each attack. Due to the randomness of transactions propagating in the network, there is uncertainty in the time when the double-spending *tx2* reaches the target node through its benign peers. Therefore, the completion time of the connection reset attack fluctuates within a certain range. From the result, the average completion time of connection reset attacks based on private transaction is 39.94s, and 38 (95%) of attacks have a completion time of less than 2 minutes. This means the attack can quickly reset the target node's connections in most cases.

TABLE III: Distribution of fluff tx propagation time span

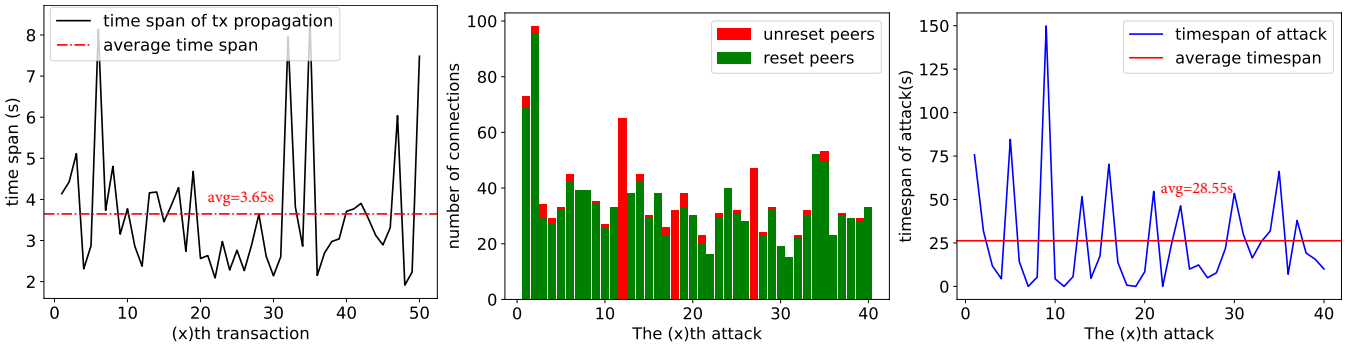
Time span (s)	[0,1)	[1,2)	[2,3)	[3,4)	[4,5)	5≤
Percentage	0%	2%	44%	28%	14%	12%

2) Attack based on Dandelion++: Connection reset attack Based on Dandelion++ cannot limit the target's propagation of the double-spending transaction *tx1*. We need to measure



(a) The number of reset connections (b) The duration of each attack

Fig. 3: Evaluation results for the connection reset attack based on private transaction



(a) Transaction propagation time (b) The number of reset connections (c) The duration of each attack

Fig. 4: Evaluation results for the connection reset attack based on Dandelion++

the time span it takes for fluff transactions to propagate from the attacker's proxy node to the target node to determine the timing of sending these two double-spending transactions.

We set up two proxy nodes named *S* and *F*, respectively. Node *F* is used to send stem stage transaction *tx1* to target node. Node *F* is used to propagate transaction *tx2* to the entire Monero network in fluff manner.

Time span for fluff transaction propagation (Fig.4a & Table III). We used node *F* to propagate *tx2* to Monero network in a fluff manner and recorded the timestamp when a transaction was sent. We also recorded the timestamp when the transaction first arrived at the target node in its log. The difference between these two timestamps indicates the time span it took for this transaction to propagate from the proxy node *F* through the neighbor node to the target node.

We measured the propagation time spans of 50 fluff transactions from the attacker node *F* to the target node. Over 70% of fluff transactions had a time span between 2 and 4 seconds, and 82% had a time span exceeding 2.4 seconds. The average propagation time was approximately 3.65 seconds. The success rate of connection reset attacks is influenced by the selected delay time. To maximize the success rate, we set the delay time to 2.4 seconds. According to our data, this delay allows 82% of connection reset attacks to successfully reset some of the target node's neighbors.

Success rate (Fig.4b). Same as Sec.IV-C1, we performed 40 successful connection reset attacks on the target node. The sending interval of *tx2* and *tx1* was set to 2.4s. Among the 40 attacks, 15 (37.5%) attacks achieved a complete reset of the benign connections of the target node, and 33 (82.5%) attacks reset more than 90% of benign peers. The average connection reset rate reaches 86.60%. Compared with the attack based on private transaction, this attack is inferior in terms of number and stability of connection resets. However, it can be used on almost all Monero nodes.

Completion time (Fig.4c). We calculated the completion time of such an attack. Similar to the connection reset attack based on private transaction, this attack can also be completed in a short period of time, averaging 28.55 seconds. However, due to network issues or the target node's internal reasons, some neighbors did not forward the double-spending *tx2* to the target node in a timely manner, resulting in a longer attack completion time. The occurrence of this situation is relatively low and does not significantly impact the stability of the attack.

Difficulty in resetting all connections. The challenge in resetting all connections of the target node with this method stems from the inherent randomness in the propagation of fluff transactions within the network. When *tx1* is sent to the target node after a specified delay (2.4 seconds in this case), there is a chance that *tx2* has not yet reached some of the target node's neighbors. If *tx1* arrives at these neighbors before *tx2*, the connections between the target node and these neighbors will

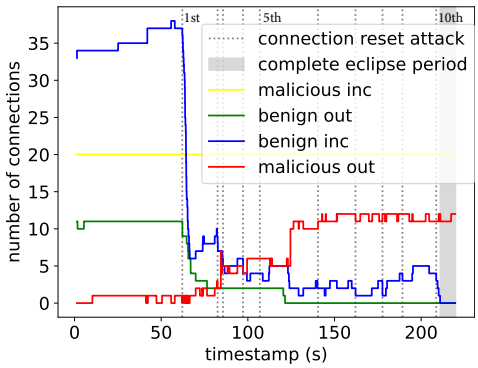


Fig. 5: How the numbers of each type of connections change over time during the complete eclipse attack

remain intact, as these neighbors will reject the later arrival of *tx2* and instead propagate *tx1* back to the target node.

The case of resetting all connections. In this case, the attacker successfully sends *tx1* to the target node. When the target node sends transaction *tx1* to the next neighbor in stem mode, the neighbor it selected has already received transaction *tx2* in advance. The neighbor refuses to receive *tx1*, which makes *tx1* only retained in the target node. Subsequently, all neighbors of the target node receive *tx2* and forward it to the target node, resulting in the reset of all the target's connections.

Failure cases. Out of 40 attacks, 3 of them failed to reset any connections. The primary reason for this outcome is that *tx2* reached the target node before *tx1*, causing the target node to reject *tx1*. Reducing the delay between *tx1* and *tx2* could decrease the likelihood of this issue; however, doing so might prevent *tx2* from fully propagating to the target node's neighbors, potentially diminishing the overall effectiveness of the connection reset attack.

3) Cost of the connection reset attack: The cost of a connection reset attack arises solely from the transaction fee required for the double-spending transactions created by the attacker, with only one of the two transactions being added to the blockchain. Consequently, the cost of a connection reset attack is equivalent to the transaction fee for creating a single transaction. In the current Monero mainnet environment, the fee for initiating a transaction is generally around $0.01, making the connection reset attack low-cost.

V. Evaluation of the Complete Eclipse Attack

We conducted complete eclipse attack experiments on our controllable target node in the Monero mainnet. Besides assessing whether the eclipse attack can successfully take over all connections of the target node, we also evaluated the effectiveness of each individual sub-attack.

Experiment settings. We use 20 simulated malicious nodes to perform graylist attack and another 1000 simulated malicious nodes to perform whitelist attack. Graylist attack and whitelist attack are carried out continuously throughout the eclipse attack. After the graylist attack and whitelist attack are launched, the connection reset attack is continuously executed to occupy the outgoing connections of the target node and continuously evict the incoming connections. We use the connection attack method based on the Dandelion++ protocol. We also use the connection reset attack method based on private transactions to perform eclipse attacks, and the results are in Appendix E.

A. Evaluation Metrics

We evaluate the practicality of the eclipse attack from the following aspects:

1) **[Time to complete eclipse attack]** is defined as the time it takes from the first connection reset attack to occupying all connections of the target node.
2) **[Time to occupy outgoing connections]** is defined as the time it takes from the first connection reset attack to occupying all outgoing connections.
3) **[Graylist/Whitelist occupation rate]** is defined as the average occupation rate of malicious records in graylist/whitelist during the attack.
4) **[Graylist/Whitelist attack blocking rate]** is defined as the ratio of the number of malicious nodes selected from graylist/whitelist to the total number of outgoing connections selected from graylist/whitelist.

B. Results Overview

We performed 10 complete eclipse attack experiments on the target node. Before each attack, the node's peerlist did not contain any malicious node records.

Experimental results are shown in Table IV. *Out-peers* and *In-peers* represent the number of benign connections of the target node before the eclipse attack begins.

TABLE IV: Summary of eclipse attack results

No.	Out peers	In peers	Whitelist / Graylist occupaction rate	Whitelist / Graylist attack blocking rate	Time to occupy out conn	Time to complete eclipse attack
1st	11	22	94.83% / 65.12%	94.87% / 91.67%	66.75s	182.29s
2nd	11	37	97.64% / 70.58%	100% / 95.45%	59.26s	148.73s
3rd	12	17	98.49% / 86.88%	97.62% / 89.29%	76.87s	98.72s
4th	12	19	83.16% / 77.05%	93.94% / 85.48%	35.94s	75.64s
5th	12	20	95.76% / 69.93%	100% / 95.19%	29.31s	45.08s
6th	10	38	96.70% / 84.97%	100% / 93.07%	63.94s	239.68s
7th	12	32	92.84% / 89.52%	100% / 100%	7.97s	329.18s
8th	11	28	99.22% / 83.17%	100% / 88.66%	29.98s	96.34s
9th	12	22	96.79% / 86.35%	100% / 96.83%	115.43s	123.58s
10th	12	21	94.90% / 60.28%	100% / 95.38%	167.12s	220.48s

We observed that the completion time of the eclipse attack varies significantly. In the fastest instance (5th attack), the attacker completed the eclipse attack in just 45.08 seconds, whereas the slowest instance (7th attack) required 329.18 seconds. This variability also applies to the occupation of outgoing connections; the fastest and slowest times were 7.97 seconds (7th attack) and 167.12 seconds (10th attack), respectively. Across all 10 attacks, the average times to fully occupy the outgoing connections and complete the eclipse attack were 65.26 seconds and 155.97 seconds, respectively. Detailed attack progression curves for the fastest and slowest cases can be found in Appendix F.

C. Detailed Discussion

To provide a clearer understanding of the eclipse attack process, we detail the second eclipse attack. This specific attack was selected for analysis because the times required

to occupy outgoing connections and to complete the eclipse attack are close to the average values. Our graylist/whitelist attack lasted a total of 4 minutes, during which we conducted 10 connection reset attacks. By the end of the process, we successfully occupied all of the target node's outgoing connections and eliminated all benign incoming connections.

During this attack, how the numbers of each kind of connections change over time is illustrated in Fig.5.

1) Effect of the graylist/whitelist attack: Throughout the attack, the average occupancy rate of the graylist reached 70.58%. The Top-20 whitelist slots were occupied at a rate of 99.67%, with an overall whitelist occupation rate of 97.64%.

The graylist/whitelist attack greatly reduced the possibility of the target node obtaining available benign nodes from the graylist/whitelist. During the entire attack, the target node tried to select 22 outgoing connections from the graylist, of which 21 selected invalid node records injected by the attacker, accounting for 95.45%; the target node tried to select 16 outgoing connections from the whitelist, all of which point to the malicious node records. From the actual effect, the graylist/whitelist attack greatly increased the probability of malicious nodes being selected as outgoing connections, which played an important role in occupying the outgoing connections of the target node.

2) Effect of the connection reset attack: During the 4-minute attack, a total of 10 connection reset attacks were performed. The first 5 attacks achieved the occupation of outgoing connections, which took 59.26s. The total time to complete the eclipse attack was 148.73s.

Occupation of outgoing connections. During the 1st attack, the majority of the target node's benign outgoing connections were reset, causing a rapid drop from 11 to 2 benign outgoing connections. By the 5th connection reset attack, the remaining 2 outgoing connections were also eliminated.

As the number of outgoing connections decreased, the target node initiated the process of selecting new outgoing connections. Due to the graylist/whitelist attack, the target node's graylist and whitelist were filled with trash and malicious node records, respectively. With a scarcity of connectable benign nodes in the graylist, the target node was forced to establish connections with the malicious nodes listed in the whitelist. Finally, the attacker successfully occupied all 12 of the target node's outgoing connections. From the initiation of the 1st connection reset attack, the entire process took 59.26 seconds.

Expulsion of incoming connections. The 1st connection reset attack also reset most of the benign incoming connections (from 37 to 6). In the subsequent attacks, the number of benign connections remained at a low level (no more than 10). This indicates that most of the benign incoming connections did not try to reconnect to the target node after being disconnected. Therefore, through continuous connection reset attacks, the original benign neighbors gradually stopped establishing outgoing connections with the target node.

After completing the occupation of the outgoing connections, the attacker performed 5 more connection reset attacks, and finally achieved the complete expulsion of the benign incoming connections. From the 1st connection reset attack, it takes 148.73s to complete the eclipse attack.

VI. Further Discussion

A. Resource Requirements for The Whitelist Attack

The capacity of the whitelist is 1,000 entries, and node records with the same IP address cannot be stored simultaneously. Therefore, to fully populate the whitelist, 1,000 distinct public IP addresses are essential (based on our estimates, the cost of acquiring 1,000 IP resources is approximately $400 per month). The primary objective of populating the whitelist is to occupy the outgoing connections of the target node. Given the mechanism by which the target node selects outgoing connections from the whitelist, the distribution of these 1,000 IP addresses is also critical.

1) An ideal case: The target node is more likely to select nodes that are in different /16 IP address domains compared to its current peers. Based on this criterion, we define the ideal scenario as one in which each of the 1,000 IP addresses owned by the attacker belongs to a distinct /16 IP address domain. Our attack simulates this scenario. For the modified target node, all of the attacker's malicious nodes are perceived to originate from different /16 IP address domains, thereby increasing the likelihood of these malicious nodes being selected by the target node for outgoing connections.

2) The worst case: In contrast to the ideal scenario, we define the worst-case scenario as one where all the IP resources owned by the attacker originate from the same /16 IP address domain. When the target node selects outgoing connections, it prioritizes peers that belong to different /16 IP address domains than its current peers, rather than prioritizing the value of the peers' *last_seen* field.

If an attacker uses 1,000 IP addresses belonging to the same /16 IP address domain to populate the whitelist of the target node, the attacker can only occupy all the target's outgoing connections by maintaining a complete populating of the target's whitelist. Once there are benign nodes in the target's whitelist that do not belong to the same /16 IP address domain as the malicious out-peers, the target will prioritize establishing connections with these nodes, regardless of their *last_seen* value. Therefore, any interference with the whitelist attack will result in the attacker not having full control over all outgoing connections of the target node.

3) A trade-off case: Although the effectiveness of the attack under the ideal scenario is superior, it requires the attacker's IPs to be distributed in different /16 IP address domains. The worst-case attack has lower requirements on IP resources, but has poor anti-interference ability.

We consider a trade-off: the attacker has at least 1,000 IP addresses distributed across n different /16 IP address domains, where n is greater than or equal to the maximum number of outgoing connections of the target node. When the target node selects outgoing connections from its whitelist, there must be a malicious node from a different /16 IP address domain than

the current malicious out-peers for the target node to select, ensuring that malicious nodes are prioritized for outgoing connections. The larger the value of n, the closer the attacker's IP resources are to the ideal scenario, making the attacker resemble a botnet attacker. Conversely, the smaller the value of n, the closer the attacker's IP resources are to the worst-case scenario, making the attacker resemble an infrastructure attacker.

B. Why The Connection Reset Attack is Indispensable?

The primary reason is to expedite the occupation of both outgoing and incoming connections.

1) For outgoing connections: Existing eclipse attacks rely on node restarts to trigger the occupation of outgoing connections, which is beyond the control of attackers. Another issue with relying on node restarts is the instability of the attack's effectiveness. If the attacker cannot occupy all outgoing connections of the node through a single restart, they have to wait for the next restart. Restart-based methods incur high time costs and uncertainties. Our connection reset attack, however, places the trigger of the attack in the hands of the attacker, making the attack process controllable and repeatable. This greatly reduces the time cost of the eclipse attack and increases its practicality. The attacker can repeat the attack process until all outgoing connections are completely occupied.

2) For incoming connections: The success of eclipse attacks in occupying incoming connections largely depends on connection eviction strategies (e.g., Bitcoin) and the attacker's persistent attempts to establish connections (e.g., Ethereum), both of which typically entail high time costs. More critically, for Monero nodes with default settings, these methods are ineffective. Connection reset attacks can evict incoming connections, making it feasible to execute eclipse attacks even on nodes with an unlimited number of incoming connections. Without the connection reset attack, the attacker cannot effectively counteract the existing and future benign connections of the target node.

VII. Countermeasures

We propose several potential attack mitigation measures and examine their potential side effects.

A. Against The Graylist Attack (i.e., Sub-Attack-①)

Modify the way of handling timed sync responses. To mitigate the risk of graylist attacks from potentially malicious incoming connections, it is recommended to limit timed sync requests to outgoing connections only, excluding incoming connections. Alternatively, when handling timed sync responses from incoming connections, ignoring the peerlist within the message can prevent the injection of malicious records. However, these adjustments may have unintended effects on other functions within the timed sync process.

Limit the number of node records shared by a connection. The Monero network protocol only limits the maximum number of node records that a single peerlist can carry, but does not limit the maximum number of node records that a node can share within a given period of time. By imposing a limit on the maximum number of node records that can be shared within a specific time frame, the difficulty of executing a graylist attack can be effectively increased. This would require the attacker to use more resources to achieve the same effect as before.

B. Against The Whitelist Attack (i.e., Sub-Attack-②)

Forbid to add in-peer's record to whitelist. This countermeasure can directly invalidate the whitelist attack, but it will slow down the propagation speed of new node record in the network, causing the new node to take a long time to be known by other nodes in the network.

Add the capacity of whitelist. Increasing the default capacity of the whitelist can significantly increase the time and resources required to populate the whitelist and increase the difficulty of the whitelist attack. The capacity of the whitelist can be modified in Monero source code.

C. Against The Connection Reset Attack (i.e., Sub-Attack-③)

Do not drop the connections when encountering double-spending conflicts. Instead of dropping connections, nodes should simply discard double-spending transactions. This approach prevents connection reset attacks and improves the anti-DoS mechanism by avoiding unnecessary disconnections. Unlike other invalid transactions, double-spending can propagate through the network, and the nodes relaying them are often just intermediaries, not the originators. Therefore, simply ignoring these transactions is sufficient. Monero team adopted it in **release v0.18.3.2** (corresponding pull request #9218).

However, this countermeasure is not perfect. It introduces new risks related to network topology leaks. Double-spending transactions are a known method for measuring Bitcoin's network topology, but the technique fails with Monero because the double-spend broadcasting would alter the network topology. This countermeasure enables the attacker to learn topology information by sending double-spending transactions,which can potentially make transaction anonymity analysis [16] possible. Fortunately, no publicly available rapid network measurement method based on double-spending transactions exists so far, yet the privacy subgraph of the Monero network is rapidly changing at every moment. Therefore, the network topology information leakage caused by this mitigation does not pose a substantial security threat [12].

VIII. Related Work

Eclipse and network attacks. Heilman *et al.* [1] proposed an eclipse attack against Bitcoin nodes. By occupying the node list of the target Bitcoin node and waiting for its restart, the attack occupied the outgoing connection of the target node. Tran *et al.* [2] proposed a more subtle eclipse attack called "Erebus" against the Bitcoin network. It takes advantage of an AS-level attacker to intercept and replace the TCP connection between the target Bitcoin node and its neighbors, thereby achieving complete occupation of the target node's

[12]Meanwhile, the Monero team is planning the long-term mitigation against spamming double spends.

connections. Later, Tran *et al.* [3] proposed an integrated defense framework to mitigate the impact of the Erebus attack.

Wust *et al.* [4] proposed an eclipse attack against Ethereum. They exploited the vulnerability of Ethereum nodes and separated the target node from the network without occupying connections. Marcus *et al.* [5] took advantage of the fact that Ethereum nodes do not distinguish between connection types (incoming or outgoing), forced the node to restart through program vulnerabilities, and established a large number of incoming connections, occupying the target node's connections. Heo *et al.* [7] proposed a partition attack called Gethlighting against Ethereum nodes. By controlling about half of the TCP connections of the target node, and sending low-rate DoS attacks to the target node, an attacker can interfere the target node's process of synchronizing new blocks, preventing the target node's block height from growing and separating the target node from the Ethereum network.

More network-level attacks refer to [37][38][39][40].

TABLE V: Network-level attacks

Attack	Targets	Used time	Restart?	Costy?	Occupation?	Controllability	Sustainability	Stealthiness
Eclipse [1]	single Bitcoin node	several hours + URT*	✓	✓	✓	✗	-	✓
Erebus [2]	single Bitcoin node	5-6 weeks + URT*	✓	✓	✓	✗	✓	✓
Sync Att. [25]	Bitcoin network	-	✗	-	✗	✗	✗	✗
Wust's [4]	single Ethereum node	$\sim$ 18 minutes	✗	✗	✗	✓	✓	✗
Marcus's [5]	single Ethereum node	several minutes + URT*	✓	✗	✓	✗	✓	✓
Heo's [7]	Ethereum network	one day	✗	✓	✗	✗	✗	✗
This work	single Monero node	several minutes	✗	✗	✓	✓	✓	✓

✓ Holding such a property (attack resistance), while ✗ vice versa.
* Unpredictable restart time: Node restart time is unpredictable.

Network topology measurements in blockchains. Biryukov *et al.* [15] and Miller et al. [8] used the timestamp of peer records in the ADDR messages to infer the connections between nodes. Cao *et al.* [11] also inferred the topology of Monero nodes based on peer record's *last_seen* value. With the improvement of the node topology protection mechanism, these methods are no longer effective. Grundmann *et al.* [9] proposed a network topology measurement method based on double-spending transactions, which realizes the inference of target connections through the mutual restrictions of double-spending transactions propagating in the network. Sergi *et al.* [10] measured the topology of the Bitcoin network using orphan transactions. However, due to differences in transaction propagation, it cannot be adapted to Monero's network. Saad *et al.* [6] measured the node distribution of Bitcoin and Ethereum in autonomous systems and found that the node distribution is highly centralized. They provided the partition risks with countermeasures. Many related studies also made contributions to similar measurements [13][14][41][42][43][44].

Monero and more. Monero is renowned for anonymity by using ring signatures [45] (distinguishing it from concurrent techniques [46] like ZKP [47][48], commitment [49][50], or TEE [51]). Most related studies focus on its non-functional properties, including accountability [52], traceability [53][54], unforkability [55], construction efficiency [56] and linkability [57]. However, none of them examine its essential components (i.e., network layer) or its fundamental resilience to attacks. In contrast, we focus on the underlying security.

IX. Conclusion

In this paper, we presented the first eclipse attack against Monero nodes with public IPs, demonstrating the ability to occupy all of their connections. We proposed two novel connection reset attack methods, which significantly accelerate the attack process. We conducted experiments on mainnet Monero nodes. Our evaluation shows that an attacker with 1,020+ IP addresses can fully occupy all connections of the target node within just few minutes. We ethically reported our findings to the Monero official team and provided countermeasures.

Acknowledgment We are deeply grateful to the anonymous reviewers for their insightful comments and suggestions. This work was supported by the Beijing Natural Science Foundation under Grant M21037, National Key Research and Development Program of China (2022YFB2702405).

References

[1] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on Bitcoin's peer-to-peer network. In *USENIX security symposium (USENIX Security)*, pages 129–144, 2015.

[2] Muoi Tran, Inho Choi, Gi Jun Moon, Anh V Vu, and Min Suk Kang. A stealthier partitioning attack against Bitcoin peer-to-peer network. In *IEEE Symposium on Security and Privacy (SP)*, pages 894–909, 2020.

[3] Muoi Tran, Akshaye Shenoi, and Min Suk Kang. On the Routing-Aware peering against Network-Eclipse attacks in Bitcoin. In *USENIX Security Symposium (USENIX Security)*, pages 1253–1270, 2021.

[4] Karl Wüst and Arthur Gervais. Ethereum Eclipse attacks. Technical report, ETH Zurich, 2016.

[5] Yuval Marcus, Ethan Heilman, and Sharon Goldberg. Low-resource Eclipse attacks on Ethereum's peer-to-peer network. *Cryptology ePrint Archive*, 2018.

[6] Muhammad Saad and David Mohaisen. Three birds with one stone: Efficient partitioning attacks on interdependent cryptocurrency networks. In *IEEE Symposium on Security and Privacy (SP)*, pages 111–125. IEEE, 2023.

[7] Hwanjo Heo, Seungwon Woo, Taeung Yoon, Min Suk Kang, and Seungwon Shin. Partitioning Ethereum without Eclipsing it. In *Network and Distributed System Security Symposium (NDSS)*, 2023.

[8] Andrew Miller, James Litton, Andrew Pachulski, Neal Gupta, Dave Levin, Neil Spring, Bobby Bhattacharjee, et al. Discovering Bitcoin's public topology and influential nodes. *Accessible https://allquantor.at/blockchainbib/pdf/miller2015topology.pdf*, 2015.

[9] Matthias Grundmann, Till Neudecker, and Hannes Hartenstein. Exploiting transaction accumulation and double spends for topology inference in Bitcoin. In *International Conference on Financial Cryptography and Data Security (FC) Workshops*, pages 113–126. Springer, 2019.

[10] Sergi Delgado-Segura, Surya Bakshi, Cristina Pérez-Solà, James Litton, Andrew Pachulski, Andrew Miller, and Bobby Bhattacharjee. Txprobe: Discovering Bitcoin's network topology using orphan transactions. In *International Conference on Financial Cryptography and Data Security (FC)*, pages 550–566. Springer, 2019.

[11] Tong Cao, Jiangshan Yu, Jérémie Decouchant, Xiapu Luo, and Paulo Verissimo. Exploring the Monero peer-to-peer network. In *International Conference on Financial Cryptography and Data Security (FC)*, pages 578–594. Springer, 2020.

[12] Matthias Grundmann, Hedwig Amberg, Max Baumstark, and Hannes Hartenstein. Short paper: What peer announcements tell us about the size of the Bitcoin P2P network. In *International Conference on Financial Cryptography and Data Security (FC)*, pages 694–704. Springer, 2022.

[13] Kai Li, Yuzhe Tang, Jiaqi Chen, Yibo Wang, and Xianghong Liu. TopoShot: uncovering Ethereum's network topology leveraging replacement transactions. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, pages 302–319, 2021.

[14] Chonghe Zhao, Yipeng Zhou, Shengli Zhang, Taotao Wang, Quan Z. Sheng, and Song Guo. Dethna: Accurate Ethereum network topology discovery with marked transactions. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 1711–1720, 2024.

[15] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. Deanonymisation of clients in Bitcoin P2P network. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 15–29, 2014.

[16] Piyush Kumar Sharma, Devashish Gosain, and Claudia Diaz. On the anonymity of peer-to-peer network anonymity schemes used by cryptocurrencies. In *The Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2023.

[17] Giulia Fanti and Pramod Viswanath. Deanonymization in the Bitcoin P2P network. *Neural Information Processing Systems*, 30, 2017.

[18] Maria Apostolaki, Cedric Maire, and Laurent Vanbever. Perimeter: A network-layer attack on the anonymity of cryptocurrencies. In *International Conference on Financial Cryptography and Data Security (FC)*, pages 147–166. Springer, 2021.

[19] Alex Biryukov and Sergei Tikhomirov. Deanonymization and linkability of cryptocurrency transactions based on network analysis. In *IEEE European Symposium on Security and Privacy (EuroSP)*, pages 172–184. IEEE, 2019.

[20] Sebastian Henningsen, Daniel Teunis, Martin Florian, and Björn Scheuermann. Eclipsing Ethereum peers with false friends. *arXiv preprint arXiv:1908.10141*, 2019.

[21] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *IEEE European Symposium on Security and Privacy (EuroSP)*, pages 305–320. IEEE, 2016.

[22] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in Bitcoin. In *International Conference on Financial Cryptography and Data Security (FC)*, pages 515–532, 2017.

[23] Muhammad Saad, Laurent Njilla, Charles Kamhoua, and Aziz Mohaisen. Countering selfish mining in blockchains. In *International Conference on Computing, Networking and Communications (ICNC)*, pages 360–364. IEEE, 2019.

[24] Bitcoin Git repository. *https://github.com/bitcoin/bitcoin/blob/master/doc/reduce-traffic.md*, 2024.

[25] Muhammad Saad, Songqing Chen, and David Mohaisen. Syncattack: Double-spending in Bitcoin without mining power. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (AsiaCCS)*, pages 1668–1685, 2021.

[26] Jaehyun Ha, Seungjin Baek, Muoi Tran, and Min Suk Kang. On the sustainability of Bitcoin partitioning attacks. In *International Conference on Financial Cryptography and Data Security (FC)*, pages 166–181. Springer, 2023.

[27] Giulia Fanti, Shaileshh Bojja Venkatakrishnan, Surya Bakshi, Bradley Denby, Shruti Bhargava, Andrew Miller, and Pramod Viswanath. Dandelion++ lightweight cryptocurrency networking with formal anonymity guarantees. *Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS)*, 2(2):1–35, 2018.

[28] Shaileshh Bojja Venkatakrishnan, Giulia Fanti, and Pramod Viswanath. Dandelion: Redesigning the Bitcoin network for anonymity. *Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS)*, 1(1):1–34, 2017.

[29] Monero source code, Git repository. *Retrieved by July 2024. https://github.com/monero-project/monero/tree/v0.18.3.1*, 2023.

[30] Angrywasp. Levin protocol. *Retrieved by July 2024. https://github.com/nerva-project/nerva/blob/master/docs/LEVIN_PROTOCOL.md*, 2019.

[31] Monero P2P, Git repository. *Retrieved by July 2024. https://github.com/monero-project/monero/blob/v0.18.3.1/src/p2p/net_node.inl*, 2023.

[32] ErCiccione. Another privacy-enhancing technology added to Monero: Dandelion++. *Retrieved by July 2024. https://www.getmonero.org/2020/04/18/dandelion-implemented.html*, April 2020.

[33] Vtnerd. Adding Dandelion++ support to public networks. *Retrieved by July 2024. https://github.com/monero-project/monero/commit/02d887c2e58bd8e66ef8823e59669439d448bfec*, 2020.

[34] Gingeropolous. Daemon RPC documentation. *July 2024. https://github.com/monero-project/monero/wiki/Daemon-RPC-documentation*, 2019.

[35] Gingeropolous. Wallet RPC documentation. *July 2024. https://github.com/monero-project/monero/wiki/Wallet-RPC-Documentation*, 2016.

[36] py-levin, Git repository. *Retrieved by July 2024. https://github.com/sanderfoobar/py-levin/tree/master*, 2023.

[37] Maya Dotan, Yvonne-Anne Pignolet, Stefan Schmid, Saar Tochner, and Aviv Zohar. Survey on blockchain networking: Context, state-of-the-art, challenges. *ACM Computing Surveys (CSUR)*, 54(5):1–34, 2021.

[38] Muhammad Saad, Jeffrey Spaulding, Laurent Njilla, Charles Kamhoua, Sachin Shetty, DaeHun Nyang, and David Mohaisen. Exploring the attack surface of blockchain: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 22(3):1977–2008, 2020.

[39] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. Hijacking Bitcoin: Routing attacks on cryptocurrencies. In *IEEE symposium on security and privacy (SP)*, pages 375–392, 2017.

[40] Maria Apostolaki, Gian Marti, Jan Müller, and Laurent Vanbever. SABRE: Protecting Bitcoin against routing attacks. In *Proceedings of the Annual Network and Distributed System Security Symposium (NDSS*, page 02A1. Internet Society, 2019.

[41] Seoung Kyun Kim, Zane Ma, Siddharth Murali, Joshua Mason, Andrew Miller, and Michael Bailey. Measuring Ethereum network peers. In *Proceedings of the Internet Measurement Conference (IMC)*, pages 91–104, 2018.

[42] Adem Efe Gencer, Soumya Basu, Ittay Eyal, Robbert Van Renesse, and Emin Gün Sirer. Decentralization in Bitcoin and Ethereum networks. In *International Conference on Financial Cryptography and Data Security (FC)*, pages 439–457. Springer, 2018.

[43] Lucianna Kiffer, Asad Salman, Dave Levin, Alan Mislove, and Cristina Nita-Rotaru. Under the hood of the Ethereum gossip protocol. In *International Conference on Financial Cryptography and Data Security (FC)*, pages 437–456. Springer, 2021.

[44] Xi Tong Lee, Arijit Khan, Sourav Sen Gupta, Yu Hann Ong, and Xuan Liu. Measurements, analyses, and insights on the entire Ethereum blockchain network. In *Proceedings of The Web Conference (WWW)*, pages 155–166, 2020.

[45] Shi-Feng Sun, Man Ho Au, Joseph K Liu, and Tsz Hon Yuen. RingCT 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency Monero. In *European Symposium on Research in Computer Security (ESORICS)*, pages 456–474, 2017.

[46] Ghada Almashaqbeh and Ravital Solomon. SoK: Privacy-preserving computing in the blockchain era. In *IEEE European Symposium on Security and Privacy (EuroSP)*, pages 124–139. IEEE, 2022.

[47] George Kappos, Haaroon Yousaf, Mary Maller, and Sarah Meiklejohn. An empirical analysis of anonymity in Zcash. In *USENIX Security Symposium (USENIX Security)*, pages 463–477, 2018.

[48] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. Zerocoin: Anonymous distributed e-cash from Bitcoin. In *IEEE Symposium on Security and Privacy (SP)*, pages 397–411. IEEE, 2013.

[49] Rui Zhang, Rui Xue, and Ling Liu. Security and privacy on blockchain. *ACM Computing Surveys (CSUR)*, 52(3):1–34, 2019.

[50] Qin Wang, Bo Qin, Jiankun Hu, and Fu Xiao. Preserving transaction privacy in Bitcoin. *Future Generation Computer Systems (FGCS)*, 107:793–804, 2020.

[51] Rujia Li et al. SoK: TEE-assisted confidential smart contract. *Proceedings on Privacy Enhancing Technologies (PETs)*, 3:711–731, 2022.

[52] Yannan Li, Guomin Yang, Willy Susilo, Yong Yu, Man Ho Au, and Dongxi Liu. Traceable Monero: Anonymous cryptocurrency with enhanced accountability. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 18(2):679–691, 2019.

[53] Malte Möser, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, et al. An empirical analysis of traceability in the Monero blockchain. *arXiv preprint arXiv:1704.04299*, 2017.

[54] Amrit Kumar, Clément Fischer, Shruti Tople, and Prateek Saxena. A traceability analysis of Monero's blockchain. In *European Symposium on Research in Computer Security (ESORICS)*, pages 153–173, 2017.

[55] Dimaz Ankaa Wijaya, Joseph K Liu, Ron Steinfeld, Dongxi Liu, and Jiangshan Yu. On the unforkability of Monero. *ACM Asia Conference on Computer and Communications Security (AsiaCCS)*, 2019.

[56] Tsz Hon Yuen, Shi-feng Sun, Joseph K Liu, Man Ho Au, Muhammed F Esgin, Qingzhao Zhang, and Dawu Gu. RingCT 3.0 for blockchain confidential transaction: Shorter size and stronger security. In *International Conference on Financial Cryptography and Data Security (FC)*, pages 464–483. Springer, 2020.

[57] Andrew Miller, Malte Möser, Kevin Lee, and Arvind Narayanan. An empirical analysis of linkability in the Monero blockchain. *arXiv preprint arXiv:1704.04299*, 2017.

[58] Moneromooo-monero. P2P: Do not send last_seen timestamp to peers. *Retrieved by July 2024. https://github.com/monero-project/monero/commit/28a7d31565cb722e1a388743e2cbc492b6ad3bed*, 2019.

APPENDIX

A. (*Tech*) Peerlist Creation

When a Monero node sends a handshake request or timed sync request to a remote peer, the payload of remote peer's response always contains a list called *peerlist*, under which up to 250 peer records randomly selected by the remote peer from its own whitelist are stored. Generally, when the remote peer adds a selected peer record to the peerlist, the *last_seen* value of it will be set to 0[58]. This prevents mailicious attackers from inferring remote peer's outgoing connections through timestamp information in the peerlist[11].

Monero node maintains a *sent_addresses* list for each connection. When the node randomly selects 250 peer records from whitelist, it will remove the records existing in *sent_addresses*, the remaining records will be added to *peerlist*. Under this mechanism, for a benign neighbor of the target node, longer the connection lasts, fewer records the neighbor will carry in the peerlist of the timed sync response. This can reduce the impact of benign peers on graylist attack.

B. (*Tech*) Anchorlist in Monero Nodes

In addition to *(IP: Port)*, each node record in the *anchorlist* includes a crucial field named *first_seen*, which marks the timestamp of the node's initial outgoing connection with the corresponding peer. The records within the *anchorlist* are organized in ascending order based on the *first_seen* timestamp.

The *anchorlist* primarily functions during the startup phase of a node. Upon startup, the node connects to the peer in the *anchorlist* with the earliest *first_seen* timestamp that is currently available for connection. Subsequently, the *anchorlist* is cleared by the node, a process further detailed in Appendix C.

While the node operates, each time it forms an outgoing connection with a new peer, that peer's record is added to the *anchorlist*, setting the *first_seen* field to the time of connection establishment. This *first_seen* timestamp remains immutable unless the record is removed. Should the outgoing connection terminate, the corresponding peer's record is also expunged from the *anchorlist*. Consequently, the *anchorlist* consistently reflects the current state of the node's outgoing connections.

When the node stops running, what is saved in *anchorlist* are peer records corresponding to the outgoing connections before the node goes offline. The value of the *first_seen* field reflects the time span that outgoing connection to the peer was maintained before the node went offline. To a certain extent, it also reflects the connection stability of the peer and the node's trust in it. Therefore, when the node restarts to establish outgoing connections, it will give priority to the peer node with the smallest *first_seen* field in *anchorlist*.

1) The role of anchorlist in the process of connection reset attacks: Combined with the management mechanism of the *anchorlist*, it can be seen that during the connection reset attack, dropping the outgoing connection will lead to the deletion of the corresponding peer record in node's *anchorlist*. When the number of outgoing connections for a node is 0, *anchorlist* also becomes empty. *Anchorlist* does not provide the target node with any benign peer choice of outgoing connections during the connection reset attack, nor does it provide any help for the node to resist eclipse attacks.

C. (*Tech*) Selecting Outgoing Connections

Starting from seed nodes. When a node starts, it first checks the number of peer records in its whitelist: if the number of nodes in whitelist is 0, the node is a brand new node in the network and has not currently established a connection with other peers. At this time, the node will first establish a connection with the seed nodes hard-coded in Monero node program[13], and obtain the initial peers in the peerlists from these nodes. Remote peer records will be added to graylist. The new node obtains its first set of information about other peers in the network by connecting to the seed nodes.

Selecting peers from *anchorlist*. If the node's whitelist is not empty, it will first select a peer from its *anchorlist* in ascending order of records' *first_seen* to establish outgoing connection, and this process will be accompanied by the clearing of *anchorlist*. Once the outgoing connection is successfully established, subsequent peer records will be discarded. The peer that successfully establishes outgoing connection will be re-added to the *anchorlist*. Now the node has 1 outgoing connection, and the number of peers in the *anchorlist* is 1.

According to the logic of Monero source code, when the number of outgoing connections for the node is less than 2, the node will continue to try to establish outgoing connections with peers in the *anchorlist* until the number of connections reaches 2 or there is no available peer in the *anchorlist*. So after successfully establishing an outgoing connection with the first peer, the node will repeat the previous process: Get all peer records from *anchorlist* and try to select peers from the fetched records to establish a new outgoing connection. But the only peer record in *anchorlist* at this time corresponds to the outgoing connection that has been established, there is no more available peer record in *anchorlist*. Therefore, the node proceeds to the next stage of outgoing connection selection: selecting from the whitelist. At this time, the number of outgoing connections of the node is 1, and the number of peer records in *anchorlist* is 0. This is the only case where the peer records in *anchorlist* does not correspond to the node's outgoing connections. We are unclear whether it is a logic error or intentional for this part of code.

Selecting peers from whitelist. If the node fails to establish outgoing connections with peers listed in *anchorlist*, or the number of outgoing connections is greater than 2 but less than *expected_white_connections*[14], it then selects outgoing peers from the whitelist.

[13]https://github.com/monero-project/monero/blob/v0.18.3.1/src/p2p/net_node.inl, line 854.

[14]The value of *expected_white_connections* is associated with the node's pruning operation, https://github.com/monero-project/monero/blob/v0.18.3.1/src/cryptonote_protocol/cryptonote_protocol_handler.inl, line 2824.

When Monero node selects peer record from whitelist, it tends to establish connections with peers that are in different /16 IP address domains from the current peers. However, if the node has no other choice, it will consider peers that share the same /16 IP address range as the current peers. Monero node will never establish more than one outgoing connection with the same IP. Monero node prefers to choose the top 20 peers that meet the above conditions and have the largest *last_seen* value as these peers tend to have closer relationship with the node. Such a strategy increases the difficulty and required IP resources of eclipse attacks.

Selecting peers from graylist. If a node fails to establish outgoing connection with peers in the whitelist, or the number of outgoing connections of the node is less than the upper limit of outgoing connections, but not less than *expected_white_connections*, the node will select out-peers from the graylist. The strategy for selecting out-peers in graylist is almost the same as that in whitelist. The biggest difference is that peers are randomly selected from the set of peers that meet the IP requirements, instead of selecting the top peers in graylist. More details about outgoing connections establishment can be found in the *connections_maker()* function[15] of Monero source code.

*D. **Pseudo Code** for Our Eclipse Attack*

We provide implementation details of our attack (cf. Algorithm 1), covering the main logic and several key functions.

Our attack starts with two attack threads. The first attack thread starts 20 malicious nodes to perform graylist attack. The second attack thread starts 1000 malicious nodes to handshake with the target node in a fixed order to perform whitelist attack. This initial phase is crucial for preparing the target node's environment and creating conditions favorable for the subsequent steps.

Then, we enter the attack loop of connection reset attack. In this loop, the number of currently occupied outgoing connections (tracked by var. *OccupiedOutConnections*) is compared against the maximum allowable outgoing connections (var. *MaxOutConnectionsOfTheTargetNode*, which is set to 12 by default). If the number of occupied outgoing connections is less than the maximum, the attacker proceeds to execute a round of connection reset attacks.

This process is repeated until all the target node's outgoing connection slots are occupied by the attacker's nodes. Once complete control over the outgoing connections is achieved, the algorithm continuously monitors the status of the target node to detect whether incoming connections are completely evicted. When the incoming connections are completely expelled, the eclipse attack is complete.

*E. **Experiment Results** of the Eclipse Attack using connection reset attack based on Private Transaction*

The results are detailed in Table VI. Across all 10 attacks, the fastest and slowest times to occupy outgoing connections were 7.03 seconds (the 3rd attack) and 116.62 seconds (the 2nd attack), respectively. The shortest and longest times to complete an eclipse attack were 28.75 seconds (the 3rd attack) and 242.91 seconds (the 1st attack).

On average, it took 36.0 seconds to occupy outgoing connections and 134.8 seconds to complete the eclipse attack. Compared to the Dandelion++-based attack, the Private Transaction-based eclipse attack demonstrated faster performance, particularly in occupying outgoing connections. This improvement can be attributed to the stability of the connection reset attack mechanism when using Private Transactions.

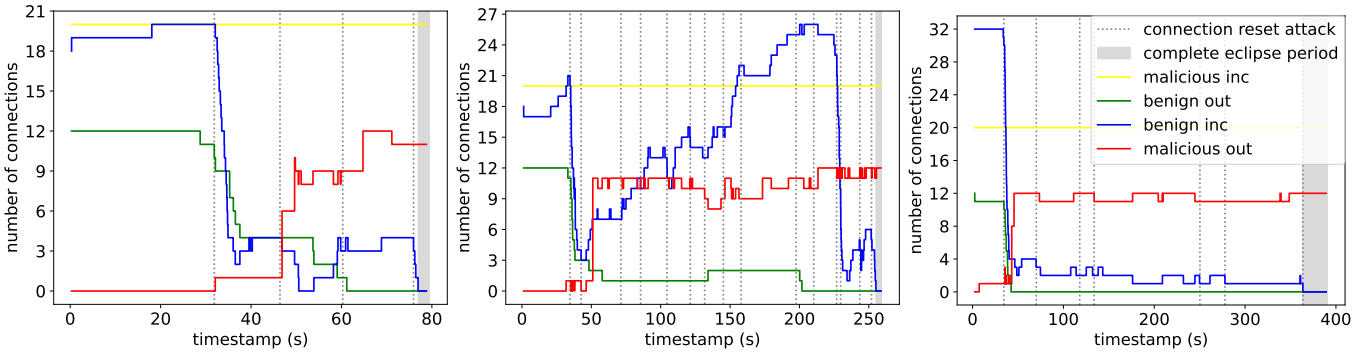
Despite these differences in performance, there is no fundamental distinction between the two types of connection reset attacks. Both effectively achieve complete occupation of all the target node's connections.

Algorithm 1: Our **Eclipse Attack** against Monero

```
Input: TargetNode = (target_ip, target_port)
Data: OccupiedOutConnections = 0,
MaxOutConnectionsOfTheTargetNode = 12,
WhitelistAttackNodeSet = 1000 nodes for whitelist attack,
GraylistAttackNodeSet = 20 nodes for graylist attack
foreach node in GraylistAttackNodeSet do
    StartThread(graylist_attack(node, target_node))
StartThread(whitelist_attack(WhitelistAttackNodeSet, target_node)
while OccupiedOutConnections <
  MaxOutConnectionsOfTheTargetNode do
    connection_reset_attack(target_node)
    OccupiedOutConnections = get_occupied_out_count()
while is_completely_occupied() == false do
    connection_reset_attack(target_node)
return ok

Subroutines
    graylist_attack(attack_node, target_node)
    Data: param1: attack_node, param2: target_node;
    attack_node.connect(target_node)
    while true do
        attack_node.wait_timedsync_request(target_node)
        attack_node.send_timedsync_response(target_node)
    whitelist_attack(attack_node_set, target_node)
    Data: param1: attack_node_set, param2: target_node;
    foreach node in attack_node_set do
        attack_node.connect(target_node)
        attack_node.send_pong_msg(target_node)
        attack_node.disconnect(target_node)
    connection_reset_attack(target_node)
    Data: param1: target_node;
    tx1 = create_tx()
    tx2 = create_tx()
    tx1 and tx2 are double-spending
    send_tx(target_node, tx1)
    broadcast_tx(tx2)
    is_completely_occupied(target_node)
    Data: param1: target_node;
    MaliciousConnections
    tx = create_tx()
    broadcast_tx(tx)
    if MaliciousConnections.recv_tx_from(target_node) then
        return false
    return true
```

[15]https://github.com/monero-project/monero/blob/v0.18.3.1/src/p2p/net_node.inl, line 1811.



(a) **fastest** eclipse attack (b) **slowest** occupation on out-connections (c) **slowest** eclipse attack&**fastest** occupation

Fig. 6: Extreme cases and attack boundaries

TABLE VI: Results for performing the eclipse attack using connection reset attack based on Private Transaction

No.	Out peers	In peers	Whitelist / Graylist occupaction rate	Whitelist / Graylist attack blocking rate	Time to occupy out conn	Time to complete eclipse attack
1st	11	25	97.44% / 83.08%	99.15% / 94.82%	77.04s	242.91s
2nd	12	37	98.06% / 60.23%	93.10% / 100%	116.62s	124.95s
3rd	11	32	96.34% / 88.80%	100% / 93.62%	7.03s	28.75s
4th	12	23	94.85% / 67.59%	100% / 94.12%	17.61s	98.07s
5th	10	26	92.33% / 82.03%	100% / 98.02%	12.82s	90.81s
6th	9	40	97.96% / 81.27%	100% / 100%	20.08s	146.39s
7th	12	34	93.04% / 91.32%	100% / 100%	9.96s	78.80s
8th	12	26	92.92% / 79.80%	100% / 87.50%	67.44s	186.73s
9th	12	20	93.02% / 77.57%	100% / 96.92%	16.69s	112.40s
10th	11	49	95.21% / 65.63%	100% / 81.48%	14.70s	238.16s

*F. **Extreme Cases** for the Complete Eclipse Attack using connection reset attack based on Dandelion++*

The fastest case to complete the eclipse attack corresponds to the 5th attack in Table IV. The time used is 45.08s. The connection change during the attack is shown in Fig.6a.

The slowest case to occupy outgoing connections corresponds to the 10th attack in Table IV. The time to complete the occupation of outgoing connections is 167.12s. The connection change during the attack is shown in Fig.6b.

The slowest case to complete the eclipse attack & the fastest case to occupy outgoing connections both occurred in the 7th attack. This attack took only 7.97s to complete the occupation of the outgoing connection, but it took 329.18s to complete the eclipse attack. Experimental result indicate that a rapid occupation of outgoing connections does not necessarily lead to a swift completion of the eclipse attack. The connection change during the attack is shown in Fig.6c.

*G. How to **Maintain** the State of Eclipse Attacks?*

As an eclipse attack seeks to control all connections to the target node, the maintenance of any benign connections compromises the attack. To detect the eclipse attack, a fluff transaction can be sent to the network. If the target node forwards the transaction to the attacker's malicious node, it indicates that the target node has established other benign connections, thereby compromising the eclipse attack state. The attacker can then use this information to initiate a new connection reset attack to reestablish the eclipse attack state.

*H. **Large-Scale** Eclipse Attack*

Experiments demonstrate the effectiveness of our eclipse attack against a single node. Beyond that, our approach is further applicable to large-scale attacks against Monero network.

An attacker, by modifying the maximum outgoing connections limit, can use a group of malicious nodes to perform graylist attacks simultaneously on multiple nodes in the network.This is feasible because these malicious nodes only need to respond to the timed sync requests of the target nodes. The capacity to handle a high volume of such requests hinges on the performance capabilities of the malicious nodes.

Similarly, the attacker can set up another group of malicious nodes to perform whitelist attacks. Each node in this group of nodes exchange handshake messages with the nodes in the target nodes set in a fixed order. In the process of attacking a single node, the node that completes the whitelist attack will wait for the node that has not yet completed the attack until the round of attack is over. In the scenario of attacking a set of nodes, this process can be organized into a pipeline to achieve simultaneous attacks on multiple targets.

Connection reset attacks can also be performed on multiple targets at the same time. For public RPC service nodes, the attacker only needs to propagate the double-spend transaction *tx1* to the transaction pool of these nodes through the RPC method and then propagate the double-spend transaction *tx2* to the network to complete the connection reset of these nodes at the same time. As for other nodes in the network, the attacker can send the stem phase transaction *tx1* to them at the same time, then propagate the fluff phase transaction *tx2* in the network, thereby using a pair of double-spending transactions to perform connection reset attack on multiple nodes at the same time. However, the more nodes that directly receive *tx1*, the faster *tx1* propagates to the entire network. This leads to a decrease in the success rate of the connection reset attack.

The detection method for the eclipse attack state is also applicable to large-scale attacks. By sending a transaction *tx* to the network, the attacker continuously detects whether the node in the target nodes set has sent the transaction *tx* to the malicious nodes controlled by the attacker within a short period of time. The node that sent *tx* is considered to have its eclipse attack state destroyed.