Yang *et al. Cybersecurity* (2023) 6:50
https://doi.org/10.1186/s42400-023-00182-9

Cybersecurity

RESEARCH

Open Access

Evicting and filling attack for linking multiple network addresses of Bitcoin nodes

Huashuang Yang[1], Jinqiao Shi[1], Yue Gao[2*], Xuebin Wang[3], Yanwei Sun[1], Ruisheng Shi[1] and Dongbin Wang[1]

Abstract

Bitcoin is a decentralized P2P cryptocurrency. It supports users to use pseudonyms instead of network addresses to send and receive transactions at the data layer, hiding users' real network identities. Traditional transaction tracing attack cuts through the network layer to directly associate each transaction with the network address that issued it, thus revealing the sender's network identity. But this attack can be mitigated by Bitcoin's network layer privacy protections. Since Bitcoin protects the unlinkability of Bitcoin addresses and there may be a many-to-one relationship between addresses and nodes, transactions sent from the same node via different addresses are seen as coming from different nodes because attackers can only use addresses as node identifiers. In this paper, we proposed the *evicting and filling attack* to expose the correlations between addresses and cluster transactions sent from different addresses of the same node. The attack exploited the unisolation of Bitcoin's incoming connection processing mechanism. In particular, an attacker can utilize the shared connection pool and deterministic connection eviction strategy to infer the correlation between incoming and evicting connections, as well as the correlation between releasing and filling connections. Based on inferred results, different addresses of the same node with these connections can be linked together, whether they are of the *same* or *different* network types. We designed a multi-step attack procedure, and set reasonable attack parameters through analyzing the factors that affect the attack efficiency and accuracy. We mounted this attack on both our self-run nodes and multi-address nodes in real Bitcoin network, achieving an average accuracy of 96.9% and 82%, respectively. Furthermore, we found that the attack is also applicable to Zcash, Litecoin, Dogecoin, Bitcoin Cash, and Dash. We analyzed the cost of network-wide attacks, the application scenario, and proposed countermeasures of this attack.

Keywords Security and privacy, Bitcoin, Address linking

Introduction

Bitcoin is a decentralized P2P cryptocurrency that has gained widespread attention over the past decade (Hou 2017; Cai et al. 2021; Nadeem et al. 2021). The most attractive highlight of Bitcoin is its protection for the anonymity and privacy of Bitcoin users, which is mainly achieved from the data and network layers (Reid and Harrigan 2013; Ober et al. 2013; Khalilov and Levi 2018). At the data layer, Bitcoin supports users to create any number of random-looking Bitcoin pseudonyms. These pseudonyms can be used instead of users' real identities to send and receive cryptocurrency transactions. Such a pseudonym mechanism prevents user transactions from being directly linked to the user's real identity, thus protecting the anonymity of users. But there have been many researches on breaking pseudo-anonymity. One type of attack they proposed is *pseudonym clustering* that links multiple Bitcoin pseudonyms belonging to the same user and analyzes the user's transaction behaviors. For

*Correspondence:
Yue Gao
gaoyue2017@126.com
[1] School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing, China
[2] Tsinghua University, Beijing 100084, China
[3] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China

example, Androulaki et al. (2013) proposed two heuristic rules, *multi-input transactions* (the multiple input pseudonyms of a single transaction belong to the same user) and *"shadow" pseudonyms* (the new pseudonym that is used to collect back the "change" is the "shadow" pseudonym, which most likely belongs to the sender), for clustering Bitcoin pseudonyms. To identify change pseudonyms more accurately, Meiklejohn et al. (2013) introduced four identification conditions: FirstAppearance, NotACoinGeneration, NoSelfchangeAddress, and UniqueNewOutputAddress. Other pseudonym clustering works include TransferAmountFeature (Wang et al. 2020), *coinbase pseudonyms* and *mining pool pseudonyms* (Zheng et al. 2020), blockchain browser WalletExplorer.com (2023), and Blockchair.com (2023). Another type of attack is *transaction tracing attack*, which can directly correlate each transaction with the network identity of its user. This attack cuts through the network layer by deploying eavesdropper nodes in the Bitcoin network and analyzing the traffic to find the originating network address[1] that sent one transaction into the network. Such an address is usually the network identity of the transaction holder (Koshy et al. 2014; Biryukov et al. 2014; Fanti and Viswanath 2017; Gao et al. 2018). This attack seems more destructive than pseudonyms clustering since it exposes the user's real network identity. To mitigate the transaction tracing attack, Bitcoin community has paid lots of attention to privacy protection in the network layer. Bitcoin community supports the configuration of multiple network addresses for a single node and has not provided a unique global node identifier to link these addresses together. In this way, users can configure multiple addresses for their nodes and send transactions through different addresses, which can be of the same network type, such as IPv4 or IPv6, or of different network types, such as IPv4 and IPv6. Thus, each transaction can only be traced to the address that generate it, while transactions from the same user are seen as coming from different users with the address as the node identifier. Meanwhile, Bitcoin community encourages users to run their node as an onion/I2P service, which can only be reachable from Tor/I2P network (Community 2023a, b). With the anonymity of these networks, tracing attacks can only trace transactions to anonymous addresses which cannot be associated with clear network addresses, i.e. the user's identity is not exposed.

Ensuring address unlinkability of the network layer is of great significance, as it mitigates transaction tracing attacks. However, achieving this goal is not easy. This is because node addresses have been used without distinction in various network mechanisms from the beginning of Bitcoin design. Researches show that attackers may exploit the common characteristics or apparent behaviors of network mechanisms to expose the correlations among addresses, thus linking different addresses of the same node (Pieter 2020; practicalswift 2020; Grundmann et al. 2022). We call this *address linking attack*. Based on this attack, attackers who have the transaction tracing ability can not only trace each transaction to the originating address but also further cluster all transactions from different addresses of the same node if these addresses are linked. For Bitcoin nodes running both clear and anonymous network addresses on a dual stack, this attack can associate their clear and anonymous addresses, which defeats the effort of the Bitcoin developing community to improve user privacy with the anonymity network.

Address linking attacks have been the focus of researchers for a long time. Biryukov et al. (2014) linked addresses of the same node by the common entry nodes set (all nodes to which the target node has established outgoing connections) across different addresses. Miller et al. (2015) followed a similar idea. Biryukov and Pustogarov (2015) tried to link different addresses of the target node by actively emitting a unique combination of possibly fake addresses (address cookie) to the address database (a database that stores all known Bitcoin addresses of non-local nodes) of the target node from one address and checking the cookie from other addresses. Mastan and Paul (2018) argued that a passive attacker who can monitor the traffic of Bitcoin nodes has the ability to link addresses of the same node by analyzing the block requests made by different addresses in a Bitcoin session graph. Pieter (2020) notes that addresses of the *same* network type of the same node share the common address cache (cached address information that is stored in the cache map of the target node and used to respond to address query requests) in the specific Bitcoin v22.0, through which node addresses of the *same* network type can be linked. Since Bitcoin developers have been concerned about address unlinkability, they fix vulnerabilities that have been disclosed each time the client updates. Therefore, against the *updated* Bitcoin version[2], all existing address linking attacks are ineffective.

While address linking attacks and related vulnerability fixes is a game of cat and mouse, Bitcoin developers currently do not conduct a systematic analysis of

[1] To avoid confusion, we use *pseudonyms* to refer to Bitcoin transaction addresses used to send and receive cryptocurrency transactions, *address* to represent the network address (IPv4/IPv6/Onion) and *node address* for all addresses that belong to the same node in this paper.

[2] In this article, we experiment with Bitcoin version 22.0 (the official C++ implementation). Despite of the short update cycle of Bitcoin, our attack still work in newer releases (see Sect. "Impact and countermeasure") while attack (Pieter 2020) fails.

the unisolated usage of local addresses in all network mechanisms. In this paper, we propose a new effective address linking attack that exploited the unisolation of the incoming connection processing mechanism. First, through source code inspection, we find that all network addresses across network types (IPv4/IPv6/Tor) of a Bitcoin node share one common connection pool (a pool that stores all incoming connections established), and the connection pool size is fixed (115 in default). Second, we find that the connection eviction strategy (a strategy for selecting an existing connection to evict in order to accept new incoming connection when the connection pool is full) is deterministic, which means that when a new incoming connection arrives at a full connection pool, the connection to be evicted in the pool is specific. For two addresses that belong to the same node, the attacker can emit elaborately designed incoming connections from one of them and achieve predictable evicting connections from the other. Also, the attacker can release connections from one of them and achieve predictable incoming connections from the other. Thus by mounting such *evicting and filling attack*, the attacker can use the two characteristics to link Bitcoin addresses of both the *same* or *different* network types, with high accuracy. Applying this attack to the result of the transaction tracing attack will further disclose users who disguise themselves with multiple addresses. Our main contributions are as follows:

1. We introduce the evicting-filling attack based on the unisolation of Bitcoin incoming connection processing mechanism, which is effective in linking node addresses of (a) both *same* and *different* network types, (b) all Bitcoin versions to date, and (c) mainstream Bitcoin forks.
2. We analyzed the factors that affect the attack efficiency and accuracy, including the number of available connection slots of the victim, the frequency of evictions caused by normal nodes during the attack duration, and the fluctuation of available slots number. We obtained empirical values of these factors through measurements in the *Mainnet* and suggested reasonable attack parameters.
3. We designed a multi-step attacking procedure and verified against our self-run nodes and real-world multi-address nodes in the *Mainnet*, achieving an average accuracy of 82% after *one* round attacking, which can be up to more than 95% after four rounds.
4. We proposed two acceleration methods for directly applying this attack on the whole network and analyzed the time and economic cost of such a network-wide attack.
5. We described the application of our evicting-filling attack, and gave countermeasures from two aspects of connection pool isolation and random disconnection time.

The remainder of this paper proceeds as follows: Sect. "Related works" summarizes the related studies on Bitcoin address linking. Section "Background" presents necessary background information. Section "Our linking attack" specifies our attack and attack parameters. The experiments are provided in Sect. "Experiments" . Section "Attack cost" analyzes the attack cost and Sect. "Application" discusses the attack application scenario. The attack impacts, and countermeasures are gaven in Sect. "Impact and countermeasure". Section "Conclusion" concludes the paper and discusses our future work.

Related works

Many Bitcoin de-anonymization works attempt to break the unlinkability of network addresses. In 2014, Biryukov et al. (2014) attempted to link addresses of the same node through the set of entry nodes. This set is cross-address and can be passively learned because when each address is connecting to the network, its entry nodes are always the first to relay its address in the network. However, due to the frequent network communications between nodes, the entry-node set of each node continues to change, making this attack inaccurate. Similarly, Miller et al. (2015) used the common set of neighboring nodes (all nodes that connects with the target node) to link addresses. They actively inferred neighboring nodes of each address by repeatedly sending GETADDRs and catching the updates of the timestamps attached to neighboring nodes in responded ADDRs. But countermeasures Community (2015b), Community (2015c) and Community (2020) have prevented this attack by removing the updates of the attached timestamps and making the neighboring nodes not inferable. In 2015, Biryukov and Pustogarov (2015) correlated different addresses of the same node by actively emitting an address cookie to the common address database from one address and checking the cookie from other addresses. But the *addr-response-caching* mechanism (Community 2020) introduced by Bitcoin developers makes the cookie easily to be overwritten or propagated out during the linking, invalidating this attack. In 2018, Mastan and Paul (2018) proposed an address linking attack for attackers who can passively monitor the Bitcoin network traffic. In this attack, different addresses of the same node can be linked by analyzing their block requests in a Bitcoin session graph. But the attack can only be launched by gateway-level attackers and the attacking scope relies on the

coverage of the monitoring traffic. In 2021, Pieter (2020) pointed out that addresses of the same Bitcoin v22.0 node with the *same* network type share the common address cache. Thus addresses can be linked based on address cache collisions (also called cache map collisions). But this attack is only applicable in *same-network* linking against Bitcoin v22.0. In 2022, Grundmann et al. (2022) noted that Bitcoin forwards addresses through different IP addresses. Attacks can send a batch of spam addresses with the same timestamp to a specific node and then node addresses relaying subsets of the spam addresses should be grouped to the same node. But this is a theoretical model. Bitcoin network will constantly forward these spam addresses, making it difficult to distinguish between source forwarding addresses and intermediate forwarding addresses, while the author does not mention attack parameters and accuracy. Besides, this attack exploits the shared relayed addresses in the address relay mechanism, which is consistent with our argument for the non-isolation in network mechanisms.

The Bitcoin developing community has taken many measures against address linking. They restricted requests to non-main-chain blocks to make the potential linking based on chain tip blocks (Community 2015a) prohibitively costly. They introduced the addr-response-caching mechanism (Community 2020) along with the cache map to prevent connection leakage and invalidate linking based on neighboring nodes. They added randomness on every cycle for transaction forwarding and cache updates to avoid potential linking based on the timing of node cyclical behavior (Community 2022b). They required all nodes to respond the same when receiving deliberately designed HEADERS from malicious nodes and prevented their local block information from being inferred through different responses, since some nodes may contain unique local blocks that others do not have and these blocks can be used as fingerprints (Community 2022g). They indexed the cache map (Community 2020) by *network type* to prevent potential linking against node addresses of *different networks* (practicalswift 2020). In versions after v22.0, they added a second index by local socket addresses to the cache map, thus preventing linking against node addresses of the *same network* based on cache map collisions (Pieter 2020).

Although the Bitcoin developing community has taken effective countermeasures against address linking, the complexity of multi-address support for IPv6, Onion, and I2P, and the complexity of network mechanisms such as the addr-response-caching mechanism, the address relay mechanism, the incoming connection processing mechanism, and etc, make it a quite difficult problem to thoroughly ensure the unlinkability among network addresses.

Background

This section introduces the necessary background of the Bitcoin network and address management.

Bitcoin network

The Bitcoin network is a fully distributed p2p network. Nodes in the network communicate with each other by directly establishing peer-to-peer connections. The connections can be divided into incoming connections, which are initiated by non-local nodes to the local node, and outgoing connections, which are initiated by the local node to non-local nodes. Each node with public network addresses (public node) can establish 10 outgoing connections and accept up to 115 incoming connections by default[3] (Community 2022d). While each node without public addresses (behind NATs and firewalls) does not accept incoming connections and relays on 10 randomly selected public nodes for outgoing connections to access the network (Biryukov et al. 2014; Wang and Pustogarov 2017; Franzoni and Daza 2020). It can be seen that public nodes are the backbone of Bitcoin network. For these nodes, more incoming connections than 115 will result in the eviction of existing connections, and the selection of evicting connections follows the *connection eviction stategy* (Community 2022c).

Bitcoin address management

Bitcoin nodes support four network types: IPv4, IPv6, Onion and I2P (though Bitcoin claimed support for I2P anonymity network from v22.0, there are no nodes of such network type currently (Foundation 2010)). Such support for multiple network types means that each node can use address combinations of *four types of network*, such as IPv4+IPv6 and IPv4+Onion, or just *one type of network* for communication. The multi-address configuration can be achieved by passing in Bitcoin startup parameters, which is shown in Fig. 1.

Each Bitcoin node customizes a key-value pair container *mapLocalHost* (Community 2022e) to store all its network address information (local addresses), which takes each address as a key and stores the corresponding running port for that address as the key value. Users can obtain multiple network addresses for their nodes at least in these ways: (a) run a dual stack and add more addresses of *different* network types by accessing the IPv6 network and/or creating local Bitcoin Onion services, (b) map more addresses of the *same* network type through host proxies, port forwarding, and multiple NICs. In fact, Bitcoin connections are established between two

[3] Bitcoin Core is configured default with a maximum number of incoming connections *DEFAULT_MAX_PEER_CONNECTIONS*(125) − *MAX_OUTBOUND_FULL_RELAY_CONNECTIONS*(8) − *MAX_BLOCK_RELAY_ONLY_CONNECTIONS*(2) = 115.

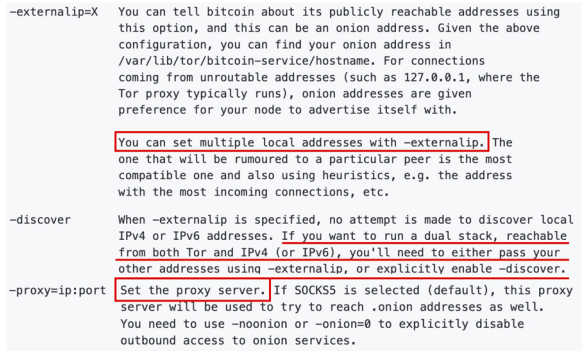


Fig. 1 Examples of command-line arguments in Bitcoin Core that support multi-address configuration (Community 2023b)

"addresses", as two neighbors do not know each other's more addresses.

Each Bitcoin node stores all known address information of other nodes (non-local addresses) in its address database *CAddrMan* (Community 2022i). Due to no node identifier, the address database is managed using address as the identifier and it may contain multiple addresses of the same node. Each time a node has established an outgoing connection with one other node, it will send a GETADDR message to this new neighbor to query more addresses of others. In versions older than v22.0, the neighbor will respond with no more than 2,500 randomly selected addresses from its *CAddrMan*, and every 1000 addresses are packaged into one ADDR. In v22.0, to avoid the neighbor's *CAddrMan* potentially being scraped quickly by responding to many maliciously repeated GETADDRs from attackers, the number of addresses that respond is reduced to 1000. Meanwhile, the chosen addresses being responded are cached into the cache map and returned to any GETADDR requests within a period of 21–27 h. This is the *addr-response-caching* mechanism (Community 2020). To prevent address linking *across networks*, the cache map is indexed by the network types to which local addresses belong. The second index by local socket addresses to prevent address linking *in the same network* is added in versions after v22.0.

Our linking attack

In this section, we will introduce the basic idea, attacking procedure, and attack parameters of our linking attack.

Basic idea

The Bitcoin incoming connection processing is witnessed in the code (Community 2022a, also simplified in Algorithm 1). By binding to the local running port, each Bitcoin node listens for incoming connections from others. Once receiving an incoming connection, the node first checks whether the remote address that initiates the connection is malicious, i.e. if it has ever delivered invalid or erroneous blocks in the network. If it hasn't, the node then counts the number of incoming connections itself has held. If there are no more than 115 connections established, the node will directly accept the new incoming connection and store it in a built-in array *vNodes*[4] (Community 2022j). Otherwise, the node will execute the *connection eviction strategy* to try to select one existing connection to disconnect and then accept the new one. If no existing connection meets the eviction criteria, the new connection will be rejected (Community 2022a).

Algorithm 1 Bitcoin incoming connection processing

Input: The bind socket for an incoming connection $hListenSocket$
Output: Accept or reject connection $hListenSocket$.
function AcceptConnection($hListenSocket$)
  $\cdots$;
  $hSocket = accept(hListenSocket, \&addr)$;
  $CreateNodeFromAcceptedSocket(hSocket, addr)$;
end function

function CreateNodeFromAcceptedSocket($hSocket, addr$)
  $nInbound \leftarrow$ *calculate number of incoming connections*;
  $nMaxInbound = 115$;
  if $IsBanned(addr)$ *or* $IsDiscouraged(addr)$ **then**
    $CloseSocket(hSocket)$;
    return
  end if
  $\cdots$;
  if $nInbound >= nMaxInbound$ **then**
    if $!AttemptToEvictConnection()$ **then**
      $CloseSocket(hSocket)$;
      return
    end if
  end if
  $pnode \leftarrow$ *create a new node to accept this connection*;
  $vNodes.push(pnode)$;
end function

From this processing, we can see that Bitcoin does not check the local address it uses to receive a new incoming connection and directly stores all accepted connections that may be associated with different local addresses into the common array *vNodes*. We refer to this array as Bitcoin's *connection pool* with default size (115 slots, the maximum number of incoming connections). And it can be concluded that all local addresses of both *same* and *different* network types share the common connection pool.

We now drive into the *connection eviction strategy* to see what kind of connections are preferred to be evicted. As shown in the code (Community 2022c, simplified in Algorithm 2), Bitcoin first preserves connections established with specific remote addresses to which the user has granted special privileges (NoBan privilege,

[4] In most recent v24.0, this array is renamed *m_nodes* and is also shared.

Community 2022f), as well as connections that are about to be disconnected. Then among all the remaining connections, Bitcoin follows the steps below to select a specific one for eviction:

(1) Select 4 peers to protect by netgroup (the network group is determined by the prefix of each connection's remote address[5]).
(2) Protect 8 connections with the lowest minimum ping time.
(3) Protect 4 connections that most recently sent novel transactions accepted into mempool.
(4) Protect up to 8 non-transaction-relay connections that have sent novel blocks.
(5) Protect 4 connections that most recently sent novel blocks.
(6) Protect half of the remaining eviction candidates according to their network types and connection duration.
(7) Identify the network group with the most connections and youngest member and evict a most recently established connection from it.

We refer to the first six steps simply as the *special connection protection* policy. This policy is used to protect some potentially secure connections with *certain characteristics* (Community 2022h), such as belonging to one of the network groups randomly selected, maintaining a minimum ping time with the node, having relayed the latest block or transaction to the node, or being initiated from an Onion address or an I2P address. Assuming that an attacker can establish many connections to the node, then even if his 4 connections are protected due to their network group being selected, there are still plenty of attacking connections left. If the attacker can only establish a few connections with the node, it means that the node's connection pool is nearly full and there are no groups that contain more connections to evict. In that case, the number of distinct groups in the pool is large, and the probability of the attacker's group being selected is extremely low. Besides, having a minimum ping time with the node can be avoided easily by adding a little response delay. The attacker can also bypass the rest conditions by initiating connections from a standard IPv4 address and not relaying recent blocks or transactions to the node. Thus, the attacker can circumvent the special connection protection policy by constructing connections *with no certain characteristics*. And these attacking connections can reach the last step of the eviction strategy, becoming the connections that are preferred to be evicted. In order to avoid other non-attacking connections remaining in the pool for preferred eviction, these attacking connections can reach the maximum size of the fixed connection pool (115) in number and are all from the same network group. Then these connections will continue to evict existing connections that can be evicted until the attacking group becomes the group with the highest priority for eviction. We can conclude that evicting connections are predictable and controllable.

Based on the above two findings, an attacker can first emit elaborately designed incoming connections to one address of the target node, in order to a) fill up its connection pool and push the node into the connection eviction phase, b) make these attacking connections become connections with the highest priority to be evicted. Then, the attacker can initiate more incoming connections to another address of the same node and observe whether his connections with the former address are being evicted. Note that the number of successfully established connections with the latter address should be equal to the number of evicting connections with the former address. We call this an *incoming-evicting* test.

Since the incoming-evicting change, i.e. the number of accepted or evicted connections, may be too small due to the high network delay in the real world and the victim continuously evicting the last few emitted connections. We proposed another *releasing-filling* test, in which once the attacker releases some connections with one address of the target node, he can fill up these released connection slots through another address of the same node.

The two tests can be used to link Bitcoin network addresses. We designed a precise multi-step attacking procedure, combining these two tests in parallel to ensure attack efficiency and accuracy.

Algorithm 2 Bitcoin connection eviction strategy

```
function AttemptToEvictConnection
    vEvictionCandidates ← all incoming connections without NoBan privilege and are not to be disconnected;
    SelectNodeToEvict(vEvictionCandidates)
end function

function SelectNodeToEvict(()vEvictionCandidates)
    // Protect connections with certain characteristics.
    vEvictionCandidates ← remove 4 peers to protect by netgroup;
    vEvictionCandidates ← remove 8 nodes with the lowest minimum ping time;
    vEvictionCandidates ← remove 4 nodes that most recently sent novel transactions accepted into local mempool;
    vEvictionCandidates ← remove 8 non−tx−relay peers that have sent novel blocks;
    vEvictionCandidates ← remove 4 nodes that most recently sent novel blocks;
    vEvictionCandidates ← remove half of the remaining candidates by ratios of desirable;
    targetNetGroupNodes ← identify the network group with the most connections and youngest member;
    targetNode ← the youngest connection from the group;
    Disconnect(targetNode);
end function
```

[5] The selected network groups are unpredictable for attackers.

Attacking procedure for one pair addresses

We now present the attack model, attacking procedure for one pair addresses and attacking procedure for multiple addresses.

Algorithm 3 linking for single address pair

Input: $A \leftarrow Target\ address\ A$; $B \leftarrow Target\ address\ B$
Output: $result$
function LinkingForSingleAddressPair(A, B)
  $connections\_with\_A = []$;
  $connections\_with\_B = []$;
  $MaxConnectionPoolSize = 115$;
  // This is the first phase.
  // AF_1: Number of connections available to A
  // BF: Number of accepted incoming connections with B
  // AF_2: Number of remaining connections with A after the eviction
  for $i \leftarrow 1$ to $MaxConnectionPoolSize$ **do**
    $connections\_with\_A+ = initOneConnection(A)$
  end for
  $AF_1 \leftarrow count\ stable\ connections\ in\ connections\_with\_A$;
  for $i \leftarrow 1$ to $MaxConnectionPoolSize$ **do**
    $connections\_with\_B+ = initOneConnection(B)$
  end for
  $BF \leftarrow count\ stable\ connections\ in\ connections\_with\_B$;
  $AF_2 \leftarrow count\ remaining\ active\ connections\ in\ connections\_with\_A$;
  $characteristic_1 \leftarrow check\ if |AF_1 - (AF_2 + BF)| < TH_1$;
  // This is the second phase.
  // BS: Number of connections available to address B
  $disconnectConnections(connections\_with\_A)$;
  $disconnectConnections(connections\_with\_B)$;
  for $i \leftarrow 1$ to $MaxConnectionPoolSize$ **do**
    $connections\_with\_B+ = initOneConnection(B)$;
  end for
  $BS \leftarrow count\ stable\ connections\ in\ connections\_with\_B$;
  $characteristic_2 \leftarrow check\ if |AF_1 - BS| < TH_2$;
  if $characteristic_1$ & $characteristic_2$ **then**
    $result = True$;
  else
    $result = False$;
  end if
  return $result$;
end function

Attack model

Our attack model assumes that the victim node V is a Bitcoin node accepting incoming connections, with multiple local addresses that may belong to the *same* network type or *different* network types (see Fig. 2). For any two addresses A and B of node V, our goal is to verify if they belong to the same node. As for the attacker, we assume that he controls one or more attacking nodes. Each attacking node owns a public network address for establishing connections with the victim and the address is of the same network group as other attacking nodes. No attacking node needs to maintain a blockchain, but instead executes a lightweight script with the following functions: a) supporting up to 230[6] parallel outgoing connections, b) not relaying new transactions and blocks to the victim, c) adding a little response delay (0.2s) each time responding to PING from the victim, d) for each connection successfully established, initiating a heartbeat test once every two minutes to keep alive. For simplicity, we suppose that the attacker controls one attacking node S whose public address is P_S here.

Attacking procedure

Our evicting-filling attack consists of two phases.

First phase - step a As shown in Fig. 3(1-a) (also simplified in Algorithm 3), the attacker fills up the connection pool of victim V through address A by initiating 115 Bitcoin connections without characteristics from the same IPv4 address P_S.[7] Notice that Bitcoin allows multiple connections from one single address (Saad et al. 2021). This property significantly reduces the attack cost. If the connection pool is not full, node V will accept the incoming connections in turn until its pool becomes full. Then for the remaining pending connections received, node V will continuously evict as many connections as possible from all existing connections and try to accept them. If the connection pool is full, node V will directly enter the connection eviction phase. From the attacker's view, he will eventually establish a certain number of connections with address A after all his connections are responded to or timeout disconnected. This is the number of connections available to address A and we assume it to be AF_1. At this moment, the AF_1 connections will have the highest priority for eviction.

First phase—step b As shown in Fig. 3(1-b), the attacker emits more incoming connections through address B by initiating 115 connections from address P_S.[8] Meanwhile, the attacker monitors evicting connections with address A. Since the above AF_1 connections associated with A have the highest eviction priority, V will evict the most recently established connection or connections from the AF_1 connections to accept incoming connections. From the attacker's perspective, he will observe that the number of connections established with address B gradually stabilizes (we assume this number to be BF), and the original AF_1 connections with address A are decreased to AF_2. The equivalence between the evicting connections count and accepted incoming connections count is our *first expected behavioral characteristic*, i.e. $AF_1 = BF + AF_2$.

Second phase As shown in Fig. 3(2), the attacker disconnects actively all connections established with address A and address B in the first phase. And then he

[6] If the two victim addresses do not belong to the same node, filling up their connection pools needs at most 230 connections.

[7] In fact, it is not mandatory to be the same address, the addresses belonging to the same network group are sufficient. But using a single address reduces the cost of the attack.

[8] Using addresses of the same network group with P_S is actually sufficient, but not the cheapest.

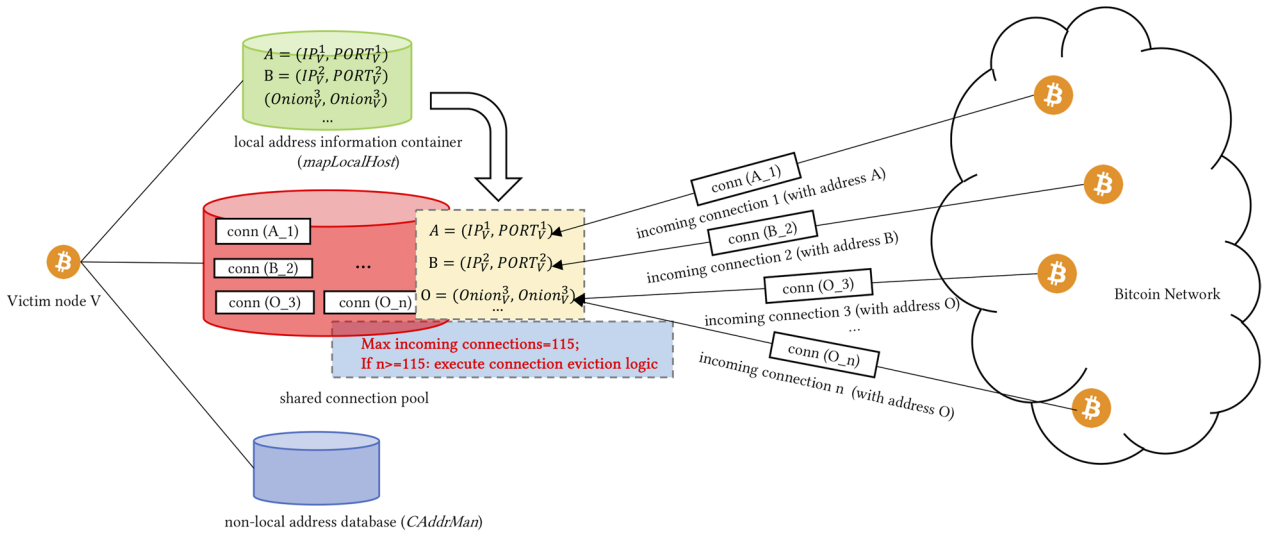


Fig. 2 Victim model for the linking attack

fills up the connection pool through address B by initiating 115 connections. After these connections are all responded to or timeout disconnected, he will record the number of successfully established connections, BS,

which is the number of connections available to B. The equivalence between the released connection slots count and the number of connections successfully established with address B is our *second expected behavioral characteristic*, i.e. $AF_1 = BS$.

To figuratively show the two behavioral characteristics, we mounted the entire attack against a Bitcoin multi-address node built on our server and plotted Fig. 4. In the first phase (from moment a to b), the number of connections available to address A, AF_1 (106), was first measured. Then 115 connections were initiated to address B and BF (46) connections were eventually established, while connections with address A dropped to AF_2 (59). We can see that AF_1 is highly close to $BF + AF_2$. In the second phase (from moment b to c), we disconnected all connections with addresses A and B. Then measured the number of connections available to address B, BS (106). We can see that AF_1 is highly close to BS. Thus, we can successfully conclude that addresses A and B belong to the same node.

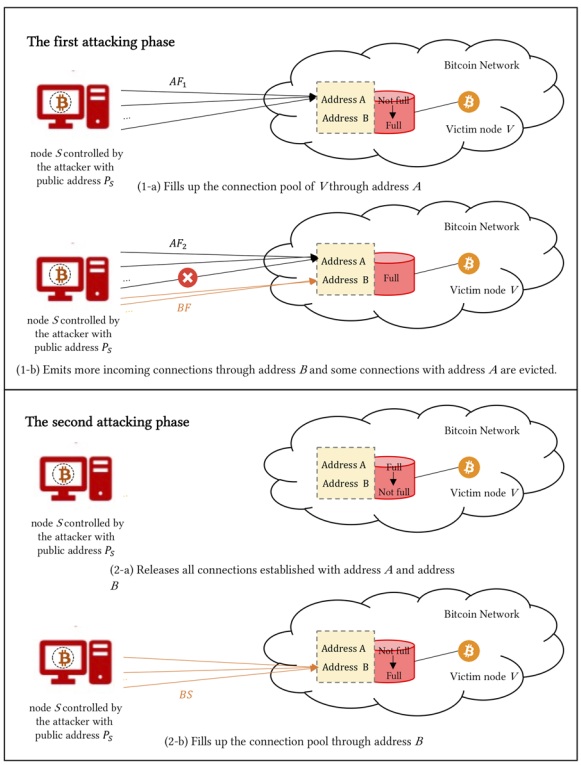


Fig. 3 Attacking procedure of the linking attack

Attacking process for multiple addresses

An attacker can in-depth use the above linking attack for two addresses to link all node addresses within a network, thus achieving a certain scale of privacy leakage. The whole process is as follows (also simplified in Algorithm 4):

- Obtaining the set T of all Bitcoin addresses within the network.
- Enumerating all possible combinations of addresses in T.

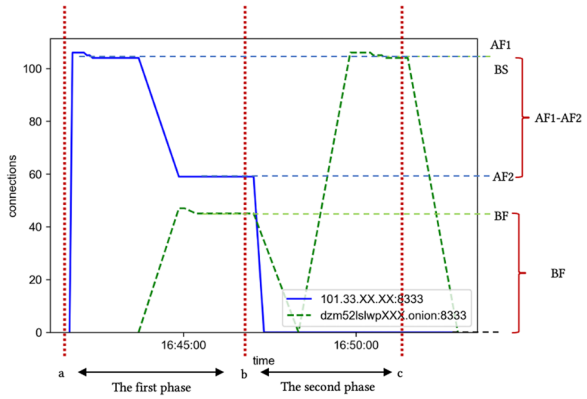


Fig. 4 Illustrative diagram of two behavioral characteristics. The two behavioral characteristics: $AF_1 = AF_2 + BF$; $AF_1 = BS$

- Mining the correlation between each pair of addresses via the evicting-filling attack for two addresses.
- Clustering associated addresses into nodes.

Finally, the attacker will get a list $I = \{(IP_1, IP_2, O - nion \cdots), \cdots\}$, where IP_1, IP_2, and *Onion* are all addresses of the same node. Based on the results, attackers can link transactions from different addresses of the same node together to analyze the user's transaction behavior.

Algorithm 4 linking for multiple addresses

Input: $T \leftarrow$ *All target addresses*; $filtered\_addr\_pairs \leftarrow$ *filtered address pairs through acceleration methods*
Output: $result\_final$
1: **function** LinkForMultiAddrs(T, $filtered\_addr\_pairs$)
2: **for** each $addr_a$ in T **do**
3: **for** each $addr_b$ in T **do**
4: **if** $(addr_a, addr_b)$ not in $filtered\_addr\_pairs$ **then**
5: $result = LinkingForSingleAddressPair(addr\_a, addr\_b)$;
6: **else**
7: $continue$;
8: **end if**
9: $result\_final.push(result)$;
10: **end for**
11: **end for**
12: **return** $result\_final$;
13: **end function**

Attack parameters

We note that three factors will affect the attack efficiency and accuracy, including the number of available connection slots of the victim, the frequency of evicting connections caused by normal nodes during the attack duration, and the fluctuation of available connection slots number. These factors have a high probability of causing AF_1 and $BF + AF_2$ to be unequal, as well as AF_1 and BS. Thus, we analyze these factors and obtain empirical values for them through measurements.

Dataset

To facilitate experiments, we construct a dataset consisting of self-run and real-world nodes.

We deployed five self-run nodes on the Bitcoin Mainnet. Each node is a v22.0 Bitcoin Core running with default parameters and configured with multiple addresses of *various* combinations of three network types, IPv4, IPv6, and Onion (as shown in Table 1).

As mentioned earlier, the linking attack based on cache map collisions (Pieter 2020) can be launched in v22.0, through which we also captured some real-world nodes with multiple addresses of the same network in the *Mainnet*. Note that there is currently no quantitative analysis of cache map collision linking, so we supplement this content in Appendix to better explain this attack. Below, we only present the collection process and results of *Mainnet* nodes. From February 20 to February 26, 2022, we obtained all reachable addresses running v22.0 clients in the Mainnet each day by using an open-source crawler (Foundation 2010) and captured their address cache by sending address query requests. During the daily address cache collecting, we found that some addresses accept connections but do not respond ADDR to our GETADDR. So we could not collect all caches for these addresses and the final number we collected is shown in Table 3, averaging 3,943 per day. After applying SimHash (Wikipedia 2022b) and cosine similarity algorithm (Wikipedia 2022a), we considered address caches with the same SimHash signatures and cosine similarity higher than 90% to be identical.[9] Addresses with the same address cache are clustered on the same node. The number of collided caches and corresponding clustered nodes we collected is also shown in Table 3, with a total of 404 caches and 179 nodes.

The number of available connection slots of the victim

To estimate this number, we crawled 8,601 reachable addresses in the *Mainnet* on March 2, 2022. We initiated 115 parallel connections to each address and recorded the number of eventually established connections. Figure 5 shows the result. It can be seen that 95% of these addresses accept incoming connections. 52% accept 5 or more connections, and 20% accept up to 30 connections. Only a few nodes do not accept connections, and even if we can not establish a sufficient number of connections with them immediately, we can wait a long time to establish enough connections since the number of their

[9] Although the cached addresses are fixed, we found that caches in each response of the same node varied slightly, mainly due to the IPv6 address zero compression. So we do not request them to be entirely identical.

available connection slots is continuously and dynamically changing (according to Fig. 9). In addition, although 43% addresses can only accept less than 5 connections, the multi-address nodes within them can still satisfy two behavioral characteristics if the 5 connections are not affected by other factors and only change with the attack behaviors.

The frequency of evictions caused by normal nodes during the attack duration

We launched multiple rounds of our evicting-filling attacks on both self-run and real-world nodes and calculated the attack duration. Our results are shown in Figs. 6 and 7. Taking the median as a reference value, the attack time required for the first attacking phase is distributed between 56 and 157 s, and the attack time required for whole attack is distributed between 65 and 290 s.

We also measured the frequency of evictions caused by normal nodes by establishing lots of connections with our node set and monitoring the change in the number of connections over time. Assume that the attack time required for the first attacking phase is Δt. Figure 8 shows that the number of evicted connections is no more than 8 in 95% of the experiments during the Δt from 60 to 180 s which covers the time required for the first phase.

The fluctuation of available connection slots number

Assume that the attack time required for the whole attack is Δt. Since the number of all connection slots is fixed, the number of available connection slots mainly depends on the number of existing connections. Thus, we monitored our self-run nodes in March 2022 and recorded the number of their existing connections every minute. Figure 9 shows that in the interval Δt between 60 s and 360 s that covers the time required for the whole attacking procedure, the number does not exceed 7 in 95% of the experiments.

Selection of attack parameters

Considering the impact of the above factors on the attack, we set two thresholds, TH_1 and TH_2, to balance attack accuracy and efficiency. If the difference between AF_1 and $BF + AF_2$ is less than TH_1, and the difference between AF_1 and BS is less than TH_2, we take the address pair as satisfying two behavioral characteristics.

Since the number of evictions caused by normal nodes has a 95% probability of not exceeding 8 within the interval Δt from 60 to 180 s, we set the attack parameter $TH_1 = 8$. Since there is a 95% probability that the fluctuation of available connection slots number does not exceed 7 within the interval Δt between 60 and 360 s, we set the attack parameter $TH_2 = 7$.

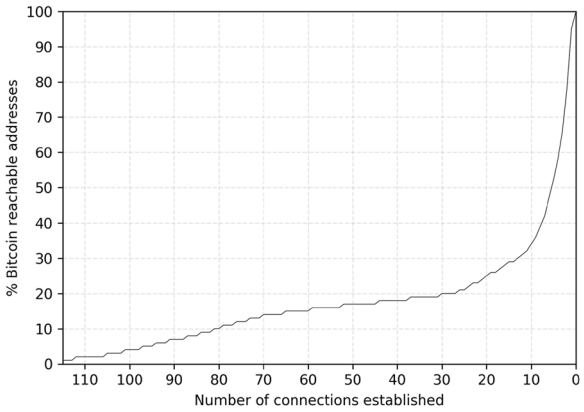


Fig. 5 Distribution of available connection slots number of Mainnet addresses

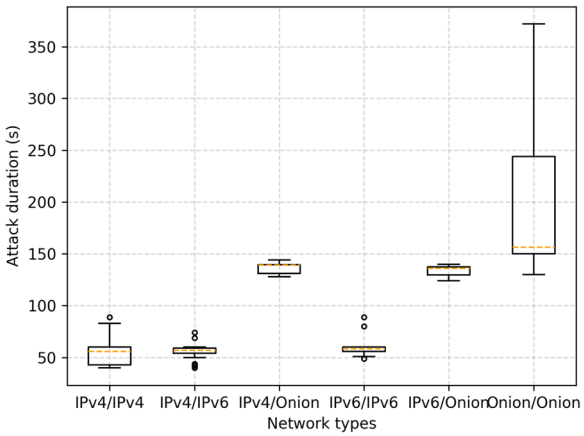


Fig. 6 Distribution of attack time required for the first attacking phase (The dashed line is the median line)

We suppose that there is an interfering address X unrelated to address A, with x number of available slots. In the worst situation, $AF_1 - 8 + x = AF_1 + 8$ ($AF_2 = AF_1 - 8$) and $x = AF_1 \pm 7$, thus X will be wrongly linked with address A. We can solve for x at such situation to be 16 and $AF_1 = 23$. In fact, the smaller the number of available connection slots of the victim address, the greater the interference of the normally evicting frequency. Therefore, we set a smaller threshold value $TH_1 = AF_1 \times \alpha (\alpha < 1)$ for $AF_1 \leq 23$ to ensure the attack accuracy (Based on practical experience, we set $\alpha = 0.2$ in our experiments). Similarly, we set a smaller threshold value $TH_2 = AF_1 \times \beta (\beta < 1)$ for $AF_2 \leq 7$ (Based on practical experience, we set $\beta = 0.2$ in our experiments). In general, the thresholds are set as follows:

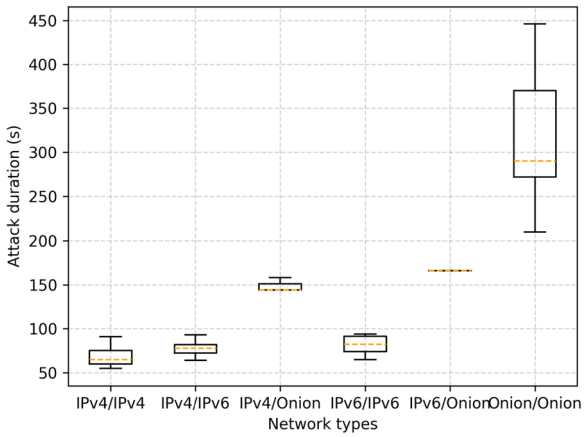


Fig. 7 Distribution of attack time required for the whole attack (The dashed line is the median line)

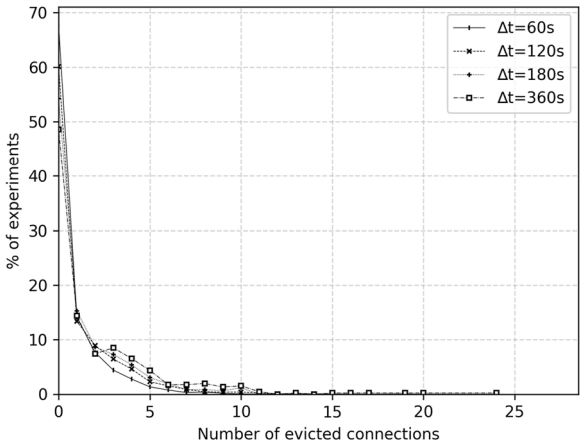


Fig. 8 Probability density of normally evicted connections number for multi-address nodes

$$TH_1 = \begin{cases} 8 & AF_1 > 23 \\ AF_1 \times 0.2 & AF_1 \leq 23 \end{cases}$$

$$TH_2 = \begin{cases} 7 & AF_1 > 7 \\ AF_1 \times 0.2 & AF_1 \leq 7 \end{cases}$$

Experiments

With the above two attack parameters, we mounted the self-run nodes verification experiment and the Mainnet nodes verification experiment from February 20 to February 26, 2022, to verify the feasibility of our attack.

Self-run nodes verification

We conducted a week-long experiment on the eleven addresses of five self-run nodes. Our goal was to link all associated addresses and identify the corresponding multi-address nodes on a daily basis without knowing the correlations between these addresses at all. As a small-scale validation experiment, we directly used these addresses as the set T for linking and then verified the fifty-five possible combinations of these addresses sequentially by evicting-filling attacks. our results are shown in Table 2.

The true positive rate and true negative rate are extremely high, showing that our method performs well and one run of attacks can accurately cluster the addresses of all self-run nodes on most days. Especially, the false positive rates are 0%, which means that there were no unrelated address pairs being clustered to the same node and reflects the strong identity of the two behavioral characteristics we designed. The false negative rates showed there were some misjudgments in the attacks of February 24 and February 26, which means address pairs belonging to the same nodes were judged as unassociated. The possible reason is that the connection pool fluctuations of misjudging nodes during that attack duration exceed the limit of our attack parameters, including the temporary full-state of the victim connection pool, the frequency of evicting connections caused by normal nodes, and the fluctuating number of existing connections. To validate this, we conducted a second round of attacks, and these misjudgments were resolved successfully. In short, this experiment verifies that our evicting-filling attack is feasible for both *same-network* and *cross-network* address linking, with an average accuracy of 96.9% for one round of attacks.

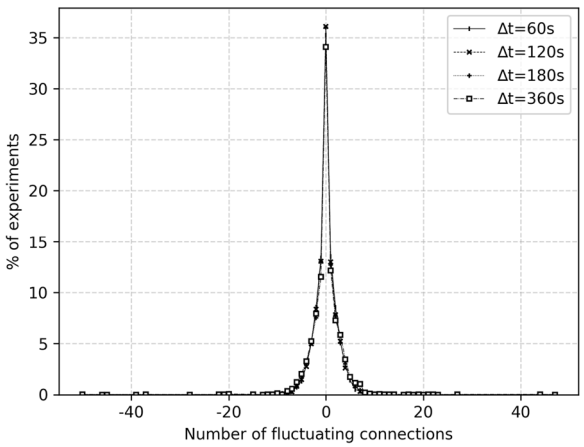


Fig. 9 Probability density of fluctuating number of existing connections

Mainnet nodes verification

In this experiment, our goal was to verify the correlations between real-world addresses. During the verification, we found that there were some dynamic addresses

among the dataset. The caches of such addresses collided, but they were active in the Bitcoin network successively, with no overlap in time. The possible reason could be that these nodes switched their proxies or their hosts used DHCP. Our attack cannot link these addresses, as they do not share the connection pool simultaneously. Meanwhile, we found that there were also some addresses of supernodes. The clients of such nodes are often specially modified by their users, making their connection pool very large while the exact size is unknown to us. It's hard to fill their pools up, so our attack is also not applicable to them. For the remaining node addresses, we verified them by our evicting-filling attacks, and the results are shown in Table 3.

After one round of attacks, we get an average true positive rate of 82% and an average false negative rate of 18% after one round of attacks. This true positive rate is lower than that of self-run nodes and the false negative rate is higher than that of self-run nodes, which may be because the standard connection evictions and connection pools of real-world nodes fluctuate more volatile. To validate this, we conducted more consecutive rounds of attacks. As shown in Fig. 10, the false negative rate significantly decreases as the number of attacking rounds increases. This experiment was conducted against real-world nodes. Although the cache map was used to collect experimental addresses, we did not use this property throughout our verification. The high true positive rate and low false negative rate show that our evicting-filling attack still has strong feasibility and high accuracy in the real world.

In addition, we classified the local network types of the 105 multi-address nodes collected and showed results in Table 4.[10] This classification result confirms the diversity and complexity of the multi-address nodes in the real world. Since the *same-network* linking attack based on cache map collisions can only be applied in v22.0, we believe that our attack is better in *cross-network* linking and *same-network* linking against all versions as there has no attention been given to the unisolation in incoming connection processing mechanism yet.

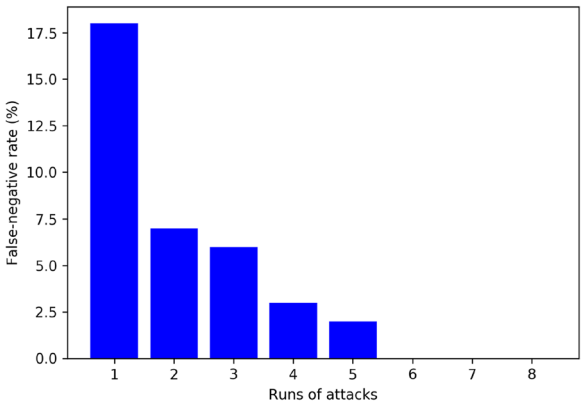


Fig. 10 False negative rate decreases with increasing number of attack rounds

Table 1 Address configuration for self-run nodes

Node	Addresses count	Network types of addresses
Node 1	2	IPv4/IPv6
Node 2	2	IPv4/IPv4
Node 3	2	IPv4/Onion
Node 4	3	IPv4/IPv6/Onion
Node 5	2	Onion/Onion

Table 2 Self-run nodes verification results

Date	Attacks count (%)	True positive (%)	True negative (%)	False positive (%)	False negative (%)
2022-02-20	55	100	100	0	0
2022-02-21	55	100	100	0	0
2022-02-22	55	100	100	0	0
2022-02-23	55	100	100	0	0
2022-02-24	55	85.7	100	0	14.3
2022-02-25	55	100	100	0	0
2022-02-26	55	71.4	100	0	28.6

More details

Conducting multiple rounds of attacks is a way to improve the accuracy rate by avoiding misjudgment caused by accidental connection pool fluctuations, which include the temporary full-state of the victim connection pool, the frequency of evicting connections caused by normal nodes, and the fluctuation number of existing connections. In our experiments, *self-run nodes verification* and *Mainnet nodes verification*, we did not modify our attack parameters since the probability of each fluctuation occurring is small (only about 5%) and our parameters (obtained from long-term measurements) cover 95% of our measurement experiments. Instead, we took advantage of the time interval among multiple rounds of attacks since one round of our attacks lasted about two hours. Such time interval plays a role in avoiding accidental connection pool fluctuations, as the fluctuations depend on how busy the Bitcoin network is and the timing of multiple attacks may cover the network state from busy to non-busy.

[10] We unexpectedly found that eleven nodes are across network types, which may be because their users did not use the addr-response-caching mechanism properly.

Table 3 Mainnet nodes verification results

Date	Address caches	Collisions count	CN	DN	SN	AN	Attack counts	True-positive (%)	False-negative (%)
2022-02-20	3910	69	31	11	4	16	28	87.5	12.5
2022-02-21	3894	50	23	5	2	16	23	75	25
2022-02-22	3889	54	24	8	2	14	21	78.6	21.4
2022-02-23	3937	67	32	13	2	17	21	82.4	17.6
2022-02-24	4008	61	23	8	2	13	71	92.3	7.7
2022-02-25	3983	52	24	7	2	15	21	73.3	26.7
2022-02-26	3980	51	22	6	1	15	23	86.7	13.3

CN means clustered nodes, DN means dynamic nodes, SN means supernodes, AN means remaining nodes to be attacked

Table 4 Network type combinations of multi-address nodes

Network type	Number of nodes
IPv4/IPv4	59
IPv6/Ipv6	10
Onion/Onion	25
IPv4/IPv6	2
IPv4/Onion	1
IPv4/IPv6/Onion	8

There is another way to mitigate the impact of accidental connection pool fluctuations. The attack parameters we suggested in this paper are empirical values obtained from long-term measurements of three factors that correspond to these fluctuations, but these fluctuations are in real-time. Thus, it is recommended to deploy some sampling nodes in the Bitcoin network and measure these three factors in real time. Based on the collected real-time data, we can statistics the distribution of available connection slots number of Mainnet addresses, and calculate the probability density of normally evicted connections number for multi-address nodes and fluctuating number of existing connections. According to the analysis, we can choose reasonable values of attack parameters that can cover most measurements. Such instant values reflect how busy the network is, thus mitigating the impact of fluctuations on attacks.

Attack cost

For ethical reasons, we do not conduct a network-wide attack and only analyze the cost of it here. Suppose the number of all public Bitcoin addresses in the network is N and the time required for one attack is t. Verifying whether any address A^* in the network is associated with a given address, A, requires $N-1$ attacks and lasts $T_0 = t(N-1)$. And verifying any addresses A and A^* requires $C_N^2 = \frac{(N(N-1))}{2}$ attacks and lasts $T = t \times \frac{(N(N-1))}{2}$. It can be seen that the time cost of a network-wide attack is high. In order to solve this problem, we propose two acceleration methods to filter out definitely unassociated address pairs before evicting-filling attacks as follows:

Unassociated address pair filtering based on basic node information After a TCP connection is established between Bitcoin addresses, VERSION messages are first sent to exchange their basic node information, which includes *version*, *services*, *user_agent*, *start_height*, *relay* fields (Wiki 2021). Among them, *version* identifies the protocol version used by the corresponding node, *services* identifies the functions it supported, *user_agent* identifies its user agent information, *start_height* identifies its synchronization height, and *relay* identifies whether the node is involved in transaction forwarding. Since the basic information of the same node is identical, an attacker can determine that addresses A and A^* with different basic node information ([*version*, *services*, *user_agent*, *start_height*, *relay*]) are unassociated.

Unassociated address pair filtering based on synchronized blocks Block synchronization of Bitcoin nodes is realized through three messages INV, GETDATA, and BLOCK (Developer 2022). After a block is received or created by address A, the transaction hash is first sent to address B via an INV message. If address B has not received the block before, it will send back a GETDATA message, and address A will return the complete block information via a BLOCK message. Since the blocks synchronized by the same node are identical, when an attacker receives an INV from address A, he can immediately send a GETDATA to address A^*. If address A^* returns a BLOCK, it may be associated with A. Otherwise, they must not belong to the same node.

To simply verify the two methods, we conducted the following experiment. Through network snapshots crawled from Bitnodes between July 12 and July 16, 2022, we calculated 4593 unique Bitcoin Onion addresses

remaining persistently online and targeted them for linking. Before acceleration, the original number of address pairs that need to be attacked is $C_{4593}^{2} = 10,545,528$. According to the 1093 network snapshots during these four days, we screened out address pairs with different basic node information or different synchronization heights at the same time. To accommodate the not entirely real-time snapshots,[11] we require that the block heights of candidate address pairs differ by two or less in the same snapshot, not exactly equal. The result is shown in Fig. 11, it can be seen that 10,464,488 (99%) address pairs are filtered out after applying all snapshots. Through analyzing the remaining pairs, we get 4,072 addresses and each of them has an average of ≈ 40 potentially associated addresses, with a maximum of 2,482 and a minimum of 1. Thus in the worst case, 2,482 attacks are required for a given address, which are lasting only about $2,482 \times 290s \approx 8.3$ days for one attacking node. Moreover, the total time for all 4593 Onion addresses can be reduced to $81040 \times 290s \div 10 \approx 27$ days for ten attacking nodes. In fact, this attack time will be much shorter if these addresses are a mixture of IPv4, IPv6 and Onion.

In our attack, the attacking nodes only need to be configured with a public network address and capable of running lightweight scripts. Thus, an attacker can simply rent basic cloud virtual machines ($4 per month for one VM (Ocean 2022)) and acquire static IP addresses ($\approx$ $39 for one IPv4 address (Group 2022)). Since attacking one network consisting of 4593 nodes lasts at most 27 days for ten attacking nodes, the cost is $\approx$ $430. If an attacker wants to increase the attack accuracy to 95%, four rounds attacking costs $\approx$ $550.

Application

In this section, we discuss our application scenarios in detail.

As shown in Fig. 12, Bitcoin communicates at the network layer using network addresses as identifiers and trades at the data layer using pseudonyms. We see correlating all transactions of a user as a breach of the data layer pseudonym mechanism, which exposes the user's transaction behavior. And associating all network addresses of a user is a violation of the network layer address unlinkability, which completely discloses the user's network identities. Traditional transaction tracing techniques can only correlate each transaction with the source address that issued it, but cannot infer the association between each address and the user. Thus, while it undermines Bitcoin's anonymity to some extent, it does not fully break through the anonymity protection

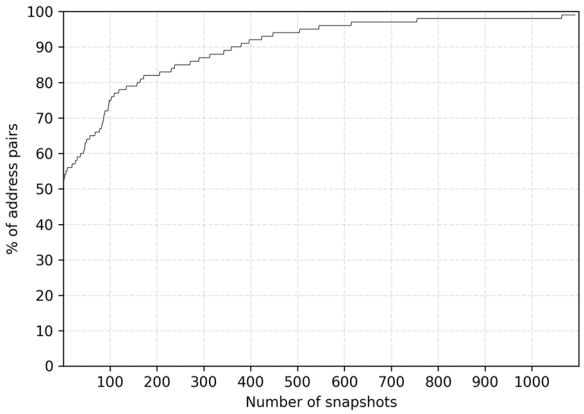


Fig. 11 The number of filtered address pairs increases as the number of overlay snapshots increases

mechanisms at the data and network layers. The traditional pseudonym clustering technology can associate all transactions of a user to break the anonymity protection of the data layer, but it does not break through the network layer.

To fill the gap where the association between addresses and users cannot be inferred, we propose two solutions. The first solution is to apply the pseudonym clustering results to the transaction tracing results, associating different network addresses at the bottom layer through the correlation of upper-layer transactions. There has been no research work in this direction so far. We believe that pseudonyms clustering is essentially achieved with the help of heuristic rules, which have inherent limitations in terms of both comprehensiveness and accuracy. The second solution is the address linking attack. This attack exploits flaws in the design and implementation of Bitcoin network mechanisms, which can provide an intuitive solution to the problem of unlinkability between network addresses. In addition, the combination of address linking attack and traditional transaction tracing technology can cluster the upper-layer transactions based on the correlation of the underlying addresses. In this way, the double anonymity of the Bitcoin data layer and network layer can also be destructed.

The complete deanonymization process of our evicting-filling attack combined with transaction tracing technology is shown in Fig. 13. First, the attacker deploys eavesdropper nodes in the network and establish connections with all online addresses. Once the eavesdropper node receives transactions, such as tx_A and tx_B, it traces them back to the earliest forwarded addresses, such as A and B, according to the received time series. Next, the node can check whether A and B belong to the same node through the evicting-filling attack. If A and B are linked, the transactions tx_A and tx_B can be clustered to a

[11] According to our observations, Bitnodes saves one network snapshot every 5 min and the node block heights it provides are not completely real-time.

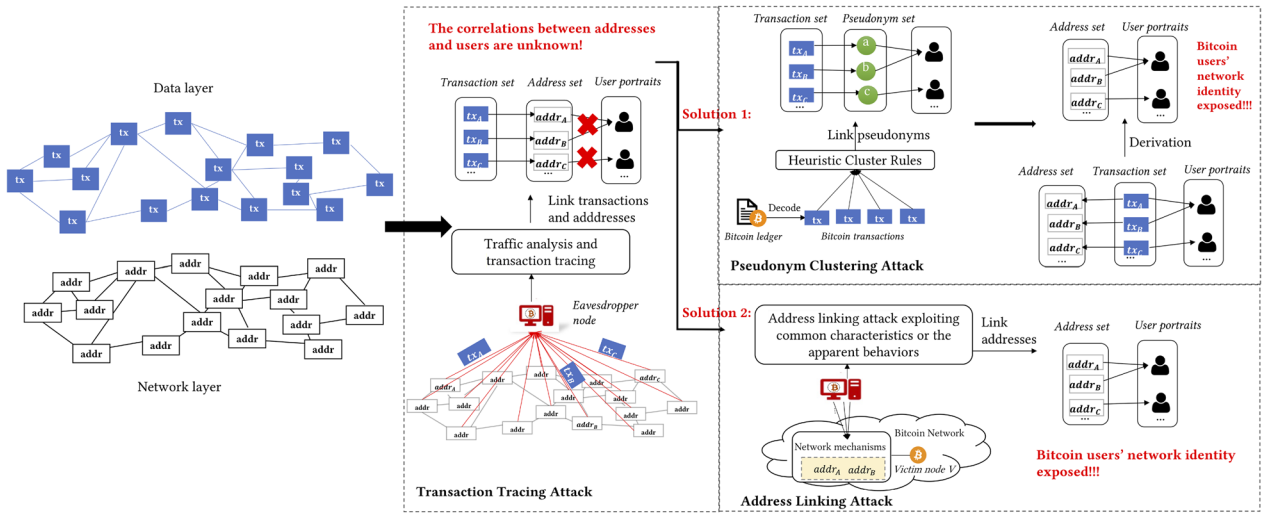


Fig. 12 Multi-dimensional linking view for Bitcoin

user whose network identity is (A, B). If A and B are not linked, then transaction tx_A belongs to one user whose network identity is (A) and transaction tx_B belongs to another user whose network identity is (B).

Impact and countermeasure

Impacts

Through analysis of Bitcoin source code from v0.10.0 to v24.0, we find that Bitcoin shares the connection pool in all versions. More incoming connections are directly dropped in versions v0.10.0-v0.13.0, which makes the interference between address connectivity more obvious. The connection eviction strategy is introduced in versions from v0.13.0, along with the idea of evicting connections from the network group having the most connections and youngest member. Thus, all versions of Bitcoin are affected by the attack described in this paper, even the latest released official version 24.0. Figure 14 shows source code comparison of the incoming connection processing mechanisms for Bitcoin v22.0 and v24.0.

Besides, we have manually investigated mainstream Bitcoin variants, Zcash, Litecoin, Dogecoin, Bitcoin Cash, and Dash, from Github repositories. These cryptocurrencies follow very similar network designs to Bitcoin. We take Bitcoin Cash as an example and show the source code comparison of the incoming connection processing mechanisms for Bitcoin v22.0 and Bitcoin Cash v26.0 in Fig. 15. For simplicity, we just position associated locations of the shared connection pool and deterministic eviction strategy in Table 5 for the rest cryptocurrencies.

Countermeasures

Here we suggest two countermeasures for our linking.

Isolate the connection pool by different local addresses

Since our evicting-filling attack exploits Bitcoin shared connection pool, thus the first measure is to check the local address used to accept one new connection and assigns a separate connection pool for each local address when processing incoming connections. As for the size of each connection pool, it can be set by either Bitcoin developers or users. By using isolated pools, connections associated with different local addresses lose the ability to affect each other.

Reduce the predictability of evicting connections count and the releasing empty slots count

Our linking attack needs to be completed in a short time and depends on real-time changes in the number of connections. If Bitcoin adds random time each time it disconnects instead of disconnecting in real-time, the attacker will have to wait a longer time to observe the change in the number of connections, which makes the attack more susceptible to three affecting factors. In this way, the evicting connections count and releasing slots count are difficult to be predicted.

From the point of our view, the previous works (Pieter 2020; practicalswift 2020) actually exploited the flaw of shared address cache in the addr-response-caching mechanism and Grundmann et al. (2022) exploited the flaw of shared relay addresses in the address relay mechanism. These works explored the unisolation in different network mechanisms but did not awaken

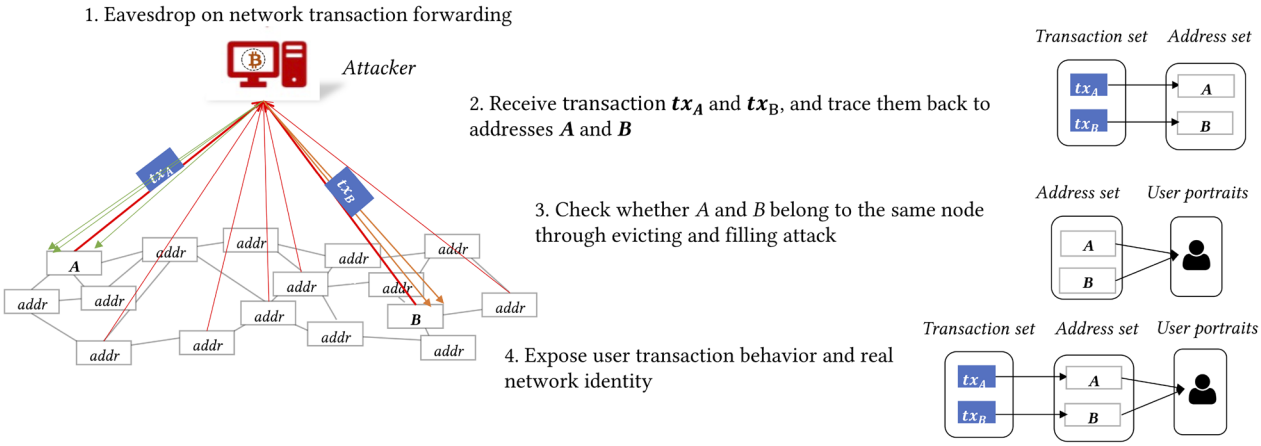


Fig. 13 Complete deanonymization process for evicting-filling attack combined with transaction traceability technology

developers' awareness of a comprehensive analysis of unisolation. In addition to the problem of shared connection pool in the incoming connection processing mechanism, We also find that in the banning and discouragement mechanism, all local addresses share the banned and discouraged lists. For each address in the banned list, incoming connections from it will be rejected by all local addresses. For addresses in the discouraged list, connections with them are preferred for eviction no matter from which local address the new incoming connection is received. These are all potential pitfalls that could be utilized to undermine address unlinkability, so the Bitcoin developing community may need to seriously analyze the isolation in all network mechanisms in the next upgrade.

Conclusion

In this paper, we present the evicting-filling attack that can link multiple addresses belonging to the same Bitcoin node regardless of network type. The attack is a new side channel attack, which is the first work to focus on the shared connection pool and deterministic connection eviction strategy of Bitcoin's incoming connection processing mechanism. We design a multi-step attacking procedure and mount this attack in the *Mainnet*, achieving high accuracy. To be noticed, this attack can be combined with traditional transaction tracing techniques for further de-anonymization against both the data and network layers. In such an application scenario, the attack can link transactions from different addresses and associate clear and anonymous addresses of a dual-stack system, exposing the transaction behavior and real network identities of users. By demonstrating the great harm that can be caused by unisolation, We take this work as a stepping stone and aim at awakening Bitcoin developers' awareness of comprehensive analysis for unisolation in all network mechanisms.

In the future, we are planning to further utilize other unisolated natures of existing network mechanisms, such as shared banned and discouraged lists. And then do a comparative analysis of the efficiency and accuracy of different address-linking attacks. In addition, we mention combining transaction tracking and transaction clustering as another solution for de-anonymization in this paper. Our next step is to validate the feasibility of this solution.

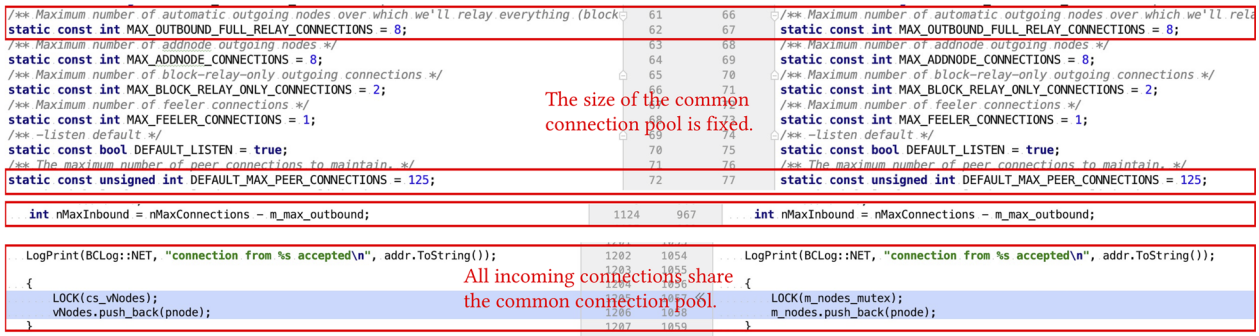
User safety and ethics

We disclosed the attack to Bitcoin Core developers before the publishing of this article. To protect user privacy, we restricted from linking in the whole Bitcoin Mainnet. Although analyzing affecting factors requires us to conduct measurements on the Mainnet, we do not cause any network anomalies. Moreover, we do not use our linking results for further de-anonymization attacks or privacy acquisition.

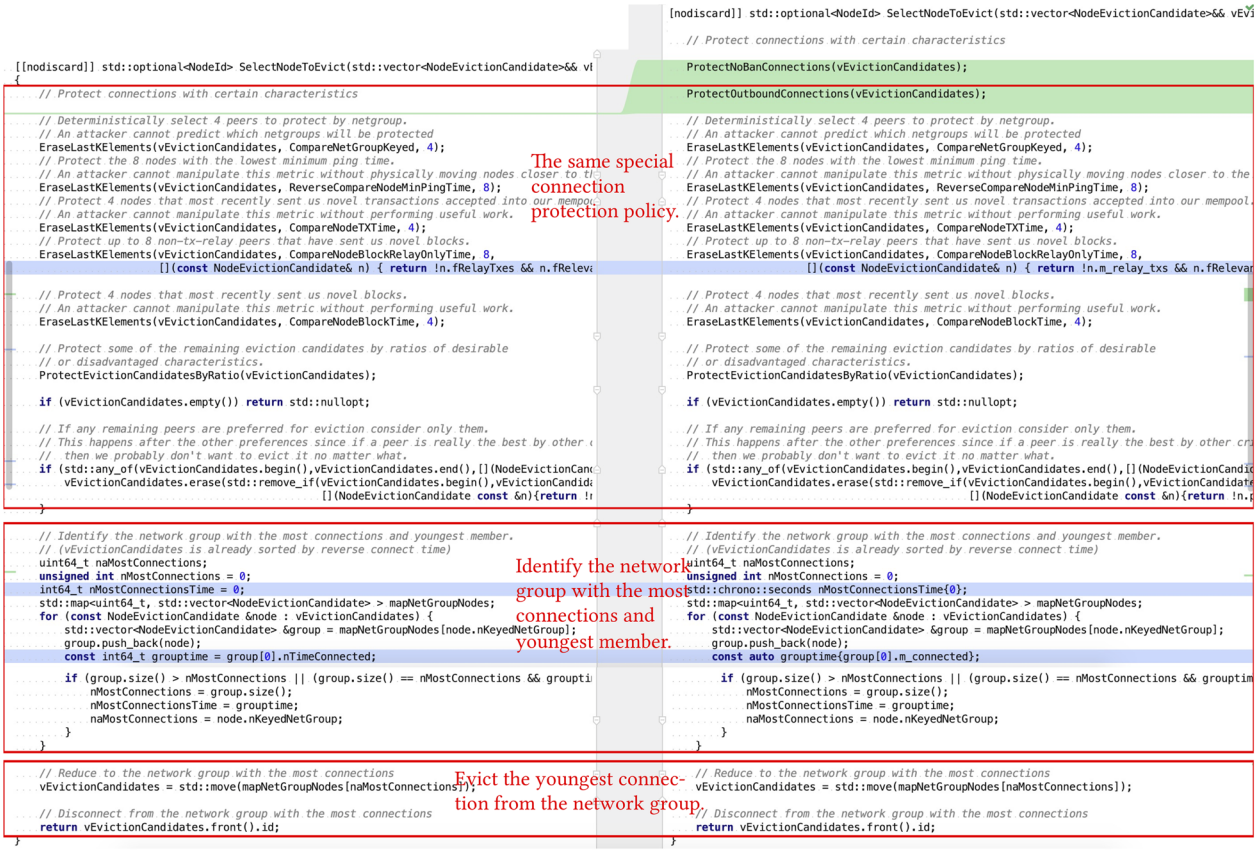
Appendix

Quantitative analysis for cache map collision linking

Bitcoin developers implemented addr-response-caching mechanism through the cache map to prevent neighboring nodes leakage. The cache map stores cached addresses that responded to address query requests. Cache map collision refers to the phenomenon of two different Bitcoin addresses with the same address cache. In v22.0 of Bitcoin,



(a) Characteristic I: Shared connection pool

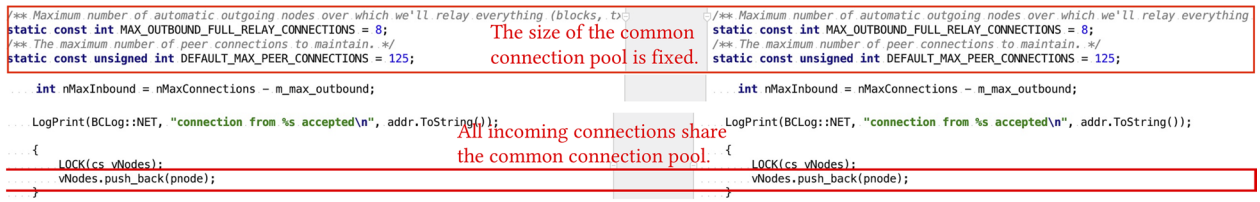


(b) Characteristic II: Deterministic eviction strategy

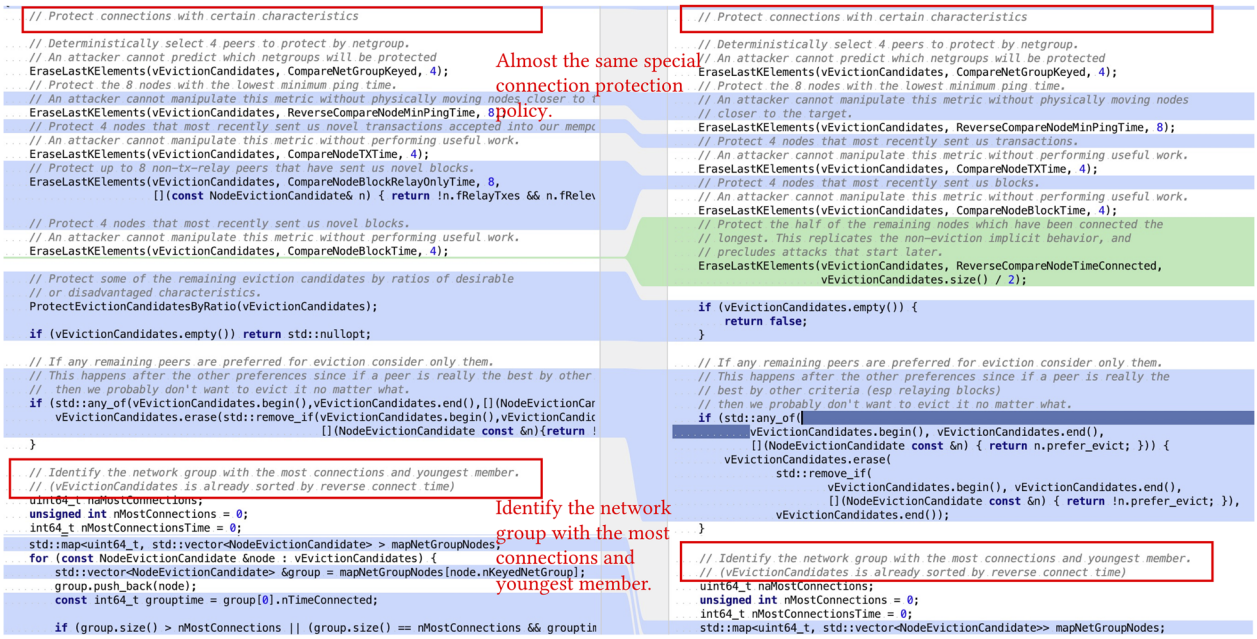
Fig. 14 Example: Source code comparison of incoming connection processing mechanism for Bitcoin v22.0 and v24.0 (the left side is the code of v22.0 and the right is of v24.0)

the cache map is indexed only by *network types* of local addresses. Thus, addresses of the same node with the same network type must collide on their cache maps. We now demonstrate why addresses with conflicting cache maps must belong to the same node.

Each Bitcoin address database contains a maximum of 81,920 addresses, and the actual size is typically smaller. Thus, we counted the address database sizes of our five self-run nodes that have been running on the Mainnet for two weeks. As shown in Fig. 16, we can see that their address database size is relatively stable, with an average of 65,731



(a) Characteristic I: Shared connection pool



(b) Characteristic II: Deterministic eviction strategy

Fig. 15 Example: Source code comparison of incoming connection processing mechanism for Bitcoin v22.0 and Bitcoin Cash v26.0 (the left side is the code of Bitcoin and the right is of Bitcoin Cash)

Table 5 Source code locations of two connection pool processing characteristics of mainstream Bitcoin variants

Cryptocurrency	Version	Shared connection pool	Deterministic eviction strategy
Zcash	5.4.2	At line No.1091 of Zcash (2023)	At line No.983 of Zcash (2023)
Litecoin	0.21	At line No.1157 of Litecoin (2022)	At line No.1015 of Litecoin (2022)
Dogecoin	1.14.7	At line No.1107 of Dogecoin (2023)	At line No.995 of Dogecoin (2023)
Bitcoin Cash	26.0	At line No.1099 of Cash (2023)	At line No.947 of Cash (2023)
Dash	19.x	At line No.1218 of Dash (2022)	At line No.1147 of Dash (2022)

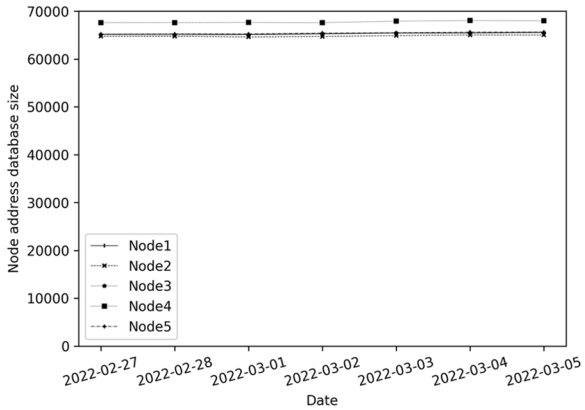


Fig. 16 Node address database size change

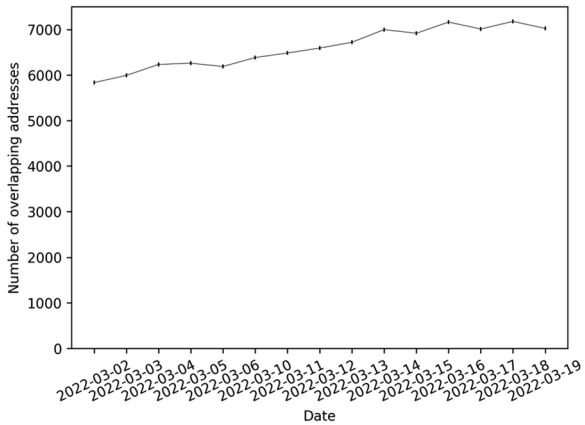


Fig. 17 Daily average number overlapping addresses in the address database of self-run nodes

addresses. Meanwhile, we counted the number of overlapping addresses with identical address information (network address, running port, timestamp, service list, network type) in the address database of any two nodes from March 2 to March 18. As shown in Fig. 17, the number of overlapping nodes stabilizes after a period of growth. The highest average number of overlapping addresses is on March 18, which is 7,317. According to these measurements, the probability of generating the same address cache containing 1000 identical addresses from two separate Bitcoin nodes is less than:

$$\left(\frac{C_{7317}^{1000}}{C_{65731}^{1000}}\right)^2 = \left(\frac{7317!(65731-1000)!}{(7317-1000)!65731!}\right)^2 < 10^{-981}$$

Thus, it is unlikely for the cache maps of different nodes to collide and we believe that addresses with the same address cache belong to the same node.

Acknowledgements
This work was supported by the Key Research and Development Program for Guangdong Province under Grant 2019B010137003 and the Beijing Natural Science Foundation under Grant M21037. Besides, we thank our anonymous reviewers for their helpful feedback and guidance.

Author contributions
HY: investigation, methodology, materials, writing, editing, experiment, validation, review, resources. JS: discussion, review, supervision. YG, XW, RS, DW: discussion, review. All authors read and approved the final manuscript.

Funding
This work was supported by the Key Research and Development Program for Guangdong Province under Grant 2019B010137003 and the Beijing Natural Science Foundation under Grant M21037.

Availability of data and materials
Our data and codes are provided at https://github.com/twinkleluna/Evicting-Filling/. For the multiple addresses belonging to the same node, we desensitized them.

Declarations

Competing interests
The authors declare that they have no competing interests.

Received: 4 May 2023 Accepted: 26 July 2023
Published online: 07 October 2023

References

Androulaki E, Karame GO, Roeschlin M, Scherer T, Capkun S (2013) Evaluating user privacy in bitcoin. In: Financial cryptography and data security: 17th international conference, FC 2013, Okinawa, Revised Selected Papers 17, Springer, pp 34–51
Biryukov A, Pustogarov I (2015) Bitcoin over tor isn't a good idea. In: 2015 IEEE symposium on security and privacy, IEEE, pp 122–134
Biryukov A, Khovratovich D, Pustogarov I (2014) Deanonymisation of clients in bitcoin p2p network. In: Proceedings of the 2014 ACM SIGSAC conference on computer and communications security, pp 15–29
Blockchair.com: Blockchair (2023). https://blockchair.com/. Accessed 27 Mar 2023
Cai L, Sun Y, Zheng Z, Xiao J, Qiu W (2021) Blockchain in China. Commun ACM 64(11):88–93
Cash B (2023) Shared connection pool and deterministic eviction logic in Bitcoin Cash. https://github.com/bitcoin-cash-node/bitcoin-cash-node/blob/v26.0.0/src/net.cpp. Accessed 7 Mar 2023
Community BD (2015a) Better fingerprinting protection for non-main-chain getdatas. https://github.com/bitcoin/bitcoin/commit/85da-07a5a001a563488382435202b74a3e3e964a. Accessed 5 Mar 2023
Community BD (2015b) Ignore getaddr messages on Outbound connections. https://github.com/bitcoin/bitcoin/commit/dca799e1db6e-319fdd47e0bfdb038eab0efabb85. Accessed 5 Mar 2023
Community BD (2015c) Reduce fingerprinting through timestamps in 'addr' messages. https://github.com/bitcoin/bitcoin/commit/9c-2737901b5203f267d21d728019d64b46f1d9f3. Accessed 5 Mar 2023
Community BD (2020) Cache responses to GETADDR to prevent topology leaks. https://github.com/bitcoin/bitcoin/pull/18991. Accessed 5 Mar 2023
Community BD (2022a) AcceptConnection. https://github.com/bitcoin/bitcoin/blob/v22.0/src/net.cpp. Accessed 2022
Community BD (2022b) Add randomness on every cycle to avoid the possibility of P2P fingerprinting. https://github.com/bitcoin/bitcoin/-blob/master/src/net_processing.cpp. Accessed 5 Mar 2023
Community BD (2022c) AttemptToEvictConnection. https://github.com/bitcoin/bitcoin/blob/v22.0/src/net.cpp. Accessed 2022

Community BD (2022d) Bitcoin v22.0. MAX_PEER_CONNECTIONS/MAX_-OUTBOUND_FULL_RELAY_CONNECTIONS/MAX_BLOCK_RELAY-_ONLY-_CONNECTIONS. https://github.com/bitcoin/bitcoin/blob/v22.0/src/net.h. Accessed 2022
Community BD (2022e) mapLocalHost. https://github.com/bitcoin/bitcoin/blob/v22.0/src/net.cpp Accessed 19 Mar 2023
Community BD (2022f) Noban NetPermissionFlag. https://github.com/-bitcoin/bitcoin/blob/v22.0/src/net_permissions.h#L31. Accessed 27 Aug 2022
Community BD (2022g) Prevent block index fingerprinting by sending additional getheaders messages. https://github.com/bitcoin/-bitcoin/pull/24571. Accessed 27 Aug 2022
Community BD (2022h) Protect connections with certain characteristics. https://github.com/bitcoin/bitcoin/blob/d571cf2d2421c6f8efb2b61ca-844034eaf230945/src/node/eviction.cpp#L180. Accessed 27 Aug 2022
Community BD (2022i) Stochastical (IP) address manager. https://git-hub.com/bitcoin/bitcoin/blob/v22.0/src/addrman.h. Accessed 19 Mar 2023
Community BD (2022j) vNodes. https://github.com/bitcoin/bitcoin/blob/v22.0/src/net.h. Accessed 19 Mar 2023
Community BD (2023a) I2P support in Bitcoin Core. https://github.com/bitcoin/bitcoin/commits/master/doc/i2p.md. Accessed 5 Mar 2023
Community BD (2023b) Tor support in Bitcoin Core. https://github.com/bitcoin/bitcoin/blob/master/doc/tor.md. Accessed 5 Mar 2023
Dash: Shared connection pool and deterministic eviction logic in Dash (2022). https://github.com/dashpay/dash/blob/v19.x/src/net.cpp. Accessed 7 Mar 2023
Developer B (2022) Bitcoin Block Synchronization. https://developer.bitcoin.org/devguide/p2p_network.html Accessed 5 Mar 2023
Dogecoin: shared connection pool and deterministic eviction logic in Dogecoin (2023). https://github.com/dogecoin/dogecoin/blob/1.14.7-dev/src/net.cpp. Accessed 7 Mar 2023
Fanti G, Viswanath, P (2017) Anonymity properties of the bitcoin p2p network. Preprint arXiv:1703.08761
Foundation B (2010) Bitnodes: reachable Bitcoin nodes. https://bitnodes.io/. Accessed 25 Mar 2023
Franzoni F, Daza V (2020) Improving bitcoin transaction propagation by leveraging unreachable nodes. In: 2020 IEEE international conference on blockchain (Blockchain), IEEE, pp 196–203
Gao F, Mao H, Wu Z, Shen M, Zhu L, Li Y (2018) Lightweight transaction tracing technology for bitcoin. Chin J Comput 41:989–1004
Group IM (2022) IPv4 Transfer Pricing. https://ipv4marketgroup.com/ipv4-pricing/ Accessed 27 Aug 2022
Grundmann M, Baumstark M, Hartenstein H (2022) On the peer degree distribution of the bitcoin p2p network. In: 2022 IEEE international conference on blockchain and cryptocurrency (ICBC), IEEE, pp 1–5
Hou H (2017) The application of blockchain technology in e-government in china. In: 2017 26th international conference on computer communication and networks (ICCCN), IEEE, pp 1–4
Khalilov MCK, Levi A (2018) A survey on anonymity and privacy in bitcoin-like digital cash systems. IEEE Commun Surv Tutor 20(3):2543–2585
Koshy P, Koshy D, McDaniel P (2014) An analysis of anonymity in bitcoin using p2p network traffic. In: Financial cryptography and data security: 18th international conference, FC 2014, Christ Church, Revised Selected Papers 18, Springer, pp 469–485
Litecoin: Shared connection pool and deterministic eviction logic in Litecoin (2022). https://github.com/litecoin-project/litecoin/blob/0.21/src/net.cpp. Accessed 7 Mar 2023
Mastan ID, Paul S (2018) A new approach to deanonymization of unreachable bitcoin nodes. In: Cryptology and network security: 16th international conference, CANS 2017, Hong Kong, Revised Selected Papers 16, Springer, pp 277–298
Meiklejohn S, Pomarole M, Jordan G, Levchenko K, McCoy D, Voelker GM, Savage S (2013) A fistful of bitcoins: characterizing payments among men with no names. In: Proceedings of the 2013 conference on internet measurement conference, pp 127–140
Miller A, Litton J, Pachulski A, Gupta N, Levin D, Spring N, Bhattacharjee B (2015) Discovering bitcoin's public topology and influential nodes
Nadeem MA, Liu Z, Pitafi AH, Younis A, Xu Y (2021) Investigating the adoption factors of cryptocurrencies-a case of bitcoin: empirical evidence from china. SAGE Open 11(1):2158244021998704
Ober M, Katzenbeisser S, Hamacher K (2013) Structure and anonymity of the bitcoin transaction graph. Future Internet 5(2):237–250
Ocean D (2022) Simple, predictable pricing for Basic Droplets. https://www.digitalocean.com/pricing/droplets. Accessed 27 Aug 2022
Pieter W (2020) Address linking in the same network based on cache map collisions. https://github.com/bitcoin/bitcoin/pull/18991-#issue-comment-668219345. Accessed 5 Mar 2023
practicalswift: address linking cross networks based on cache map collisions (2020). https://github.com/bitcoin/bitcoin/pull/18991-#issuecomment-629745947. Accessed 5 Mar 2023
Reid F, Harrigan M (2013) An analysis of anonymity in the Bitcoin system. Springer
Saad M, Chen S, Mohaisen D (2021) Syncattack: double-spending in bitcoin without mining power. In: Proceedings of the 2021 ACM SIGSAC conference on computer and communications security, pp 1668–1685
WalletExplorer.com: WalletExplorer (2023). https://www.walletexplorer.com/. Accessed 27 Mar 2023
Wang L, Pustogarov I (2017) Towards better understanding of bitcoin unreachable peers. Preprint arXiv:1709.06837
Wang M, Ichijo H, Xiao B (2020) Cryptocurrency address clustering and labeling. Preprint arXiv:2003.13399
Wiki B (2021) Bitcoin VERSION Message. https://en.bitcoin.it/wiki/Protocol_documentation#version. Accessed 5 Mar 2023
Wikipedia: cosine similarity (2022a). https://en.wikipedia.org/wiki/Cosine_similarity. Accessed 7 Mar 2023
Wikipedia: SimHash (2022b). https://en.wikipedia.org/wiki/SimHash. Accessed 7 Mar 2023
Zcash: Shared connection pool and deterministic eviction logic in Zcash (2023). https://github.com/zcash/zcash/blob/v5.4.2/src/net.cpp. Accessed 7 Mar 2023
Zheng B, Zhu L, Shen M, Du X, Guizani M (2020) Identifying the vulnerabilities of bitcoin anonymous mechanism based on address clustering. Sci China Inf Sci 63:1–15

Publisher's Note