Received 27 June 2022, accepted 25 August 2022, date of publication 5 September 2022, date of current version 15 September 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3204387

SURVEY

SoK: Network-Level Attacks on the Bitcoin P2P Network

FEDERICO FRANZONI[1] AND VANESA DAZA[2]
[1]QPQ, 6300 Zug, Switzerland
[2]Department of Information and Communication Technologies, Pompeu Fabra University, 08002 Barcelona, Spain

Corresponding author: Federico Franzoni (fed.franzoni@gmail.com)

This work was supported in part by AEI/FEDER, UE, under Project RTI2018-102112-B-I00.

ABSTRACT Over the last decade, Bitcoin has revolutionized the global economic and technological landscape, inspiring a new generation of blockchain-based technologies. Its protocol is today among the most influential for cryptocurrencies and distributed networks. In particular, the P2P layer represents a reference point for all permissionless blockchains, which often implement its solutions in their network layer. Unfortunately, the Bitcoin network protocol lacks a strong security model, leaving it exposed to several threats. Attacks at this level can affect the reliability and trustworthiness of the consensus layer, mining the credibility of the whole system. It is therefore of utmost importance to properly understand and address the security of the Bitcoin P2P protocol. In this paper, we give a comprehensive and detailed overview of known network-level attacks in Bitcoin, as well as the countermeasures that have been implemented in the protocol. We propose a generic network adversary model, and propose an objective-based taxonomy of the attacks. Finally, we identify the core weaknesses of the protocol and study the relationship between different types of attack. We believe our contribution can help both new and experienced researchers have a broader and deeper understanding of the Bitcoin P2P network and its threats, and allow for a better modeling of its security properties.

INDEX TERMS Blockchain, bitcoin P2P network, bitcoin security, survey.

I. INTRODUCTION

Over the past decade, blockchain has revolutionized the economic sector. Since the release of Bitcoin in 2008, more than 6000 cryptocurrencies have been launched in the market, currently generating a trading volume of more than 2 trillion dollars [1].

Despite the competition, Bitcoin is still the most important and widespread cryptocurrency, with a daily activity of more than 200k transactions and a market capitalization of more than 1 trillion dollars [2]. Furthermore, its decade-long history and its vast user base make Bitcoin highly relevant for other blockchains. In fact, thanks to a lively community of developers and researchers who constantly improve its protocol, Bitcoin is arguably the most advanced and experimented blockchain network in the wild. Notably, the Bitcoin network is also one of the largest in terms of participating nodes, with an average of more than 10.000 connected clients [3].

The associate editor coordinating the review of this manuscript and approving it for publication was Mohamed Elhoseny.

As such, the Bitcoin P2P protocol serves as a reference for other cryptocurrencies, many of which are directly based on it. Specifically, while the application and consensus layers can be very different in other blockchains, the underlying network protocol often overlaps with Bitcoin. As a consequence, issues in the Bitcoin P2P protocol directly reflect on other cryptocurrencies. This is especially relevant for what concerns security threats, which can undermine the guarantees of the above layers. It is then of paramount importance to properly study and address security threats in the Bitcoin P2P protocol.

This paper aims to help the Bitcoin community raise the bar against network-level adversaries. To that purpose, we review in detail all Bitcoin network-level attacks known in the literature, and analyze them with the help of a formal adversary model and a new taxonomy based on their goals or uses.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

a: BITCOIN SECURITY

Numerous studies in the literature covered the different aspects of Bitcoin security. Research works include analyses of the guarantees of the consensus protocol [4], [5], vulnerabilities at the application level [6], [7], and anonymity of the transaction graph [8]. However, while much effort has been expended on the application- and consensus-level security, relatively limited attention has been given to the underlying P2P network.

This gap can be partially attributed to an implicit assumption on the safety of Internet communications, which is somewhat inherent to any application-level protocol, where the security guarantees rely on that of the lower levels. Nonetheless, flaws and vulnerabilities in the network layer can lower or even invalidate the guarantees of Bitcoin application and consensus protocols. For instance, as we will show in this paper, numerous p2p-level attacks can be used for double-spending, which is supposed to be prevented in Bitcoin. A secondary cause of this gap is the lack of a formal definition and documentation of the Bitcoin P2P protocol, which greatly limits the ability to study its properties.

Some previous work partially covered Bitcoin network security as part of broader surveys. However, no work has specifically addressed it in a methodological way. To bridge this gap, we systematically gathered and reviewed available information on network-level attacks, extracting essential details and studying them in a network-level security framework.

In particular, we define a generic network adversary model that defines her characteristics and primary goals. We then classify attacks in three major categories: *Direct* attacks, which directly pursue one of the primary goals, *Infrastructure* attacks, which exploit weaknesses in external protocols, and *Auxiliary* attacks, which enable, ease, or improve other attacks. We review all known attacks in the literature by summarizing their procedure, impact, and implemented countermeasures. Additionally, we specify the corresponding adversary and targets according to our model.

Finally, we discuss reviewed attacks from a high-level perspective, inferring common characteristics and studying their relationship with each other. We use this information to identify the components of the protocol whose weaknesses are typically exploited to perform the attacks.

To allow a full understanding of attacks, we also provide a detailed, up-to-date description of the Bitcoin P2P protocol, based on both the reference implementation (*Bitcoin Core*), academic literature, and publicly-available information.

b: CONTRIBUTION

In summary, we make the following contributions:

- We provide a detailed, up-to-date description of the Bitcoin P2P protocol;
- We define a threat model framework to standardize the definition of network-level adversaries and targets;
- We propose a new objective-based taxonomy of network-level attacks;
- We review in detail all known network-based attacks in the literature, including their procedures, impact, and implemented countermeasures;
- We identify weak spots of the Bitcoin P2P protocol as inferred from the reviewed attacks;
- We show and study the relationship between attacks;
- We summarize the current status of the Bitcoin P2P protocol security, and indicate future research directions.

c: METHODOLOGY

In order to perform an exhaustive search of network-level attacks and their solutions, we gathered information from the following sources:

- Scholar-indexed research papers: we queried Google Scholar with related keywords, and retrieved all papers describing a network-level attack; we then recursively followed citations from and to retrieved papers to have a comprehensive list of published works;
- Bitcoin Core Github repository: we studied the official Bitcoin source code and explored Issues, BIPs (Bitcoin Improvement Proposals), and Pull Requests to understand if and how network-level threats have been addressed in the protocol;
- We complemented the information retrieved from the above-listed sources with online blogs and forums, which were often used to disclose and discuss network-level threats in the early years of Bitcoin.

From each retrieved source, we also extracted information about the Bitcoin P2P protocol and verified it against the official source code.

d: ORGANIZATION OF THE PAPER

First, in Section II, we discuss related work and motivate the need for this work. In Sections III and IV, we describe Bitcoin and its P2P protocol, focusing on the aspects involved in security attacks.

Then, we introduce our *Network Adversary* model in Section V, and our attack taxonomy in Section VI. In Sections VII to IX, we systematically review network-level attacks. Then, in Section X, we discuss all attacks and study their relation with each others. Finally, in Section XI, we summarize the current status of the Bitcoin network security, and indicate future research directions. Section XII concludes the paper.

II. RELATED WORK

Numerous surveys in the literature covered the security of blockchain networks. These works often provide similar contents but differ in their perspective and level of detail.

For instance, Saad *et al.* [9] explore the attack surface of public blockchains dividing attacks into three broad categories: blockchain, P2P network, and application context. For each attack, the authors give a generic description and report real-life case studies. Similarly, Wen *et al.* [10] review attacks on blockchain, based on a six-layer framework: data, network, consensus, incentive, smart contracts, and application. For each attack, the authors provide a description and a list of countermeasures. With a different approach, Hassan *et al.* [11] perform a systematic literature review on the security of cryptocurrencies. In their work, threats are categorized by blockchain component: cryptographic primitives, transactions, consensus protocol, and P2P network.

While the above-mentioned works covered blockchain networks in general, other works specifically focus on Bitcoin. Vyas and Lunagaria [12] shortly summarized attacks such as majority attack, selfish mining, double spending, timejacking, and attacks to wallet software. Similarly, Shalini and Santhi [13] cover major attacks on Bitcoin such as DoS, double spending, sybil, and eclipse.

A more extended survey by Conti *et al.* [14] thoroughly investigates security and privacy issues in Bitcoin. Their analysis includes most attack vectors associated with the use of bitcoins, including consensus, network, and client-side threats, as well as attacks to exchanges and mining pools. For each attack, they review its primary targets, adverse effects, and possible countermeasures.

Due to their widespread spectrum, these works study network attacks at a high level, leaving to the reader the burden of retrieving details from other sources. This task can be particularly cumbersome, as their peculiarities can be hard to understand. In turn, the protocol itself is poorly documented and constantly evolving, making the retrieval and verification of information even harder.

At the same time, most surveys highlight the importance of network-layer security. For instance, Wen *et al.* [10] stress that the security of a blockchain should entail that of the underlying network infrastructure. Saad *et al.* [9] conclude that the P2P architecture is the most dominant element of the blockchain attack surface, due to the permissionless nature of the network, which allows any malicious actor to join (we confirm this point § X).

Nevertheless, as noted by Hassan *et al.* [11], despite the numerous threats to the networking layer, very few papers address them specifically. Indeed, we were able to find only two works focusing on network-level security: a short paper by Cao *et al.* [15], which briefly reviews routing attacks and deanonymization, a work by Mostafa *et al.* [16], which covers a vast range of attacks in the Bitcoin network. While the latter is the closest to our work, attacks and countermeasures are still described at a high level, without showing the details of how they are performed. Moreover, important attacks related to double-spending and mining are left out of scope.

Our work provides a detailed description of all known network-level attacks, along with a review of their impact, and implemented countermeasures. We model and compare attacks under a new framework, and quantify their impact on Bitcoin security. Ultimately, our work provides researchers with a comprehensive compendium for network-level security in Bitcoin.

III. BITCOIN

Bitcoin is a cryptocurrency, proposed in 2008 by Satoshi Nakamoto [17], which allows users to exchange money (also called Bitcoin) without the use of a central authority. Users can participate freely by joining a P2P network with compatible client software [18].

To operate with Bitcoin, users make use of asymmetric keys, of which the public part is used, in the form of a derived *Bitcoin address*, to receive coins, and the private part is used to spend them. In particular, coins are transferred by means of *transactions*, which specify the recipient by its address. In turn, the recipient can spend such coins by digitally signing a new transaction with its private key. To verify the validity of new transactions, clients maintain a ledger in the form of a *blockchain*, which stores all past transfers.

Transactions are added to the ledger by special users called *miners*, who compete against each other to create new *blocks*. To generate a valid block, miners have to find (by brute-force search) a *nonce* value that, hashed with the block header, yields an output below a certain threshold. This threshold, known as *difficulty*, is periodically adjusted to have an average production rate of one block every ten minutes. Since creating a block requires a non-trivial amount of computation, the nonce is regarded as a *Proof of Work* (PoW). Network nodes independently store a copy of the ledger,[1] and verify the validity of blocks as they are received.

e: CONSENSUS

Due to mining competition, it is possible that two different blocks are created approximately at the same time. This generates a so-called *block race*, in which the two blocks compete in being accepted as the new chain *tip*. When this occurs, two separate blockchain branches, or *forks*, will co-exist for some time. As nodes can only accept one of the two blocks, the network will split into two parts, each using a different fork.

To resolve this situation, a *consensus* mechanism is used, which allows the nodes to agree on a single fork. In particular, Bitcoin nodes always choose the fork with the most cumulative PoW (typically the longest one). This mechanism, known as *PoW consensus*, allows the network to automatically solve conflict situations, and have all nodes use a single branch (known as the *main chain*).

As we will see in § VIII-E, the occurrence of a block race can be exploited by an attacker to perform double spending (§ IX-B) and selfish mining (§ IX-D1).

f: TRANSACTION CONFIRMATION

When a transaction is included in a block of the main chain, it is said to be *confirmed*. Similarly, each new block subsequently appended on top of it acts as an extra confirmation (because it implicitly accepts the previous block as valid). Specifically, a transaction is said to have n confirmations if there are $n - 1$ blocks appended on top of the confirming block. In contrast, a transaction that has not yet been included in the blockchain is said to be *unconfirmed*.

[1]For the sake of simplicity, we only consider *full nodes* in this work, excluding clients that do not download or store the entire blockchain.

As we will see in § IX-B, accepting transactions with 0 or few confirmations, expose users to Double-Spending attacks. For this reason, Bitcoin developers recommend users to wait a minimum of 6 confirmations before accepting a transaction as a payment.

g: MINING POOLS

Given the high costs of the mining process, and the low chances of beating the competition, working alone is hardly profitable in Bitcoin. For this reason, miners typically join forces by creating *pools*. In a mining pool, several miners work in parallel to find the PoW for the next block. If the block is produced and added to the main chain, the rewards are split among all members of the pool. This mechanism allows amortizing the risks, making profits more stable over time. Currently, although solo mining is still possible, joining a pool is the only way to make this activity profitable.

The existence of mining pools affects the ability to perform double-spending attacks where the attacker needs to produce her own blocks (see § IX-B).

h: RELAY NETWORKS

To speed up the exchange of blocks, miners often interconnect outside the Bitcoin P2P network through fast-speed *relay networks*. Currently, the best-known ones are FIBRE [19] and Falcon [20].

As we will see, relay networks help protect from delay attacks (§ VIII-E) and makes partitioning (§ VIII-D) harder.

A. THE BITCOIN NETWORK

The Bitcoin network constantly changes over time, due to nodes joining and leaving, as well as to protocol updates, which change how nodes connect to each other.

i: NETWORK ADDRESSES

Four types of addresses are currently supported in Bitcoin: IPv4, IPv6, OnionCat, used by Tor [21], and I2P [22], another anonymity network.

j: PUBLIC NETWORK

Currently, the network counts an average of 15.000 *reachable* (i.e., accepting connections) nodes [3]. Of these, IPv4, between 6000 and 8000 units, constituted the vast majority (70-80%) until 2021. However, over the past year, Tor nodes have surged from around 1000 to more than 7000 units, surpassing IPv4 nodes for the first time [23]. In contrast, the number of IPv6 nodes, around 1200, has not varied over time.

Interestingly, this portion of the network is overall very stable, with very few changes in the connected nodes. As discussed in § X, reachable nodes are the primary target of most network-level attacks.

k: UNREACHABLE NETWORK

Not all nodes in the Bitcoin network are reachable. In fact, some nodes do not accept incoming connections due to configuration settings or, more commonly, for being behind a firewall or NAT [24]. Notably, according to estimates, these nodes represent around 90% of the network [25].

Unlike reachable nodes, unreachable ones only establish outgoing connections, thus having much less varied connectivity compared to their peers. This characteristic makes them more vulnerable to deanonymization attacks (§ IX-C). On the other hand, as discussed in § X, their inability to accept connections from other peers, protects these nodes from a large number of other attacks.

l: TOPOLOGY

Bitcoin nodes choose their peers in a pseudo-random way, with the goal of creating a random-looking topology graph [26].

However, for security reasons, information on the actual Bitcoin network topology is hidden in the protocol. At the same time, several techniques have been found to infer connections among reachable nodes (we describe such techniques in § VIII-B). Thanks to these techniques, an approximation of the actual network graph has been calculated, which proved to be relatively centralized. In particular, studies revealed the presence of highly connected nodes (probably used as gateways by mining pools) [26] and highly interconnected communities [27].

As we will see in § VIII-B, knowledge of the topology graph can facilitate attacks like Double Spending (§ IX-B) and Deanonymization (§ IX-C), and enable Partitioning attacks (§ VIII-D).

IV. THE BITCOIN P2P PROTOCOL

The Bitcoin P2P protocol defines how nodes choose their peers and spread information through the network. In this section, we review the most relevant details of the protocol, with particular emphasis on those related to network-level attacks.

m: SPECIFICATION

While few resources are available to learn how the Bitcoin protocol works at a high level [28], [29], no official documentation has ever been released. Therefore, the protocol is only specified by its reference implementation, that is the Bitcoin Core client sources [30]. As a consequence, obtaining up-to-date information on how Bitcoin nodes behave can be challenging.

In this section, we provide a detailed description of the Bitcoin P2P protocol, focusing on the aspects relevant to network-level attacks. Our description is based on the Bitcoin Core source code and related documents on the official Github repository, and complemented with information retrieved from the literature. We refer to the latest version of the client, *v0.22*. Moreover, in the rest of the paper, we will refer to previous versions, with the notation ‘‘v0.x’’, to highlight important changes to the protocol.

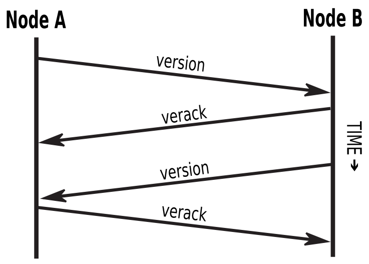

FIGURE 1. Connection establishment.

n: CONNECTIONS

Bitcoin nodes use TCP/IP to establish connections with their peers. Public nodes usually accept connections on the standard port 8333, although any available port can be used.

Since connections are not encrypted, all exchanged messages are sent in clear. As we will see in § VII-B, this exposes Bitcoin connections to Man-in-the-Middle attacks, with important consequence for the security of the network.

o: PROTOCOL MESSAGES

To understand the details of network-level attacks, it is often important to be familiar with the specific protocol messages. Here, we give a list of all the relevant ones.

- **VERSION/VERACK**: exchanged to establish a new connection.
- **PING/PONG**: exchanged to keep the connection alive.
- **GETADDR**: it requests a list of known active peers.
- **ADDR**: sent unsolicited or in response to GETADDR, it contains a list of known Bitcoin nodes (in the form *IP*:*port*).
- **INV**: sent unsolicited or in response to GETBLOCKS, it contains a list of known transactions or blocks.
- **GETDATA**: it requests the data of blocks or transactions.
- **TX**: sent in response to GETDATA, it contains a single transaction.
- **BLOCK**: sent in response to GETDATA, it contains a single block.
- **GETHEADERS**: it requests block headers starting after a specified block.
- **HEADERS**: sent in response to GETHEADERS, it contains a list of block headers.
- **SENDCMPCT/GETBLOCKTXN/BLOCKTXN**: messages used for Compact Blocks.
- **MEMPOOL**: requests the mempool content; the response is an INV message containing the list of all transactions in the mempool.

A. PEERING

In this section, we describe how Bitcoin clients establish and manage connections with other peers.

p: CONNECTION ESTABLISHMENT

New connections are established with a three-way handshake, as shown in Figure 1. Specifically, when a node N_A wants to connect to node N_B, the following messages are exchanged:

1) N_A sends VERSION to N_B;
2) N_B sends VERACK and VERSION to N_A;
3) N_A sends VERACK to N_B.

The VERSION message includes information about the node (client name, current protocol version, services provided, etc...) as well as the current *tip* of the blockchain (i.e., the last known block), which is used to synchronize the ledger. Additionally, a timestamp is included, which is used to synchronize a *network clock*.

The connection is said to be *outbound* for the node that initiates the handshake, and *inbound* for the other node. Similarly, a peer is outbound or inbound depending on its connection type.

To keep the connection alive, nodes send a PING message every 30 minutes, to which the peer replies with PONG messages. If no message is received during 90 minutes, the connection is closed.

1) PEER MANAGEMENT

Bitcoin nodes usually maintain 8 outbound peers and, if reachable, accept up to 117 inbound peers.

In addition, since v0.19, two extra *block-relay-only* outbound connections are also maintained by each node [31]. This feature mitigates Topology Inference attacks (§ VIII-B), thus protecting from network partitioning (§ VIII-D). Since v0.21, these nodes are also used as *anchors*: when restarting, the node will try to re-connect to the same block-relay-only nodes it was connected before [32]. This feature mitigates the risk of Eclipse attacks (§ VIII-C).

Since v0.12, if all inbound slots are occupied, and a new connection request is received, an *eviction* mechanism is used to disconnect one of the current inbound peers [33]. This mechanism mitigates the risk of an attacker controlling all inbound connections at the same time (see § VIII-C). In particular, the choice of the evicted peer is made to minimize the possibility of the attacker to avoid being disconnected.

a: PEER DISCOVERY

When a Bitcoin client runs for the first time, a *bootstrapping* procedure is executed to discover other nodes. This procedure queries a list of DNS servers, called *seeds*, which maintain an up-to-date list of known IP addresses where a reachable Bitcoin node is running. When queried, DNS seeds return a random subset from such list. The client will then choose 8 random addresses as outbound peers. If the DNS method fails, a hard-coded list of known stable nodes is used.

Once the client has joined the network, it learns new addresses from other peers. Specifically, when a new connection is opened, the client sends a GETADDR message, to which the peer replies with an ADDR message with a list of known addresses. All the addresses learned by the client are stored in a local database, which is periodically saved to file to be used subsequent executions. This means DNS seeds are usually only needed in the first execution of a client.

In fact, as discussed in § VII-D, the use of DNS seeds is undesirable, as it is can lead to node eclipsing (§ VIII-C).

b: THE ADDRESS DATABASE

The address database is used by clients to choose new outbound peers, and to reply to `GETADDR` requests. It is composed of two tables: `tried` and `new`. The `tried` table contains addresses to which the node has successfully established an outbound connection. The `new` table contains all other known but untested addresses, advertised from peers or DNS seeds. Tables are further divided into groups, called *buckets*, which store up to 64 addresses each. Entries are in the form of '`IP:port`' tuples, with an attached timestamp indicating their *freshness*. Specifically, the `tried` table has 256 buckets, while the `new` table has 1024.

When an address is added to the `new` table, a bucket is selected depending on the prefix (e.g., /16 for IPv4) or Autonomous System (AS), as well as the source (peer or DNS) address. Because of this, a single address can be stored in multiple buckets (up to 8) if advertised by different peers. This increases the chances of selecting addresses that are advertised more. When an address maps to an occupied slot of a bucket, the existing address is overwritten if it is also stored in another bucket, or it is considered *terrible*, that is, it does not meet some quality criteria (e.g., if older than 30 days or has more than 3 failed connection attempts).

If an address in `new` is successfully connected, it is moved to `tried`. Here, bucket selection is based on both the network prefix of the address and the full address. If the assigned slot is occupied, the currently-stored address is tested for eviction: if offline, it is moved to `new`, and replaced by the new address.

To increase the number of addresses in `tried`, a random entry in `new` is tested every two minutes with a short-lived *feeler* connection. If the feeler connection is successful, the address is moved to the `tried` table.

As later shown, the address database is an essential component, and a bad design can be used for fingerprint (§ VIII-F) and eclipse (§ VIII-C) attacks.

c: PEER SELECTION

When a new outbound connection has to be established, the candidate peer is chosen among entries in the address database using a pseudorandom procedure.

First, the `new` or `tried` table is selected with a 50% chance. Then, a bucket and slot are chosen uniformly at random. If the slot is empty, the procedure iterates through subsequent positions until an address is found. If the whole bucket is empty, a new one is picked at random. As under-populated buckets will have more chances of being selected, this procedure favors addresses in under-represented network zones.

2) ADDRESS PROPAGATION

When connecting to a peer for the first time, most nodes advertise their address with an `ADDR` message. Subsequently, a `GETADDR` message is sent to the new peer to request a list of other known addresses. In turn, the peer returns this list via an `ADDR` message, which is stored by the requesting node into the address database. When replying to a `GETADDR` message, up to 23% of the addresses stored in the database is sent. However, since 2020 (v0.21), to prevent Topology Inference attacks (§ VIII-B), nodes only send up to 1000 addresses per day, in response to `GETADDR` messages [34]. Additionally, only one `GETADDR` message will be replied for each connection.

`GETADDR` messages are typically only used at the beginning of a connection. After, newly discovered nodes can be advertised through unsolicited `ADDR` messages. When the client receives an `ADDR` message with up to 10 addresses, it forwards it to a limited number of peers. Specifically, reachable addresses[2] are advertised to two peers, while unreachable ones to a single one. For each peer, a node remembers which addresses have been forwarded, so as to avoid repeating their advertisement. This information is kept for each connection (i.e., per session, not per IP) and cleared every 24 hours. Recipient peers are also fixed during 24 hours, so that each address is typically propagated once per day. This prevents a peer from increasing the propagation of its own address by repeatedly advertising it.

Note that, for privacy concerns, the transmission or `ADDR` messages is implemented using the Diffusion protocol, described in § IV-B1.a.

3) PEER SECURITY

One of the risks in a permissionless network is to have an attacker control a large portion of our peers. To mitigate this risk, the Bitcoin protocol adopts several protective measures.

a: OUTBOUND PEERS

Outbound peers are generally considered safer than inbound ones. This is due to the fact that an attacker can easily open multiple inbound connections towards the same node (when reachable). In contrast, she has limited control over outbound connections besides deploying multiple nodes. As shown in § IV-B, this fact is taken into account in several choices throughout the protocol, and is at the base of various countermeasures against network-level attacks (see § VIII).

b: IP-RANGE LIMITATIONS

Some protocol choices are based on the assumption that a single adversary, even when controlling numerous IP addresses, is not likely to control nodes in many different geographical areas. For example, this is one of the reasons why buckets in the address database are based on the prefix or AS. Another limitation is on the number of outbound connections that can be established towards the same network area, which is limited to one per group (e.g. /16 for IPv4).

[2]In this context, reachability is simply determined by the network family of the address (i.e., IPv4, IPv6, OnionCat, I2P). Specifically, a node considers an address reachable if they both belong to the same family.

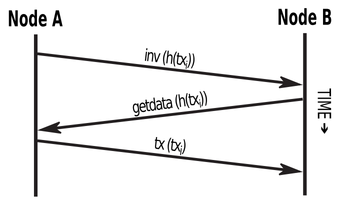

FIGURE 2. **Bitcoin transaction relay.**

c: PEER REPUTATION SYSTEM

To prevent misbehaving, Bitcoin nodes maintain a reputation score for each peer, which is increased every time the peer deviates from the protocol. The value of the increase depends on the gravity of the misbehavior. For instance, sending a message before the connection handshake determines an increase of 1, while sending an invalid block generates an increase of 100.

When a peer reaches a score of 100, it is disconnected, and its address is marked as *discouraged*. Nodes connecting from a discouraged address are allowed to open inbound connections, but they are preferred for eviction when accepting new incoming connections. Moreover, they are excluded from outbound connections, and they are not advertised to other peers. The duration of the discouraged status can last an indeterminate time, but it does not persist after reinitiating the node.

Until v0.18 [35], misbehaving nodes were disconnected and banned for 24 hours. However, this feature has been shown to allow DoS attacks on Tor nodes (see § VII-F).

B. DATA PROPAGATION

To maintain the distributed ledger, two main types of information must be spread through the network: transactions and blocks. Generally speaking, this information is propagated through *gossip*, that is, when a node receives new data, it relays it to all its peers. However, this can occur in different ways. In the following, we describe the procedures used in Bitcoin to propagate transactions and blocks.

1) TRANSACTION PROPAGATION

When a node generates or receives a transaction, it relays it to all connected peers. To avoid having a node receive the same transaction twice, a 3-step relay process is used, as shown in Figure 2. First, the transaction is announced by sending its hash identifier, or *TXID* ($h(tx_i)$ in the image), via an `INV` message; the receiver checks whether the *TXID* is known, and, if not, requests the transaction with a `GETDATA` message; upon receiving this request, the announcer transmits the transaction using a `TX` message.

After sending the `GETDATA` message, a timeout is set (currently to 1 minute) to avoid excessive delays. If the timeout expires, the transaction is requested to another peer that announced it, if any.

To mitigate the risk of Delay attacks (described in § VIII-E), the request queue is currently randomized, that is, it does not follow the order in which peers advertised the transaction. With the same purpose, outbound nodes are currently preferred over inbound ones when requesting a transaction.

a: TRANSACTION RELAY

Currently, Bitcoin transactions are spread using *Diffusion*. In this protocol, when the client receives a new transaction, it does not relay it immediately. Instead, it delays its transmission to mitigate the risk of Danonymization attacks (§ IX-C). In particular, nodes maintain a queue for each peer, containing the transactions to be forwarded. This queue is periodically flushed by sending an `INV` message to the peer, announcing all pending transactions. Each queue is flushed with an independent, exponential delay, which follows a Poisson distribution [36]. This delay is halved for outbound peers, since these are less risky for deanonymization.

Diffusion was introduced in 2015 (v0.12) to replace the *Trickle* protocol [37]. This protocol implemented the so-called *trickling*, that is, the addition of random delays to the relay of a message. In particular, in Trickle, at each loop iteration, a random node was selected, and its queue flushed. The protocol switch was done to mitigate Deanonymization attacks (§ IX-C).

As a protection against DoS attacks, all transactions are also required to pay a *minimum relay fee* (MRF) to be propagated.

b: UTXO SET

When a new transaction is received, its validity is verified by checking its inputs. To speed up this process, all known unspent outputs are stored in the *Unspent Transaction Output (UTXO)* set. To protect from Flooding attacks (§ IX-A3), the size of this set is limited.

c: ORPHAN POOL

If a transaction is received before its *parent* transactions, it is considered an *orphan*. Since such transactions cannot be validated, they are not immediately relayed to peers. Instead, they are stored in an *orphan pool* so they can be validated if the parent is received. Similar to the UTXO set, the size of this buffer is also limited (currently to 100 entries).

d: SEGREGATED WITNESS

In 2016 (v0.13), the Segregated Witness (SegWit) [38], [39] soft fork was activated, which allows separating validation data from the transaction structure. In particular, it allows to store transaction scripts and signatures into a new, separate, structure, called *witness*.

SegWit transactions have two IDs: the standard *TXID*, obtainded by hashing the transaction structure, and the *WTXID*, obtained by hashing base data and witness data. This fact solved Transaction Malleability (§ VII-A), which could lead to DoS attacks (see § IX-A3) and Double Spending (§ IX-B7).

Furthermore, by reducing transaction base data, SegWit allows storing more transactions into a single block, which in turn improves the overall network throughput.

2) BLOCK PROPAGATION

Bitcoin nodes propagate blocks in different ways. When connecting to a new peer, they perform the *block synchronization* procedure. Then, they simply relay new blocks when received. In turn, this can be done using the legacy INV/GETDATA method, the SENDHEADERS method, or using *compact blocks*. In the following, we describe all these procedures.

a: BLOCK SYNCHRONIZATION

Block synchronization is performed by two nodes when establishing a new connection. In this procedure, the newly-connected peers exchange their *tip* (last known block index) in the VERSION message. Then, the node with the lower tip (i.e., the one with fewer blocks in the ledger) sends a GETBLOCKS message requesting missing blocks, to which the peer replies with an INV message containing the corresponding block hashes. Finally, each block is requested with a GETDATA message.

Block synchronization is particularly relevant for nodes joining the network for the first time, or re-joining after leaving it for some time. Once a node has synced its ledger, it will update it as peers relay new blocks.

This procedure has been shown to allow a possible Fingerprint attack (see § VIII-F3).

b: BLOCK RELAY

In earlier versions of Bitcoin, blocks were relayed following a procedure similar to transactions. In this procedure, the relayer sends an INV message announcing the new block; then, the receiver asks, if needed, the block header (via GETHEADERS), and eventually requests the block with GETDATA.

Since 2016 (v0.12), for efficiency reasons, nodes can instruct their peers (via a SENDHEADERS message) to announce blocks by sending HEADERS messages directly, skipping the INV message [40], [41].

Similar to transactions, a timeout is set after sending a GETDATA request. Currently, the timeout is set to a minimum of 10 minutes, plus 5 extra minutes for each additional block being downloaded at the same time. If the timeout expires, the corresponding peer gets disconnected (this behavior is mostly because there is no easy way to interrupt an ongoing download).

Note that, different from transactions, new blocks are advertised to other peers with no additional delays after being added to a node's ledger, since there are no privacy concerns in this case. However, the transmission timeout could be used to implement Delay attacks (§ VIII-E).

c: COMPACT BLOCKS

With version 0.13, *Compact Blocks* were also introduced [42], [43] to reduce bandwidth consumption.

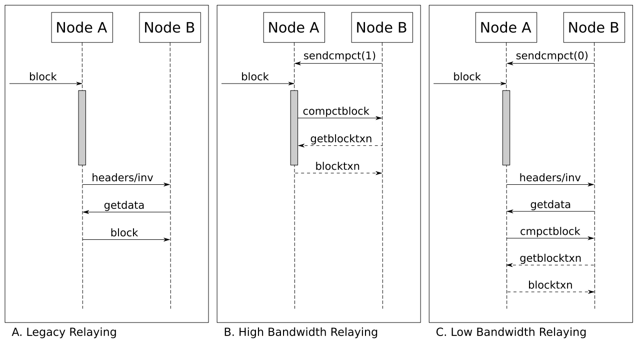

FIGURE 3. Bitcoin block relay [42].

A compact block can be sent in reply to a GETDATA message using a COMPCTBLOCK message, which includes the block header, the list of *TXID*s in the block, and a small set of full-data transactions that the sender believes have not been seen by the receiver. When a compact block is received, nodes check the list of *TXID*s to verify whether they are known. Missing transactions are then requested (by their index in the list) using a GETBLOCKTXN message. The peer will then send the requested transactions with a BLOCKTXN message.

Compact blocks also introduced the concept of *High Bandwidth* (HB) neighbors, which can be instructed to relay blocks unsolicited, that is, without previously announcing them. HB nodes send a compact block unsolicited if they believe the receiver knows the previous block but not the new one. This strategy allows to both receive blocks faster and mitigate DoS/Delay attacks (see § VIII-E).

Each node maintains an HB Neighbor (HBN) list, composed of the last three nodes that relayed a valid block. To that purpose, when a new valid block is received, the sender is added to the HBN list. If there already are three nodes in the list, the one that sent a block least recently is evicted. To instruct a peer in the HBN list to send blocks unsolicited, nodes send a SENDCMPCT(true) message. If such a node is later evicted, a SENDCMPCT(false) message is sent. The new block propagation scheme is depicted in Figure 3, along with the old relay protocol.

d: BLOCK PROCESSING

Like transactions, new blocks are validated by each node when they are received (via BLOCK messages). This process involves verifying all the transactions included in the block.

In particular, when a block is received, all included transactions are expected to be in the *mempool* (missing transactions are requested to the node that advertised the block). After the block is validated, the mempool is emptied. In other words, the mempool acts as a cache of unconfirmed transactions, which remain in memory until they are included in a block.

This structure can represent a bottleneck for Bitcoin nodes, since if the transaction's arrival rate exceeds that of mining, the mempool size increases, slowing down the verification process (and overall performances of the network) [44]. As we will show in § IX-A, this fact can be exploited to perform DoS attacks.

V. THREAT MODEL FRAMEWORK

Threat models allow to formalize the study of specific category of attacks and enable the comparison between different attacks and countermeasures. For instance, [45] defines the Eavesdropper adversary to study deanonymization attacks and quantify the anonymity guarantees of different propagation protocols. Several other adversary models have been defined across the literature to address different scenarios. However, these models are typically defined as an ad-hoc framework for a specific attack, which does not allow it to be applied to different scenarios. At the same time, for many attacks, no threat model has every been defined, making analysis and comparison difficult.

To bridge this gap, we propose a generic threat model framework that allows to define adversary and target models for all network-level attacks. We based our framework on the various threat models found in the literature.

Our framework allows to easily define the threat model of any network-level attack, and, in turn, to evaluate its risk in a quantifiable manner. Specifically, each threat model in our framework allows to define both the targets (which nodes are susceptible to the attack? how many are affected?), and the adversary (who can perform the attack? does she require special privileges? what knowledge and resources are needed for the attack?).

To allow an easier and quick specification of the threat model for each attack, we will also introduce a uniform notification, which will be described in the rest of this section.

A. TARGETS

When considering the entities affected by an attack, we distinguish between network-level devices, referred to as *targets*, and real-life users, referred to as *victims*.

In particular, from a network perspective, the target $\mathcal{T}$ of an attack can be:

- A single node (N), which can be reachable (N^R), or unreachable (N^U);
- Multiple nodes: this can be the whole network (Net), a generic portion (Net^P), all reachable nodes (Net^R), all unreachable nodes (Net^U).

On the other hand, from an user perspective, the victim $\mathcal{V}$ can be:

- A seller or merchant ($\mathcal{S}$), typically accepting paying transactions after i confirmations ($\mathcal{S}^i$);
- A payer or exchange ($\mathcal{P}$);
- A miner or mining pool ($\mathcal{M}$).

Note that the concepts of target and victim can be independent and coexist, depending on the attack. For instance, an attack can have a target but no victim (except the implicit owner of the node). Similarly, an attack can have a victim without affecting a specific target.

B. ADVERSARY

We define a *Bitcoin network adversary* $\mathcal{A}$ based on the generic model suggested by Do *et al.* [46]. In particular, we define $\mathcal{A}$ by specifying the following characteristics:

- *Goals*;
- *Assumptions*: environment, knowledge, and resources;
- *Capabilities*: interaction, and connectivity.

Additionally, we introduce the following notation to specify the adversary model for each attack:

$$\mathcal{A}^{interaction}_{environment}([knowledge], [resources]) \triangleright connectivity.$$

For instance, the notation

$$\mathcal{A}^{O}_{AS}(T, B, M) \triangleright T$$

denotes an AS-level adversary, with observing capabilities, who knows the IP of the target, controls a botnet, has mining equipment, and connects to the target's node. To complement the security framework, we also define the possible targets of a network-level attack.

e: GOALS

- *Denial of Service* (*DoS*): $\mathcal{A}$ aims at disrupting or degrading the functionality of the Bitcoin network.
- *Double Spending* (*DS*): $\mathcal{A}$ aims at spending the same bitcoins twice, typically once to a merchant to obtain a good or service, and the other one to herself to revert the payment.
- *Deanonymization* (*D*): $\mathcal{A}$ aims at distinguishing the user who created a specific transaction, or at recognizing transactions from a specific user.
- *Unfair Revenue* (*UR*): $\mathcal{A}$, participating as a miner, aims at gaining more revenues than its due share according to the computation done.

f: ASSUMPTIONS

- Environment:
  - *P2P-level* ($\mathcal{A}_{P2P}$): $\mathcal{A}$ acts as a Bitcoin node, and interacts with other nodes according to the protocol; this is the most common type of adversary, which only exploits weaknesses in the Bitcoin protocol to achieve her objective; since this is the most common case, we omit this notation.
  - *AS-level* ($\mathcal{A}_{AS}$): $\mathcal{A}$ (typically an ISP) has control of one or more Autonomous Systems (ASs) used by Bitcoin nodes to connect; this adversary takes advantage of her privileged position in the network to enable Man-in-the-Middle capabilities (§ VII-B).
- Resources:
  - *Server*: $\mathcal{A}$ has unlimited bandwidth, memory, and storage; we implicitly assume this resource and hence omit its notation.
  - *Botnet* ($\mathcal{A}(B)$): $\mathcal{A}$ controls multiple nodes, possibly with different IP addresses; this is used to impersonate multiple identities in the network, such as in Sybil attacks (§ VIII-A).
  - *UTXOs* ($\mathcal{A}(U)$): $\mathcal{A}$ has spendable Bitcoin outputs; this is needed to create new transactions, like in Dos (§ IX-A) or Double Spending (§ IX-B) attacks.

  - *Mining equipment* ($\mathcal{A}(M)$): $\mathcal{A}$ is able to mine her own blocks; this is typically needed in Double Spending attacks (§ IX-B).
- Knowledge:
  - *IP* ($\mathcal{A}(\mathcal{T})$): $\mathcal{A}$ knows the IP address of the target node(s); this is needed to connect to the target, which is needed in numerous attacks, like Delay (§ VIII-E) or Eclipse (§ VIII-C); we omit this notation when $\mathcal{A}$ connects to all reachable nodes, since this information is publicly available [3].
  - *Topology* ($\mathcal{A}(Net^R)$): $\mathcal{A}$ knows the topology of the reachable network; this is needed to observe/control the propagation of data, like in Deanonymization attacks (§ IX-C), or to exploit weaknesses in the connectivity of the network, like in Partitioning attacks (§ VIII-D).

g: CAPABILITIES

- Interaction:
  - *Observation* ($\mathcal{A}^O$): $\mathcal{A}$ observes and analyzes *data objects* (i.e., transactions, blocks, and addresses) received from other peers, but do not inject any in the network; this models some stealthy attacks like some early Deanonymization (§ IX-C) and Topology Inference (§ VIII-B) techniques.
  - *Injection* ($\mathcal{A}^I$): $\mathcal{A}$ injects or tamper with data objects and related messages; this models attacks where the adversary actively interacts with the network.
- Connectivity:
  - *Target* ($\mathcal{A} \triangleright \mathcal{T}$): $\mathcal{A}$ is connected to the target node(s); this is typically needed for Double Spending (§ IX-B) and Delay (§ VIII-E) attacks.
  - *Miners* ($\mathcal{A} \triangleright \mathcal{M}$): $\mathcal{A}$ is connected to one or more miners; this is typically needed in Double Spending attacks (§ IX-B).
  - *Network-wide* ($\mathcal{A} \triangleright Net^R$): $\mathcal{A}$ is connected to all reachable nodes at once[3]; this is typically needed for Deanonymization (§ IX-C) and Topology Inference (§ VIII-B).

VI. NETWORK-LEVEL ATTACKS

In the following sections, we will review all the network-level attacks we found in the literature. We organize such attacks based on a objective-oriented taxonomy, which we describe in this section. Additionally, we provide a small guide to the unified format we use for the attack description.

h: TAXONOMY

We classify network-level attacks as follows:

- *Direct Attacks* (§ IX): it includes attacks used to directly achieve one of the primary goals, such as *Denial of Service*, *Double Spending*, *Deanonymization*, and *Unfair Revenue*;
- *Auxiliary Attacks* (§ VIII): it includes attacks used to enable, ease, or improve Direct attacks or other Auxiliary attacks, such as *Sybil*, *Topology Inference*, *Delay*, *Eclipse*, *Partition*, and *Fingerprint* attacks.
- *Infrastructure Attacks* (§ VII): it includes Auxiliary attacks exploiting weaknesses in external protocols used by Bitcoin, such as *Man-in-the-Middle*, *Spoofing*, *Transaction Malleability*, *DNS*, *Tor*, and *BGP* (AS-level).

This taxonomy focuses on the relation between each attack and the achievement of the adversary's goals. As such, this separation does not reflect the relevance and impact of the attacks, meaning that Direct attacks are not more important than Auxiliary ones. In fact, as we will show, Auxiliary attacks are often a far more prominent threat in today's Bitcoin network.

i: ATTACK DESCRIPTION STRUCTURE

Attack descriptions follow this structure:

- *Threat model*: it specifies the adversary and targets according to the framework described in § V;
- *Procedure*: it describes the attack in detail;
- *Impact*: it discuss the level of vulnerability of Bitcoin, the success rate of the attacks, and possible real-life cases;
- *Limitations*: it mention known drawbacks of the attack;
- *Mitigation*: it describes defensive strategies and countermeasures implemented in the Bitcoin protocol.

Furthermore, for each Auxiliary attack type, we include a *Use* paragraph showing how the attack can be used to achieve primary goals or enable other attacks.

Note that, a missing paragraph in the description indicates the lack of relative information for the attack. For instance, if no countermeasure has been implemented, the Mitigation paragraph will be omitted.

j: NOTATION

In addition to the notation introduced for the Adversary Model, we will use the following object identifiers:

- tx: used for transactions;
- B: used for blocks;
- N: used for nodes.

We combine this notation with $\mathcal{A}$, $\mathcal{T}$, and $\mathcal{V}$ to specify whether the object belongs to, or is intended for, the adversary, the target, or the victim, respectively. For instance, in double-spending attacks, $\mathcal{A}$ generates a transaction $tx_{\mathcal{V}}$ for $\mathcal{V}$, and a conflicting one $tx_{\mathcal{A}}$, for herself. Similarly, $\mathcal{A}$ will use her own node $N_{\mathcal{A}}$ to connect to the target node $N_{\mathcal{T}}$.

k: METRICS

Some of the attacks, such as Topology Inference, Fingerprinting, and Deanonymization, are typically evaluated as *classification* techniques. In that respect, the following metrics are

[3]Note that $\mathcal{A}$ cannot connect to all unreachable nodes at once, given such nodes do not accept inbound connections

used:

$$Precision=\frac{|TP|}{|TP|+|FP|} \qquad Recall=\frac{|TP|}{|TP|+|FN|},$$

where $TP=TruePositives$, $FP=FalsePositives$, and $FN=FalseNegatives$.

These metrics are used to quantify the effectiveness of a technique. In particular, the *Precision* metric measures the ratio of correct identifications (e.g., how many inferred connections actually exist in the network), while *Recall* measures the ratio of correctly identified objects (e.g., how many of the existing connections are inferred).

VII. INFRASTRUCTURE ATTACKS

Being built on top of the Internet, Bitcoin communications rely on other protocols, such as TCP and DNS. Similarly, other protocols are used to provide different features. The security of the Bitcoin protocol can thus be affected by weaknesses in external mechanisms.

In this section, we review attacks that exploit vulnerabilities in other protocols or attack vectors produced by their combination with Bitcoin. Note that, similar to Auxiliary attacks, Infrastructure attacks enable the adversary to perform one or more Direct attacks.

A. TRANSACTION MALLEABILITY ATTACKS

Transaction malleability [47] refers to the ability to modify a signed transaction without invalidating it. This can be done in different ways, such as changing the transaction data (e.g., the scripts), or the digital signature [48], [49]. In particular, this can be done without any knowledge of the private keys used to sign the transaction.

In Bitcoin, this allows creating (valid) copies of the same transaction with different *TXID*s. Despite having the same set of inputs/outputs, these copies are considered as different transactions.

l: USE

As shown in § IX-B7, transaction malleability could be used to perform a form of double-spending. Moreover, as discussed in § IX-A3, malleability can be used for *Flooding* attacks.

m: IMPACT

Given an ECDSA signature, it is trivial to mathematically derive an equivalent one that uses different parameters [47].

n: MITIGATION

Transaction malleability was mitigated by the introduction of Segregated Witness (SegWit) [38], which moves signatures outside the transaction data, thus excluding them from the *TXID* hash. Nonetheless, the problem still exists for non-SegWit transactions.

B. MAN-IN-THE-MIDDLE ATTACKS

In a *Man in the Middle* (MitM) attack, $\mathcal{A}$ places herself between two connected devices and intercepts the messages they exchange. By doing so, $\mathcal{A}$ is able to drop and delay packets. Furthermore, if no encryption is used, $\mathcal{A}$ can inject, eavesdrop and tamper with messages.

o: USE

In Bitcoin, MitM attacks allow $\mathcal{A}$ to perform DoS (by dropping or delaying messages) and deanonymization (e.g., in the PERIMETER attack, § IX-C4).

Moreover, they enable powerful Auxiliary attacks, such as *Delay* (§ VIII-E), *Eclipse* (§ VIII-C), and *Partitioning* (§ VIII-D).

Finally, MitM attacks ease *Spoofing* attacks (§ VII-C).

p: IMPACT

Due to the lack of encryption and strong authentication, Bitcoin is highly susceptible to MitM [50]. Network-wide MitM attacks are also possible when $\mathcal{A}$ is at the AS level (§ VII-E1). Similarly, MitM attacks can target users connecting through Tor (§ VII-F2).

q: MITIGATION

Encryption in the Bitcoin protocol was discussed in 2016 [51]. However, to date, no countermeasure has been implemented.

C. SPOOFING ATTACKS

In a Spoofing attack [52], $\mathcal{A}$ sends messages on behalf of another entity. To do so, she sets the IP address in her packets to the address of another device (real or fake).

r: USE

In Bitcoin, spoofing can be used to perform node-level DoS attacks (e.g., in Tapsell's attack § IX-A1). Moreover, it can help an AS-level adversary to perform Eclipse attacks (see § VIII-C2). Finally, spoofing can be used for Tor DoS attacks (§ VII-F1).

s: IMPACT

Bitcoin relies on TCP sequence numbers [53] to protect from spoofing. However, this protection is not always effective. For instance, a MitM attacker can easily bypass it by intercepting and sniffing TCP packets.

D. DNS ATTACKS

As explained in § IV-A1.a, to join the network, nodes make use of *seed* DNS servers to discover other peers.

By compromising a seed, $\mathcal{A}$ could tamper with its DNS records, and replace legit addresses with others under her control. To that purpose, $\mathcal{A}$ can use known weaknesses of the DNS protocol, such as cache poisoning [54]. As a result, joining nodes would likely connect to $\mathcal{A}$'s nodes. MitM attacks on the connection with the DNS server could also achieve the same result.

t: USE

DNS attacks can enable $\mathcal{A}$ to perform *Eclipse* attacks (§ VIII-C) on new nodes.

u: IMPACT

Tapsell *et al.* [50] raised concerns about the vulnerability and centralization of Bitcoin seeds, which are very limited in number, and are mostly run by individual Bitcoin developers.

v: MITIGATION

During the bootstrap procedure, multiple DNS seeds are queried at once, to ensure a single one cannot eclipse a node.

Moreover, since 2020, some of the Bitcoin DNS seeds started enabling DNSSEC [55], which ensures data integrity and authentication.

E. AS-LEVEL ATTACKS

Being based on TCP, all Bitcoin connections go through ISP-level networks, or Autonomous System (AS), which route messages from the source to its destination. At this level, $\mathcal{A}$ can take advantage of her privileged position to inspect all TCP packets going through her AS, enabling possible MitM attacks. Moreover, $\mathcal{A}$ is able to re-route connections through her AS.

1) BGP HIJACK

In this attack, also known as *routing attack*, $\mathcal{A}_{AS}$ re-routes a TCP connection to make it go through her own AS.

The attack exploits a weakness in the BGP protocol [56], used by ASes to exchange routing information. In BGP, ASes advertise to other ASes the IP prefixes (e.g., 100.0.0.0/16) they can reach. In turn, advertisements from other ASes are used to build the local routing tables. Unfortunately, BGP routes are not verified, allowing malicious ASes to send rogue advertisements to divert traffic [57]. Therefore, $\mathcal{A}_{AS}$ can advertise a specific IP prefix to make connections towards it go through her AS.

a: USE

This attack can be used to intercept all Bitcoin traffic towards a specific destination, and perform MitM attacks (§ VII-E2). Moreover, it enables partitioning (§ VIII-D2) attacks.

b: IMPACT

Apostolaki *et al.* [58] state that more than 90% of Bitcoin nodes are potentially vulnerable to BGP hijacking attacks. Moreover, they show that large BGP hijacks often occur in real life, involving between 300 and 30,000 prefixes, and intercepting at least 100 Bitcoin nodes per month.

2) AS-LEVEL MitM

In this attack, $\mathcal{A}_{AS}$ performs a MitM attack on one or more Bitcoin connections going through her AS.

a: USE

This attack enables Delay (§ VIII-E4), and Deanonymization (§ IX-C4) attacks.

b: IMPACT

Due to its lack of geographical diversification, Bitcoin is particularly vulnerable to AS-level attacks. In fact, most nodes concentrate in just a few ASes [58], [59], making it easier for $\mathcal{A}$ to intercept a large share of connections. For instance, three of the major ASes together would be able to intercept more than 60% of all possible Bitcoin connections [58].

Thanks to the lack of encryption, and the use of a standard port (8333), $\mathcal{A}_{AS}$ can quickly identify all Bitcoin connections.

F. TOR ATTACKS

Tor is a popular anonymity network based on onion routing that allows clients to connect to a server without revealing their IP address. Each Tor connection goes through three relay nodes: *Guard*, to which the client connects, *Middle*, which acts as an intermediate relay, and *Exit*, which actually connects to the server. Therefore, a Tor connection is seen as coming from the Exit node. At each hop, the connection is encrypted with a pair of asymmetric keys, negotiated, during the setup of the connection, between the client and the corresponding Tor relay.

Bitcoin supports Tor to allow users to improve their anonymity. However, some studies revealed that combining the two protocols can create new attack vectors, such as DoS and even MitM, which we review in the following.

1) TOR DoS

In this attack, described by Biryukov *et al.* [60] in 2014, $\mathcal{A}$ prevents a public Bitcoin node from being reached via Tor. To do that, she exploits the Bitcoin reputation system (§ IV-A3.c), which, at the time of publication, banned misbehaving nodes for 24 hours.

a: USE

This attack could be used to facilitate deanonymization attacks against Tor nodes (see § IX-C2). Moreover, it could be used to perform the Tor MitM attack described in § VII-F2.

b: PROCEDURE

For the attack, $\mathcal{A}$ performs the following steps:

1) Connect N_A to $\mathcal{T}$ through as many Tor Exit nodes as possible.
2) For each Exit node N_E, send a malformed message causing a penalty score of 100. As a result, $\mathcal{T}$ disconnects N_E and bans it for 24 hours.

When all Tor Exit relays have been banned, $\mathcal{T}$ will be unreachable via Tor.

By repeating the procedure for all reachable nodes, $\mathcal{A}$ can completely isolate Tor from the Bitcoin network. When this occurs, all Bitcoin nodes are forced to connect directly (i.e., using their own address).

c: IMPACT

To successfully disconnect Tor nodes from a single target, $\mathcal{A}$ needs to establish as many connections as the current number of Tor Exit nodes, and to send a few kilobytes per connection.

At the time of publication, $\mathcal{A}$ could completely ban Tor nodes for 24 hours with approximately a million connections and less than 1 GByte of traffic (corresponding to around 1000 Bitcoin nodes and 1000 Tor Exit relays).

d: MITIGATION

Since 2018 (v0.18) [35], the 24-hour ban is not applied anymore, and misbehaving nodes are allowed to establish inbound connections even after being disconnected. As a result, this attack can currently get an Exit node disconnected but not banned.

2) TOR MitM

In this attack, described by Biryukov *et al.* in 2015 [61], $\mathcal{A}$ places herself in the middle between all Tor nodes and the rest of the Bitcoin network. To do that, she exploits the Tor Dos attack (§ VII-F1) to force Tor nodes to only use *Exit* relays or Bitcoin nodes under her control.

a: USE

This attack allows transaction deanonymization despite the target hiding its IP. In fact, by controlling all connections between Tor and the Bitcoin network, $\mathcal{A}$ can monitor all transactions sent by the target, and then use Fingerprinting (§ VIII-F) to recognize it when connected directly (i.e., without Tor) in a subsequent session.

Moreover, Tor-MitM can be used for Partitioning (§ VIII-D) and Eclipse (§ VIII-C) attacks.

b: PROCEDURE

For the attack, $\mathcal{A}$ performs the following steps:

1) Deploy multiple reachable Bitcoin nodes.
2) Frequently advertise $\mathcal{A}$'s nodes addresses, so that they are included into the maximum possible number of nodes address database (this step aims at increasing the chances of honest peers connecting to $\mathcal{A}$).
3) Deploy multiple medium-bandwidth Tor Exit relays; to filter traffic, $\mathcal{A}$ can allow only Bitcoin connections (i.e., port 8333) and only to specific IP addresses.
4) Use Tor DoS (§ VII-F1) to ban all honest nodes in $\mathcal{T}$ and all honest Exit relays; as a result, Tor nodes will only be able to connect to $\mathcal{A}$'s nodes or through $\mathcal{A}$'s Exit relays.[4]

[4] A third option would be to connect to Bitcoin nodes running as *Tor Hidden Services*, which accept connections through Onion addresses. However, the authors show how such nodes can also be inhibited (via DoS) or outnumbered by $\mathcal{A}$

c: MITIGATION

This attack has been mitigated by Tor DoS countermeasures (§ VII-F1).

VIII. AUXILIARY ATTACKS

In this section, we describe *Auxiliary attacks*, which are not harmful per se, but enable, ease, or improve other attacks. In other words, these attacks can be used by $\mathcal{A}$ as a preliminary or intermediate step to achieve her primary goals.

A. SYBIL ATTACKS

In a Sybil attack [62], $\mathcal{A}$ fakes the presence of different users in the network by deploying and controlling multiple nodes.

This attack can have two objectives: (1) to establish multiple (and diversified) connections to the same target, or (2) to increase the chances of a target connecting to $\mathcal{A}$ (the larger the fraction of Net^R controlled by $\mathcal{A}$, the higher the probability the target connects to her).

d: USE

Sybil attack can help achieve Double Spending (see § IX-B1) and Deanonymization (see § IX-C3).

Furthermore, the Sybil attack is a core component of virtually all other Auxiliary attacks, such as Topology Inference § VIII-B, Delay § VIII-E, Partitioning § VIII-D, and Eclipse § VIII-C.

In fact, Sybil attacks can be considered as an approximation of MitM attacks (§ VII-B), with $\mathcal{A}$ controlling a large fraction of a target's connections to monitor, drop, and delay messages.

e: THREAT MODEL

- $\mathcal{T}$=N, Net: the attack can target a single node or the entire network.
- $\mathcal{A}$=$\mathcal{A}(B) \triangleright \mathcal{T}$: the adversary uses multiple nodes, connected to $\mathcal{T}$.

f: PROCEDURE

$\mathcal{A}$ deploys multiple Bitcoin nodes, possibly varying their IP, prefix, and AS. Depending on their use, these nodes can be reachable or not. These nodes are then collectively controlled by $\mathcal{A}$ as a single entity. In particular, $\mathcal{A}$ coordinates these nodes and has a unified view of their status.

To increase the chances of a target node $N_{\mathcal{T}}$ connecting to $\mathcal{A}$'s nodes, Biryukov *et al.* [61] proposed the following strategies:

- Exhausting inbound connections: $\mathcal{A}$ fills up the inbound slots of all reachable nodes to prevent $N_{\mathcal{T}}$ from connecting to them (note that, at the time of publication, no eviction mechanism was implemented for inbound connections);
- Port poisoning: $\mathcal{A}$ floods $N_{\mathcal{T}}$ advertising legitimate Bitcoin addresses with a wrong port number. At the time of publication, entries in the address database were only distinguished by their IP; therefore if $\mathcal{A}$'s advertisement

of an IP reached $N_{\mathcal{T}}$ first, the legitimate one would never be stored. By reducing the number of legitimate addresses in the database, $\mathcal{A}$ increases the chances of $N_{\mathcal{T}}$ connecting to one of her nodes.

g: IMPACT

As any user can freely join the network, $\mathcal{A}$ has virtually no limit in the number of nodes she can deploy. For this reason, Bitcoin is especially vulnerable to Sybil attacks [63].

h: MITIGATION

As mentioned in § IV-A3.b, Bitcoin limits the number of outbound connections established by a single node towards addresses in the same IP range. This mitigates $\mathcal{A}$'s ability to monopolize a target's connections.

Moreover, as mentioned in § IV-A1, when inbound slots are exhausted, an eviction mechanism is used to allow new connections.

Finally, since 2021 [64], the address database distinguishes entries by the *IP*:*port* tuple instead of just the *IP*. This change effectively invalidates the port-poisoning technique.

B. TOPOLOGY INFERENCE ATTACKS

In the Bitcoin P2P network, knowledge of the topology can have important security implications [27], [63]. For this reason, the Bitcoin protocol hinders the ability to discover connections among nodes. Nevertheless, several works showed it is often possible to infer links by observing the propagation of messages, or by exploiting protocol-induced side channels.

Some of these works also raised the question of how topology obfuscation affects the ability to study the network [26], [27]. The authors of such works often consider topology inferring not as an attack to the network, but rather a way to uncover centralization problems and identify structural faults. In this respect, authors in [65] openly advocate for an open topology and propose a protocol to reliably discover connections among reachable nodes.

i: USE

Topology Inference can be used to achieve the following primary goals:

- Unfair Revenue: dishonest miners can use topology information to gain an advantage in the propagation of blocks [66], or even place corrupted nodes in the paths between honest miners [67];
- Double spending: knowing the neighbors of a target's node can facilitate 0-confirmation double spending [68], [69];
- Deanonymization: rumor-centrality heuristics (described in § IX-C1) depend on knowledge of the topology [70].

Moreover, topology inference can ease the following Auxiliary attacks:

- Partitioning (§ VIII-D): topology information can be used to identify the minimum vertex cut to split the network into two parts;
- Eclipse (§ VIII-C): if $\mathcal{A}$ knows the neighbors of a target, she can try to disrupt their connections (e.g. via DoS) to facilitate the eclipsing process.

1) BIRYUKOV *et al.*'s ATTACK (UNREACHABLE NODES)

This attack, described in 2014 by Biryukov *et al.* [60], allows identifying a subset of outbound peers of an unreachable node.

The attack is based on the fact that Bitcoin nodes advertise their own address upon establishing a new connection.

a: THREAT MODEL

- $\mathcal{T}=N^U$: the attack targets a single unreachable node.
- $\mathcal{A}=\mathcal{A}^O \triangleright Net^R$: the adversary uses a node $N_{\mathcal{A}}$ connected to all reachable nodes, and observes addresses.

b: PROCEDURE

When $\mathcal{T}$ connects to a node N_E, it sends an ADDR message containing its own address $a_{\mathcal{T}}$; in turn, N_E forwards $a_{\mathcal{T}}$ to $N_{\mathcal{A}}$ with a certain probability (which depends on how many connections $N_{\mathcal{A}}$ maintains with N_E).

To infer a subset of the entry nodes of $\mathcal{T}$, $\mathcal{A}$ simply tracks which nodes advertise $\mathcal{T}$'s address $a_{\mathcal{T}}$.

To improve effectiveness, $\mathcal{A}$ can make use of Sybil nodes.

c: IMPACT

The success rate of the attack depends on the number of connections between $N_{\mathcal{A}}$ and nodes in Net^R.

2) BIRYUKOV *et al.*'s ATTACK (REACHABLE NODES)

This attack, described by Biryukov *et al.* in 2014 [60], allows $\mathcal{A}$ to determine if two reachable nodes are connected. This method is based on the advertisement of marker addresses, and fine-tuned through the choice of appropriate timestamps.

a: THREAT MODEL

- $\mathcal{T}=N^R$: the attack targets a single reachable node, and can be extended to the whole reachable network (Net^R).
- $\mathcal{A}=\mathcal{A}^{I/O}(\mathcal{T}) \triangleright \mathcal{T}$: the adversary connects to $\mathcal{T}$, of which she knows the IP; for the attack, she injects and observes addresses.

b: PROCEDURE

The attack allows determining whether $\mathcal{T}$ is connected to a peer N_P, and it consists of two phases.

In the first phase, $\mathcal{A}$ performs the following steps:

1) Send $\mathcal{T}$ a set of fake *marker* addresses (i.e., addresses with no running clients), in 10-entry ADDR messages (this ensures the messages are forwarded).
2) Monitor markers received from other peers.
3) Estimate a value k like follows: as the number of markers increases, the amount received by $\mathcal{A}$ converges to

$\frac{2}{1+k}$, if $\mathcal{T}$ considers the markers belong to the same network, or $\frac{1}{1+k}$, otherwise.

To increase accuracy, $\mathcal{A}$ can run multiple listening nodes and repeat the experiments several times.

In the second phase, $\mathcal{A}$ performs the following steps:

1) Use k to compute the number of markers to forward to $\mathcal{T}$'s peers.
2) Choose a set of markers and send them to $\mathcal{T}$ in 10-entry `ADDR` messages.
3) Send `GETADDR` messages to N_P and observe replies: if the number of markers received corresponds to k, mark P as a peer of T.

To avoid false positives, $\mathcal{A}$ sets the markers timestamps to values close to 10 minutes in the past. Since, at the time of publication, addresses older than 10 minutes were not forwarded, the markers would be unlikely to travel through more than few nodes.

The procedure can be applied to all reachable nodes to infer the topology of the whole public network.

c: IMPACT

Experimental results showed this method could verify a connection with very high accuracy. Moreover, thanks to the timestamp-based improvement, all experiments showed zero false positives.

3) AddressProbe

This attack, proposed by Miller *et al.* in 2015 [26], allowed inferring connections among reachable nodes by using timestamps in `ADDR` messages.

At the time of publication, addresses in the database were updated like follows: for outbound connections, the timestamp updates each time a message is received from the peer; for inbound connections, the timestamp is set to when the connection is created, but it is not updated afterward; for all the other addresses, the node uses the original timestamp plus a 2-hour penalty. Based on this, it was possible to determine outbound peers by looking at timestamps in `ADDR` messages.

a: THREAT MODEL

- $\mathcal{T}$=Net^R: the attack targets all reachable nodes.
- $\mathcal{A}$=$\mathcal{A}^O \triangleright Net^R$: the adversary connects to all reachable nodes, and observes addresses.

b: PROCEDURE

$\mathcal{A}$ performs the following steps:

1) Send a `GETADDR` message to each node in $\mathcal{T}$.
2) Collect returning `ADDR` messages.
3) Analyze the timestamps of received addresses: for each pair of nodes (N_1,N_2), if N_1 reports N_2 with a timestamp t of less than 2 hours, and t is unique (i.e., no other node reports the same t for N_2), then infer the link $N_1 \rightarrow N_2$.

c: IMPACT

Experimental results showed that AddressProbe had very high accuracy (with an average of 95% confidence), with a low number of false negatives and near-zero false-positive rates.

d: MITIGATION

Since 2015 (v0.11) [69], [71], address timestamps are not updated when receiving a message from a connected peer. This effectively invalidates this attack.

Moreover, in 2016 (v0.14) [72], the 2-hour penalty was removed for addresses learned from self-advertising peers.

4) NEUDECKER *et al.*'s ATTACK

This attack, proposed by Neudecker *et al.* in 2016 [73], allowed to infer the reachable topology based on the *rumor centrality* of gossip propagation: if $\mathcal{A}$ is connected to all nodes, and knows the source of a data message, she can infer connections by observing the time at which nodes propagate the data. Intuitively, the reception time for each message correlates to the network topology.

a: THREAT MODEL

- $\mathcal{T}$=Net^R: the attack targets all reachable nodes.
- $\mathcal{A}$=$\mathcal{A}^{I/O}(B) \triangleright Net^R$: the adversary controls multiple nodes connected to all reachable nodes; for the attack, she observes and possibly injects transactions.

b: PROCEDURE

To perform the attack, $\mathcal{A}$ must know the origin of a message. This can be done via Deanonymization attacks (§ IX-C), or, in a simpler scenario, $\mathcal{A}$ can create transactions and send them to a single node.

For the attack, $\mathcal{A}$ performs the following steps:

1) Observe incoming `INV` messages from each connected peer.
2) For each transaction, log a tuple (*receptiontime*, *sendingpeer*) for each peer advertising it.
3) Given the reception time, subtract the latency between the observing node and the sender to get an estimate of the *sending time*.
4) Convert sending times in time differences relative to the creation time: each time difference is a sample delay between the transaction source and the peer that forwarded it.
5) For each pair of peers, group all time differences of all messages created by such peers.
6) Estimate the length of the shortest path between each pair of nodes. To that purpose, use a likelihood function of the probability that a given path length produces a certain delay. The function is used to decide which shortest path length is the most likely, based on the observed delay.

c: IMPACT
In a theoretical framework, with a random-graph topology and no delays in the relay of messages, this method has recall close to 100% and precision over 90%. However, in the Bitcoin network, these values are both around 40%, due to the presence of highly-connected nodes [26], and the use of *trickling* (§ IV-B1.a).

While lower than the theoretically achievable, this value is still much better then simply guessing the connections of a peer.

d: MITIGATION
Their technique was made ineffective by the switch from Trickle to Diffusion [37].

5) GRUNDMANN *et al.*'s TRANSACTION-ACCUMULATION Attack)
In this attack, proposed by Grundmann *et al.* in 2018 [74], $\mathcal{A}$ exploits the fact that nodes accumulate transactions before announcing them to their peers. In particular, an `INV` message contains all transactions received since the last `INV` message was sent.

a: THREAT MODEL
- $\mathcal{T}$=Net^R: the attack targets all reachable nodes.
- $\mathcal{A}$=$\mathcal{A}^{I/O}(B) \triangleright Net^R$: the adversary controls multiple nodes connected to $\mathcal{T}$; for the attack she injects and observes transactions.

b: PROCEDURE
$\mathcal{A}$ performs the following steps:

1) Create a transaction $tx_I \in \mathbb{S}$ for each peer N_I.
2) Send all transactions to their peer at once.
3) Monitor the first `INV` messages received.
4) Infer connections following these rules:
    - If the first `INV` message received from peer N_X contains only tx_Y and no other transaction from $\mathbb{S}$, then N_X and N_Y are connected.
    - If the first `INV` message received from peer N_X contains more than one transaction from $\mathbb{S}$, at least one of the peers associated with the announced transactions is connected to N_X.

c: IMPACT
Experimental results showed, after 50 runs, a precision of 67% but a recall of only 10%. According to the authors, this attack is hardly practical in real life.

6) GRUNDMANN *et al.*'s DOUBLE-SPEND ATTACK
This attack, also proposed by Grundmann *et al.* in 2018 [74], exploits the fact that nodes do not relay double-spending transactions.

a: THREAT MODEL
- $\mathcal{T}$=N^R: the attack targets a single reachable node, and can be extended to all reachable nodes (Net^R).
- $\mathcal{A}$=$\mathcal{A}^{I/O}(\mathcal{T}) \triangleright Net^R$: the adversary connects to $\mathcal{T}$, of which she knows the IP; for the attack, she injects and observes transactions.

b: PROCEDURE
$\mathcal{A}$ performs the following steps:

1) Create a transaction tx_I for each peer N_I, except for $\mathcal{T}$, such that all transactions have the same input but different outputs (i.e., they are double spends).
2) Send all transaction to their peers at the same time.
3) Monitor incoming `INV` messages: if tx_X is received, mark N_X as peer of $\mathcal{T}$.

To infer multiple links, the procedure is repeated several times.

To reduce the number of false positives, $\mathcal{A}$ can repeat the procedure. For instance, a connection can be considered valid after the number of transactions confirming it reaches a certain threshold.

To avoid inferring the same peer twice (say N_B), $\mathcal{A}$ makes use of two different outputs o_1 and o_2, and modifies the procedure as follows:

- A transaction t_I spending o_1 is sent to all peers N_I, except $\mathcal{T}$ and N_B;
- A transaction t_T spending o_2 is sent to $\mathcal{T}$;
- A transaction t_B spending o_1 and o_2 is sent to N_B.

c: IMPACT
This technique has a very high precision (97%), which slowly decreases at each new run (because valid links have already been detected). On the other hand, recall depends on the portion of nodes the monitoring node N_M is connected to. In particular, when connected to all reachable nodes, the recall reaches 95% after 100 runs. Implementing all improvements results in a recall of 96% after 25 runs, with a precision of 94% when N_M is connected to all peers.

Experimental results showed, after 50 runs, 60% recall and 97% precision.

7) TxProbe
This technique, proposed by Delgado *et al.* in 2018 [27], exploits double-spending and orphan transactions.

In particular, it leverages the following fact: when a node receives an `INV` message advertising a transaction previously stored as orphan, it omits such transaction from subsequent `GETDATA` messages.

a: THREAT MODEL
- $\mathcal{T}$=Net^R: the attack targets reachable nodes.
- $\mathcal{A}$=$\mathcal{A}^I \triangleright Net^R$: the adversary connects to all reachable nodes; for the attack, she injects and observes transactions.

b: PROCEDURE

The attack allows inferring links between a *source set*, containing i nodes, and a *sink* set.

For the attack, $\mathcal{A}$ performs the following steps:

1) Create $i + 1$ double-spending transactions, of which, i are tagged as *parent* ($tx_P i$), and one as *flood* (tx_F).
2) For each parent $tx_P i$, create a *marker* transaction $tx_M i$ spending outputs from $tx_P i$.
3) Send tx_F to all sink nodes.
4) After a few seconds, send each $tx_P i$ to the corresponding node.
5) After a few seconds, send each $tx_M i$ to the corresponding node.
6) Ensure tx_F remains within the *sink* set, and each $tx_P i$ remains in the node N_i by using *InvBlock* (§ VIII-E2).
7) After a few seconds, send to every sink node an `INV` message containing all $tx_M i$: these nodes will request back only unknown markers.
8) Infer links by mapping markers that have not been sent back to the corresponding node N_i.

To infer the whole topology, the procedure is run multiple times, permuting the sets so that every pair of nodes have been in a different set at least once.

To ensure all markers are stored by sink nodes, $\mathcal{A}$ initially performs a *cleansing* procedure:

1) Create a *cleanser* transaction, and, spending from it, 100 distinct double-spending *squatter* transactions.
2) Send all squatters to every sink node: this aims at filling the orphan pool.
3) Send all sink nodes the cleanser: this empties the pool.

To avoid markers from being evicted from the orphan pool, $\mathcal{A}$ creates hard-to-evict transactions by exploiting a flaw in the eviction mechanism.

c: IMPACT

Experimental results showed TxProbe had a precision of 100% and a recall above 93%.

Nevertheless, as the size of the source set is bound by the size of the orphan pool, this technique required many rounds to infer the full topology. In particular, inferring the topology of the reachable network (i.e., 10.000 nodes) required more than 8 hours.

Moreover, this technique is rather invasive and can interfere with the ordinary transaction processing.

d: MITIGATION

In v0.18 [75], the design flaw in the eviction mechanism of the orphan pool was fixed. Moreover, the *InvBlock* technique was also mitigated by [76] and [77].

C. ECLIPSE ATTACKS

In an Eclipse attack [78], $\mathcal{A}$ aims at controlling all the connections of a target node to control its communications with the network. With this attack, data from the network can be completely hidden from the target (hence the name *eclipse*).

e: USE

Eclipse attacks can be used for the following primary goals:

- Double Spending: by eclipsing a merchant's node, $\mathcal{A}$ can easily conceal a double-spending transaction $tx_{\mathcal{A}}$ sent to the network; by additionally eclipsing a fraction of miners, $\mathcal{A}$ can even launch n-confirmation double-spending attacks;
- Unfair Revenue: by eclipsing miners, $\mathcal{A}$ can increase the portion of miners working on her block during a block race, making Selfish Mining (§ IX-D1) easier, and even increasing profits [67];
- Deanonymization: by eclipsing a node, $\mathcal{A}$ can detect all the transactions it generates;
- DoS: when a target is eclipsed, $\mathcal{A}$ can prevent transactions and blocks from reaching, and leaving, the node.

Eclipse-enabled DoS attacks can also be seen as a form of Delay attacks (§ VIII-E). Furthermore, by eclipsing carefully-chosen nodes (e.g., gateways of popular mining pools), it is possible to partition the network (see § VIII-D).

1) HEILMAN *et al.*'s ATTACK

This attack, described in 2015 by Heilman *et al.* [79], exploits the eviction mechanism of the address database to monopolize all the target's connections.

a: THREAT MODEL

- $\mathcal{T}=N^R$: the attack targets a single reachable node.
- $\mathcal{A}=\mathcal{A}^I(B, \mathcal{T}) \triangleright \mathcal{T}$ controls multiple nodes with different IP prefixes, connected to $\mathcal{T}$, and injects addresses.

b: PROCEDURE

To eclipse $\mathcal{T}$, $\mathcal{A}$ aims at filling its address database with her addresses, so that, when the node restarts, it only connects to $\mathcal{A}$'s nodes.

To do so, $\mathcal{A}$ follows these steps:

1) To fill $\mathcal{T}$'s `tried` table, connect to $\mathcal{T}$ from all Sybil nodes: this causes their addresses to be stored in the table, likely overwriting legitimate entries due to the freshness-based *eviction* mechanism.
2) To fill the `new` table, send unsolicited `ADDR` messages containing bogus addresses until they overwrite all legitimate addresses in the table.
3) Wait for $\mathcal{T}$ to restart (this can be forced with a DoS attack). when this occurs, $\mathcal{T}$ chooses outbound peers from the two tables; as the tables are filled with adversarial and bogus addresses, all new connections will be towards $\mathcal{A}$'s nodes with high probability.
4) Finally, occupy all $\mathcal{T}$'s inbound slots to completely isolate it.

c: IMPACT

A successful attack required no more than 6000 addresses (in different groups), which can be easily achieved by using a botnet. Moreover, real-world experiments proved that most attacks had a success rate higher than the predicted one. For instance, a botnet with 4600 addresses had a 100% success rate even in the worst-case scenario. Notably, a small botnet of 400 bots also succeeded with more than 80% probability.

Nevertheless, if the victim stores enough legitimate addresses, it cannot be eclipsed regardless of the number of IPs controlled by the adversary.

d: MITIGATION

Since v0.10.1 [80], a number of countermeasures are included in Bitcoin. First, the placement mechanism in the database is deterministic, based on the hash of the address. This prevents $\mathcal{A}$ from storing an address by repeatedly trying it. Second, outbound peers are selected uniformly at random from the two tables. This eliminates the $\mathcal{A}$'s advantage due to the freshness of her addresses. Finally, the number of buckets in the tables was increased, which increased the number of addresses needed for the attack.

In v0.12 [33], an eviction mechanism for inbound peers was included to always allow new connections. This mechanism takes various factors into account in such a way that it is hard for an adversary to systematically avoid eviction.

In v0.14 [81], *feeler connections* were introduced (see § IV-A1.b). This helps clean the `new` table from useless addresses and increase valid entries in the `tried` table.

Finally, in v0.21 [32], *anchor* nodes were implemented (see § IV-A1), which make eclipsing much harder.

Additionally, it is worth noting that miners, merchants, and online wallets typically protect themselves from Eclipse attacks by disabling incoming connections and choosing trusted well-known nodes as outbound peers [79].

2) EREBUS: AS-LEVEL ECLIPSING

In the *EREBUS* attack, proposed by Tran *et al.* in 2020 [82], $\mathcal{A}$ exploits AS-level privileges to control a vast number of network addresses reliably over an extended period of time.

a: THREAT MODEL

- $\mathcal{T}=N^R$: the attack targets a single reachable node.
- $\mathcal{A}=\mathcal{A}^I_{AS}(\mathcal{T}) \triangleright \mathcal{T}$ controls an Autonomous System $AS_{\mathcal{A}}$, connects to $\mathcal{T}$, of which she knows the IP, and injects addresses.

b: PROCEDURE

The objective of the procedure is to force $\mathcal{T}$ to connect to other (honest) nodes so that all its connections traverse $AS_{\mathcal{A}}$. In this condition, $\mathcal{A}$ can perform MitM attacks against $\mathcal{T}$.

$\mathcal{A}$ use *shadow* IP addresses, that is, addresses whose route from $\mathcal{T}$ traverses $AS_{\mathcal{A}}$. Therefore, any attempt of $\mathcal{T}$ to connect to a shadow IP is intercepted by $\mathcal{A}$.

The attack is divided in two phases. In phase 1, $\mathcal{A}$ determines the shadow IPs for $\mathcal{T}$. To do so, she follows these steps:

1) Enumerate all the ASes whose route from $\mathcal{T}$ would traverse $AS_{\mathcal{A}}$.
2) Enumerate all the available IP addresses in the selected ASes and tag them as *shadow IPs*.
3) Test whether the packets from $\mathcal{T}$ to the shadow IPs actually traverse $AS_{\mathcal{A}}$.

In phase 2, $\mathcal{A}$ creates the connections between $\mathcal{T}$ and the shadow peers, until $\mathcal{T}$ is only connected to shadow IPs. To do so, she follows these steps:

1) During several weeks, connect to $\mathcal{T}$ using (spoofed) shadow IPs: this slowly fills up $\mathcal{T}$'s address database.
2) Flood $\mathcal{T}$ with a large number of `ADDR` messages, containing the shadow IPs, until the `new` table is filled.
3) Wait until enough shadow addresses in `new` are moved to `tried`.
4) Trigger a reboot of $\mathcal{T}$ (e.g., with a DoS attack).

Upon rebooting, $\mathcal{T}$ connects all outgoing connections to shadow IPs with high probability, thus eclipsing the node.

To reduce the duration of the attack, $\mathcal{A}$ keeps track of $\mathcal{T}$'s outgoing connections going through $AS_{\mathcal{A}}$, and triggers a reboot whenever she believes it would be beneficial.

c: IMPACT

EREBUS is particularly dangerous since it is hard to detect and leaves no traces. Authors in [82] show that Tier-1 or large Tier-2 ISPs can readily target the vast majority of the reachable network.

According to their experiments, $\mathcal{A}$ can easily fill up the `new` table in 30 days, and control a large portion of the `tried` table after 40 days. In particular, the success probability mostly depends on the duration of the attack and the *age* of the victim. For example, *young* nodes (up to 30-day old) proved to be more vulnerable to the attack, with a success rate of around 30% after 50 days of attack. Nevertheless, a 50-day attack against *older* nodes (40- to 50-day old) has still around a 20% probability of success. Notably, AS ranking does not seem to influence the results.

The EREBUS attack has also a very low bandwidth cost for $\mathcal{A}$ (around 520 bit/s), which makes it highly scalable to several nodes.

d: MITIGATION

This attack was made harder by the addition of block-relay-only peers [31] and their anchoring [32], described in § IV-A1.

Moreover, in v0.20, AS-aware peering was also implemented [83], which makes the attack even harder.

D. PARTITIONING ATTACKS

In a Partitioning attack, $\mathcal{A}$ tries to isolate a portion of nodes from the rest of the network. In other words, the attack aims at splitting the network in two parts that cannot communicate with each other (hence the name *partitioning*). In this respect, the attack can be seen as an Eclipse attack against multiple nodes.

e: USE

Partitioning attacks can be used for:

- Double Spending: by partitioning miners, $\mathcal{A}$ can fork the blockchain, and try to trick a victim into accepting a double-spending transaction (an example is shown in § IX-B5);
- Unfair Revenue: chain forking can be also used to split the mining power, thus improving Selfish Mining attacks (§ IX-D1);
- DoS: a partitioning attack prevents the isolated nodes from communicating with the rest of the network.

1) NEUDECKER *et al.*'s ATTACK

In this attack, described by Neudecker *et al.* in 2015 [84], $\mathcal{A}$ uses a powerful botnet to split the network in two parts.

The attack targets the *minimum vertex cut* of the reachable network's graph, which is the smallest set of nodes whose removal causes the network to split.

a: THREAT MODEL

- $\mathcal{T}$=Net^R: the attack targets the reachable network.
- $\mathcal{A}$=$\mathcal{A}^I(B, Net^R)$: the adversary controls a large botnet with DoS capabilities, has knowledge of the network topology, and injects addresses.

b: PROCEDURE

To split the network, $\mathcal{A}$ follows these steps:

1) Deploy as many Sybil nodes as possible, which only advertise $\mathcal{A}$'s addresses: this helps reduce connections between honest nodes, thus making the attack easier.
2) Calculate the minimum vertex cut of the reachable network.
3) Make Sybil nodes stop forwarding transactions and blocks, and, at the same time, perform a DDoS attack against nodes in the minimum vertex cut of the reachable network.

The removal of the nodes in the cut should create two disconnected portions in the network.

c: IMPACT

Simulation results show the Bitcoin network could resist an attack during several hours from a botnet as large as the network itself.

The cost of the attack depends on the size of the minimum vertex cut.

d: MITIGATION

As it requires knowledge of the topology, this attack is mainly mitigated by preventing Topology Inference attacks (§ VIII-B).

For instance, the addition of *block-relay-only* connections [31] was explicitly implemented to make partitioning harder. In fact, by not relaying transactions and addresses, these connections are harder to infer by an adversary.

Additionally, miners can protect from partitioning by using relay networks.

2) APOSTOLAKI *et al.*'s ATTACK: AS-LEVEL PARTITIONING

This attack, described by Apostolaki *et al.* in 2017 [58], makes use of BGP hijacking to disconnect a portion of the network.

a: THREAT MODEL

- $\mathcal{T}$=Net^P: the attack targets a portion of the network, independently from its reachability.
- $\mathcal{A}$=$\mathcal{A}_{AS}(\mathcal{T})$ controls (at least) an AS $AS_{\mathcal{A}}$, knows the IP addresses of nodes in $\mathcal{T}$, and drops messages.

b: PROCEDURE

To partition the network, $\mathcal{A}$ follows these steps:

1) Use BGP hijacking (§ VII-E1) to divert the traffic directed to nodes in $\mathcal{T}$.
2) Intercept all Bitcoin traffic by filtering TCP/IP addresses and ports.
3) Check each packet to determine if the connection crosses the partition (i.e., it connects a node in $\mathcal{T}$ with a node not in $\mathcal{T}$); if so, drop the packet.
4) At the same time, monitor contents to detect *leakage points*, that is, nodes in $\mathcal{T}$ maintaining connections with nodes outside $\mathcal{T}$ that $\mathcal{A}$ cannot intercept; if such a node is detected, isolate it from the other nodes in $\mathcal{T}$.

Eventually, $\mathcal{A}$ isolates the maximum number of nodes in $\mathcal{T}$ that can be isolated.

c: IMPACT

This attack could isolate up to 47% of the mining power by hijacking less than 100 prefixes. Moreover, the authors show that it would take less than 90 seconds to re-route all $\mathcal{T}$'s traffic through $AS_{\mathcal{A}}$ once a hijack is initiated.

On the other hand, Bitcoin proved to quickly recover from partitioning once the attack is over.

d: MITIGATION

Since v0.20 [83], nodes diversify their connections based on their BGP route, when provided.

E. DELAY ATTACKS

In a Delay attack, $\mathcal{A}$ aims at preventing a target from receiving a transaction or block for a certain amount of time, without disrupting its connection.

e: USE

Delay attacks can be used to achieve the following goals:

- Unfair Revenue: when targeting a miner, a delay in the block reception translates to an equal amount of wasted mining power; as shown in § IX-D1, this could sensibly lower the difficulty of Selfish Mining;
- Double Spending: as shown in § IX-B4 and § IX-B5, Delay attacks can be used to withhold information from a victim, and induce him to accept double-spending payments.

- Denial of Service: $\mathcal{A}$ can prevent the propagation of information to the entire network by simply connecting to all reachable nodes and denying them the delivery of transactions and blocks [85];

Additionally, transaction-delay attacks enable so-called *node-coloring* techniques, which can be used for Topology Inference [27].

1) TIMEJACKING

In this attack, proposed by Alex Boverman in 2011 [86], $\mathcal{A}$ isolates a target $\mathcal{T}$ by increasing the gap between its network time and that of the majority of miners.

This can be seen as a block-delay attack, where $\mathcal{A}$ prevents the $\mathcal{T}$ from updating its blockchain for a certain amount of time.

Bitcoin nodes maintain an internal clock called *network time*, which is based on the time reported by other peers (in `VERSION` messages). This clock is allowed to differ up to 70 minutes from the system time (beyond this threshold, the clock is reset). Network time is used to validate new blocks, which are only accepted if they are less than 2 hours ahead (this is sometimes known as the *2-hour rule*). In this attack, $\mathcal{A}$ tries to modify the network time by reporting inaccurate times.

a: THREAT MODEL

- $\mathcal{T}=N^R$: the attack targets a single reachable node.
- $\mathcal{A}=\mathcal{A}^I(M, B, \mathcal{T}) \triangleright \mathcal{T}$: the adversary is a miner, controlling multiple nodes connected to $\mathcal{T}$ and reporting wrong times, and injects blocks.

b: PROCEDURE

$\mathcal{A}$ performs the following steps:

1) connect nodes to $\mathcal{T}$ reporting inaccurate times to move its network time 70 minutes in the past;
2) connect nodes to a majority of miners and move their network time 70 minutes in the future: this results in a total difference of 140 minutes between $\mathcal{T}$ and the miners;
3) create a new block $B_{\mathcal{A}}$ with timestamp 190 minutes in the future: while miners will accept this block (as less than 120 minutes in the future), $\mathcal{T}$ will reject it (as 260 minutes in the future);

At the end of the procedure, $\mathcal{T}$ is isolated from the main chain, since each new block appended to B will be rejected as invalid.

The attack can continue until either a block is created by an unaffected miner, or $\mathcal{T}$ resets its clock.

c: IMPACT

A successful Timejacking attack isolates the target for at least 140 minutes (70 minutes if only the target was affected).

d: MITIGATION

While this threat was discussed in the past [87], no definitive fix has been implemented to date.

However, since 2016 (v0.13) [88], users can individually choose a maximum time offset (different from the default 70 minutes) between local and network time. Additionally, a warning sign is shown to the user whenever the time offset is too high.

Moreover, since 2021 [89], nodes only consider `VERSION` timestamps from outbound peers.

2) InvBlock (TRANSACTIONS)

This attack, described by Miller *et al.* in 2015 [26], can be used to prevent a node from receiving a specific transaction for an arbitrary amount of time, and, by extension, to delay the propagation of such a transaction through the network.

The attack exploits the timeout set by nodes after requesting a transaction with `GETDATA` (see § IV-B1). At the time of publication, nodes waited 2 minutes after sending a `GETDATA(`tx`)` message, before requesting tx to another peer. Moreover, the queue of peers from which to request each tx was ordered by the time each `INV(`tx`)` message was received, and allowed the same peer to advertise the same tx multiple times. This enabled $\mathcal{A}$ to delay the delivery of transactions for an indefinite amount of time.

a: THREAT MODEL

- $\mathcal{T}=N$: the attack targets a single node.
- $\mathcal{A}=\mathcal{A}^I \triangleright \mathcal{T}$: the adversary is connected to $\mathcal{T}$, and injects `INV` messages.

b: PROCEDURE

To delay the delivery of a transaction tx to $\mathcal{T}$, $\mathcal{A}$ follows these steps:

1) Send an `INV` message to $\mathcal{T}$ containing tx
2) When receiving the `GETDATA(`tx`)` message from $\mathcal{T}$, do not reply.

This procedure prevents $\mathcal{T}$ from receiving tx for 2 minutes.

To delay the reception of tx for τ minutes, $\mathcal{A}$ sends τ/t *INV* messages (possibly from different nodes) advertising tx, without replying to the corresponding `GETDATA` requests. As a consequence, $\mathcal{T}$ will wait τ minutes without receiving tx.

c: MITIGATION

Since v0.12, `INV` messages are filtered per IP address [90]. This prevents a single node to advertise the same transaction multiple times.

In v0.19, *InvBlock* was invalidated by randomizing the list of peers from which a transaction is requested [76]. At the same time, this change reduced the `GETDATA` timeout to 1 minute.

This measure was further enforced in v0.21 [77], by making outbound peers always preferred over inbound ones, when requesting transaction data.

3) InvBlock (BLOCKS)

This attack, described in 2015 by Gervais *et al.* [85], expanded on *InvBlock* to perform block-delay attacks.

At the time of publication, block relay was akin to transaction relay, with a fixed 20-minute request timeout, and an announcement-time-based requesting queue.

a: THREAT MODEL

- $\mathcal{T}{=}N$: the attack targets a single node.
- $\mathcal{A}{=}\mathcal{A}^I{\triangleright}\mathcal{T}$: the adversary uses a node $N_\mathcal{A}$ connected to $\mathcal{T}$, and injects `INV` messages.

b: PROCEDURE

To delay the delivery of a block to $\mathcal{T}$, $\mathcal{A}$ follows these steps:

1) If $\mathcal{T}$ is reachable, fill up all available connection slots: this prevents new connections from triggering a blockchain synchronization (§ IV-B2.a).
2) To increase the probability of receiving blocks before $\mathcal{T}$, connect $N_\mathcal{A}$ to multiple nodes.
3) To increase the probability of being the first peer to announce a block to $\mathcal{T}$ (which is necessary for the attack), skip block verification.
4) When receiving a `GETDATA` message for a block from $\mathcal{T}$, do not reply.

c: IMPACT

This attack could prevent new blocks from being received by $\mathcal{T}$ until another peer advertises a block before her.

d: MITIGATION

Today, the Bitcoin protocol includes a number of measures to mitigate this attack.

Since v0.12 [40], [41], nodes have the ability of advertising blocks directly with a `HEADERS` message, without first announcing them with `INV`. Additionally, the block-download timeout was reduced to 10 minutes [91].

Finally, the implementation of Compact Block Relay protocol [42] (described in § IV-B2.c), and the use of high-bandwidth (HB) nodes, likely made this attack infeasible.

Additionally, miners can protect themselves by connecting through relay network.

4) APOSTOLAKI *et al.*'s ATTACK: AS-LEVEL DELAYING

This attack, described in 2017 by Apostolaki *et al.* [58], exploited the `INV`-based block-relay protocol, and the corresponding download timeout.

a: THREAT MODEL

- $\mathcal{T}{=}N$: the attack targets a generic node.
- $\mathcal{A}{=}\mathcal{A}^I_{AS}(\mathcal{T})$ controls one or more ASes and knows $\mathcal{T}$'s IP; for the attack, she tampers with messages.

b: PROCEDURE

To delay block delivery over a single connection, $\mathcal{A}$ follows these steps:

1) Intercept at least one direction of $\mathcal{T}$'s connection, using, if necessary, BGP hijacking (§ VII-E1).
2) If traffic from $\mathcal{T}$ is intercepted, corrupt block-requesting `GETDATA` messages, so that the requested block is not sent; to have the message accepted by the recipient and keep the connection alive, preserve the message length and structure, and update the TCP and Bitcoin checksums.
3) If traffic towards $\mathcal{T}$ is intercepted, corrupt `BLOCK` messages, so that the block is considered invalid; while discarding the block, $\mathcal{T}$ will not request the block until the timeout expires.

In both cases, the corresponding block is not received by $\mathcal{T}$ for the duration of the timeout (20 minutes, at the time of publication).

c: IMPACT

The effectiveness of the attack depends on the direction and fraction of the $\mathcal{T}$'s traffic that $\mathcal{A}$ intercepts. Specifically, the probability that $\mathcal{A}$ intercepts the connection with the peer to which $\mathcal{T}$ requests a block increases proportionally with the fraction of the connections she diverts.

Furthermore, since $\mathcal{A}$ is not connected to $\mathcal{T}$, there is no risk of being disconnected due to the download timeout, which makes the attack last longer. The authors show that intercepting 50% of a node's connections can leave it uninformed of new blocks 63% of its uptime.

On the other hand, network-wide attacks are unlikely to happen in practice. As most pools use gateways in multiple ASes, only very powerful coalitions of network attackers (e.g., all US-based ASes) could perform a network-wide Delay attack. In fact, even using a small number of ASes is enough to protect mining pools.

d: MITIGATION

This attack is mitigated by the countermeasures described in § VIII-E3.

5) TendrilStaller

This attack, described in 2019 by Walck *et al.* [92], is a block-delay attack that exploits the use of High-Bandwidth (HB) nodes.

As described in § IV-B2.c, nodes select up to 3 High-Bandwidth (HB) among their peers to receive new blocks fast. These nodes are allowed to send blocks unsolicited, that is, without previously advertising them. HB peers are selected according to the *most-recent-sent-block* rule: if a neighbor sends a new valid block, it is added to the HB Neighbor (HBN) list; if 3 HB neighbors already exist, the one that sent a block less recently is replaced.

Nodes also accept an unsolicited compact block from a non-HB peer if it is unknown and all transactions in it are known. If the reconstructed block is valid, the sender is added to the HBN list.

a: THREAT MODEL

- $\mathcal{T}=N^R$: the attack targets a single reachable node.
- $\mathcal{A}=\mathcal{A}(B) \triangleright \mathcal{T}$ makes use of three nodes connected to $\mathcal{T}$.

b: PROCEDURE

The attack proceeds in two phases.
In Phase 1, $\mathcal{A}$'s nodes follow these steps:

1) Send valid compact blocks until a `SENDCMPCT (true)` message is received (i.e., the node is added to the HBN list).
2) When all nodes are selected as HB neighbors, start Phase 2.

To increase the chances of being the first to send a new block to $\mathcal{T}$, $\mathcal{A}$'s nodes establish more connections, instruct all peers to send unsolicited compact blocks, and skip block verification.
In Phase 2, $\mathcal{A}$'s nodes follow these steps:

1) Stop relaying compact blocks and start advertising them via `HEADERS`.
2) When a `GETDATA` is received for a block, delay the delivery for the current timeout (i.e., 10 minutes).

$\mathcal{A}$ can keep delaying new blocks as long as the three nodes are in the HBN list and they are the first to announce them. Whenever an attacking node is evicted from the HBN list (i.e., it receives a `SENDCMPCT(false)` message), the three nodes switch back to Phase 1.

To increase the chances of a successful attack, $\mathcal{A}$ can use proxy nodes relaying blocks to the attacking nodes as fast as possible. the authors propose the use of two colluding lightweight nodes.

c: IMPACT

Experiments show that the TendrilStaller attack is particularly effective when the attacking nodes are geographically close to the victim. In particular, when the round-trip time between the attacker and the victim is less than 80ms, the success rate is between 50% and 85%. Moreover, the use of proxy nodes can increase this value by 15% on average.

The authors estimate that, when targeting a miner, its mining effectiveness can be reduced by 36.7% in the worst case.

d: MITIGATION

Since v22[5] [93], nodes do not replace the last outbound HB node if the new one is inbound. This strongly limits the ability to control all peers in the HBN list.

F. FINGERPRINTING ATTACKS

A Fingerprinting attack enables $\mathcal{A}$ to identify non-public nodes, such as unreachable ones and those using anonymity services (Tor,I2P), across different sessions (i.e., separate client executions).

To that purpose, $\mathcal{A}$ typically infers or injects information that allows her to recognize the target in a subsequent session. In particular, fingerprinting techniques can be divided into *block-based*, which identify nodes by the blocks they store, and *address-based*, which inject a set of addresses in a node's database and later retrieve them to recognize the node.

[5]Starting with this release, Bitcoin Core changed the versioning number from 0.x to x.

e: USE

Fingerprinting attacks can be used for Deanonymization: by identifying a node through different sessions, $\mathcal{A}$ can correlate its transactions even when its IP is unknown.

1) STALE-BLOCK FINGERPRINTING

In this attack, described by Jeff Garzik in 2013 [94], $\mathcal{A}$ uses *stale blocks* (i.e., valid blocks that are not included in the main chain) to fingerprint a target node.

a: THREAT MODEL

- $\mathcal{T}=N$: the attack targets a generic node.
- $\mathcal{A}=\mathcal{A}^{I/O}(M) \triangleright \mathcal{T}$: the adversary is connected to $\mathcal{T}$, and uses mining equipment to create and inject blocks, which she later observes.

b: PROCEDURE

$\mathcal{A}$ follows these steps:

1) Generate a block $B_\mathcal{A}$ forking an older block; typically this is done at a very low blockchain height, so that the required PoW is also low.
2) Advertise and transmit $B_\mathcal{A}$ to $\mathcal{T}$.

To recognize $\mathcal{T}$ in a subsequent session, $\mathcal{A}$ requests $B_\mathcal{A}$ via `GETDATA`.

c: MITIGATION

Since v0.9 [95], stale blocks at a lower height than the last checkpoint are not stored.

Similarly, since v0.11 [96], [97], `GETDATA` messages requesting old stale blocks are ignored (current limit is 30 days in time or equivalent PoW values). Since v0.16 [98], nodes also ignore `GETHEADERS` requests for non-main-chain blocks older than 30 days.

2) ADDRESS-COOKIE ATTACK

In this attack, described by Biryukov *et al.* in 2015 [61], $\mathcal{A}$ stores *cookies* in the address database of a target node to recognize it in future sessions.

The attack exploits the fact that Bitcoin clients accept unsolicited `ADDR` messages, and the fact that it is possible to query items from a node's address database using `GETADDR` messages.

a: THREAT MODEL

- $\mathcal{T}=N^U$: the attack targets an unreachable (or anonymous) node.
- $\mathcal{A}=\mathcal{A}^{I/O} \triangleright \mathcal{T}$: the adversary uses a node $N_\mathcal{A}$ connected to $\mathcal{T}$, and injects addresses, which she later observes.

b: PROCEDURE

To set a cookie for $\mathcal{T}$, $\mathcal{A}$ follows these steps:

1) Generate a unique set $\mathbb{F}$ of n fake addresses and send it to $\mathcal{T}$ via ADDR; the message should contain at least $n \geq 11$ addresses, so that it is not forwarded (see § IV-A2).
2) If $\mathcal{T}$ connects directly (i.e. without Tor), save the correspondence between $\mathcal{T}$'s IP and its cookie as a tuple $(\mathbb{F}, IP_{\mathcal{T}})$; otherwise save it as $(\mathbb{F}, nil)$.

To identify $\mathcal{T}$ in a subsequent session, $\mathcal{A}$ follows these steps:

1) Send a number of GETADDR messages to $\mathcal{T}$;
2) Check if the returning ADDR messages contain the previously-set cookie.

As nodes only store addresses belonging to the same type of network, $\mathcal{A}$ should set an Onion-only cookie for Tor nodes, and an IP-only one for other nodes. On the other hand, when replying to GETADDR messages, nodes include both types of address, allowing to verify a cookie regardless of its type.

As, over time, addresses in the database get overwritten by fresher ones, cookies can get partially deleted over time. To make a cookie last longer, $\mathcal{A}$ can refresh it by re-setting it each time she connects to $\mathcal{T}$

Note, however, that even a fraction of the cookie is often sufficient to distinguish a fingerprinted node.

c: IMPACT

Experimental results showed that this attack allowed $\mathcal{A}$ to recognize $\mathcal{T}$ for several hours and over multiple sessions. For instance, in one experiment, after 8 hours and 10 sessions, more than one third of the cookie addresses was still in the database.

Furthermore, while $\mathcal{A}$ might need up to 80 ADDR messages to retrieve a full cookie, about 8 such messages are typically sufficient to retrieve 90% of the cookie.

d: MITIGATION

Since v0.11 [99], GETADDR messages from outbound peers are ignored to protect nodes from *address-cookie* attacks.

Since v0.21 [34], nodes only return up to 1000 addresses per day in response to GETADDR messages, and only reply once per day for each node. This limitation, introduced to mitigate Topology Inference attacks, also affects the ability of the adversary to retrieve *cookie* addresses from the database.

3) BLOCK-SYNC ATTACK

This attack, proposed in 2018 by Mastan and Souradyuti [100], leverages block-request patterns to identify a target over consecutive sessions.

The attack exploits the fact that nodes request blocks in increasing order of blockchain height. Therefore, if $\mathcal{A}$ knows the tip of a node when it leaves the network, she can recognize it when it rejoins, by the fact that it will start from the same height.

a: THREAT MODEL

- $\mathcal{T}{=}Net^P$: the attack targets a portion of the network.
- $\mathcal{A}{=}\mathcal{A}^O \triangleright \mathcal{T}$: the adversary is connected to $\mathcal{T}$, and observes blocks.

b: PROCEDURE

To link $\mathcal{T}$'s sessions, A follows these steps:

1) Log the block IDs requested via GETDATA by nodes in $\mathcal{T}$ in each session;
2) Link consecutive sessions by connecting sequences of block IDs;
3) Build a *session graph*, where each vertex represents a sequence of block IDs, and edges connect consecutive sessions;
4) Link the sessions (not necessarily consecutive) of each node in $\mathcal{T}$ by extracting connected components in the session graph: if two vertices have a path between them, they correspond to the same node.

c: IMPACT

Experimental results show this attack can link consecutive sessions with a precision of 0.90 and a recall of 0.71.

IX. DIRECT ATTACKS

In this section, we review *Direct attacks*, which can be used by $\mathcal{A}$ to pursue DoS, Double Spending, Deanonymization, and Unfair Revenue.

A. DENIAL OF SERVICE (DoS)

DoS attacks in the Bitcoin P2P network can take different forms, and be at node scale, or network scale.

Attacks targeting a single node aim at disrupting its communications, or make it crash. This can be obtained through TCP DoS/DDoS attacks [101], by exploiting client-level vulnerabilities [102], [103], or through Bitcoin-specific network-level attacks, such as Eclipse (§ VIII-C), Delay (§ VIII-E), or Direct attacks such as the Tapsell attack, described in this section.

On the other hand, network-scale attacks typically aim at decreasing the network performances and increasing costs. These attacks are achieved by means of *spamming*, or *flooding*, attacks, which we describe in this section.

1) SPOOFING-BASED DoS: TAPSELL *et al.*'s ATTACK

In this attack, described in 2018 by Tapsell *et al.* [50], $\mathcal{A}$ use Spoofing (§ VII-C) to trigger the transmission of a large amount of data against a target node.

a: THREAT MODEL

- $\mathcal{T}{=}N^R$: the attack targets a single reachable node.
- $\mathcal{A}{=}\mathcal{A}^I(\mathcal{T})$: the adversary knows $\mathcal{T}$'s IP, and injects spoofed messages.

b: PROCEDURE

For the attack, $\mathcal{A}$ spoofs messages with $\mathcal{T}$'s IP address. In particular, she follows these steps:

1) Trick a node N into connecting to $\mathcal{T}$ by sending a spoofed `VERSION` message, followed by a `VERACK` message.
2) Send spoofed `GETHEADERS` or `MEMPOOL` messages to N to trigger a response towards $\mathcal{T}$: a `GETHEADERS` message requesting a number of blocks, results in a `HEADERS` message being sent to $\mathcal{T}$; similarly, a `MEMPOOL` message makes N send $\mathcal{T}$ an `INV` message containing up to 50.000 TXIDs.

To improve the attack, $\mathcal{A}$ tracks the mempool and initiates the attack when its size is maximized. Alternatively, $\mathcal{A}$ can generate a large amount of unconfirmed transactions right before the attack.

c: IMPACT

By spoofing a `GETHEADERS` message, $\mathcal{A}$ can generate a message of 158 Kb by sending just 168 bytes (`VERSION`, `VERACK`, and `GETHEADERS`), with an increase factor of 964. By spoofing a `MEMPOOL` message, $\mathcal{A}$ can generate, in the worst case, an `INV` message as large as 1.7 Mb by sending only 133 bytes of data (`VERSION`, `VERACK`, and `MEMPOOL`), with an increase factor of 13,534.

2) BLOCK-FILLING FLOODING

In this attack, $\mathcal{A}$ generates a vast number of valid transactions to fill up new blocks [104].

Since the space for transactions inside a block is limited (currently to 1 MB), the more transactions $\mathcal{A}$ gets included, the less space is left for other transactions. Therefore, this attack causes legit transactions to experience delays in their confirmation time, and, in the worst case, to get stalled. In turn, this increases the size of the mempool, wasting nodes memory, and eventually degrading transaction validation and propagation performances. As a side effect, mining fees also increase, because users will likely pay more to have their transactions mined faster.[6]

a: THREAT MODEL

- $\mathcal{T}$=*Net*: the attack targets the whole network.
- $\mathcal{A}=\mathcal{A}^I(U)$: the adversary uses *UTXOs* to create and inject transactions.

b: PROCEDURE

$\mathcal{A}$ generates lots of valid and standard transactions (typically sending money to her own accounts).

To maximize efficiency and minimize costs, $\mathcal{A}$ only uses transactions spending the minimum spendable amount.

c: IMPACT

The potential of block-filling DoS was shown in 2015 by a 10-day flood attack, in which 30 BTC were transferred using transactions of 0.00001 BTC [105], [106], [107]. Due to this attack, transaction confirmation times increased by 7 times, from 0.33 to 2.67 hours [104], with peaks of almost 24 hours [108]. Similarly, transaction fees increased from 45 to 68 Satoshis per byte [104].

Curiously, in the attempt of cleaning up these spam transactions, the biggest transaction in Bitcoin history (999 KB) was also produced (by a mining pool), causing even further distress to the network [109].

d: MITIGATION

The use of *minimum relay fees* specifically aims at reducing the risk of spamming attacks. To enforce this measure, in v0.10 [110], the default *minimum relay fee* was increased from 0.00001 to 0.00005 BTC.

3) RESOURCE-WASTING FLOODING

In this type of attack, $\mathcal{A}$ generates transactions aimed at overloading nodes resources, such as bandwidth, CPU, or memory. To reduce costs, $\mathcal{A}$ tries to minimize the probability the such transactions are included in a block.

a: THREAT MODEL

- $\mathcal{T}$=*Net*: the attack targets the whole network.
- $\mathcal{A}=\mathcal{A}^I(U)$: the adversary injects transactions, for which she might need some *UTXO*s.

b: PROCEDURE

To waste network bandwidth, $\mathcal{A}$ generates transactions that will not be included in the blockchain, such as double-spending or orphan transactions. In a more sophisticated approach, $\mathcal{A}$ can exploit transaction malleability (§ VII-A) to create multiple copies of a transaction (possibly from another user), which get validated and propagated, but only one of which will be eventually mined.

To waste CPU, $\mathcal{A}$ creates very large transactions (e.g., with lots of inputs and lots of outputs). As validation depends on the transaction size, these cause heavy computation loads.

To waste memory, $\mathcal{A}$ targets the UTXO set and the mempool. As these structures are typically stored in memory, their size directly impacts the efficiency of validation speed [111]. To bloat the UTXO set, $\mathcal{A}$ creates transactions that split few inputs into many outputs [104]. To bloat the mempool, $\mathcal{A}$ creates either large transactions or multiple *dust* transactions (i.e., spending trivial amounts). In § IX-A4, we show a specific instance of this attack.

c: IMPACT

Memory-wasting attacks are particularly worrisome: since the contents of both UTXO set and mempool are typically shared by all nodes, an increase in their size can affect the efficiency and speed of the whole network.

Again in 2015, an attack brought the mempool size to almost 1 GB, causing a thousand nodes to crash [112]. In 2017, the mempool exceeded 115k due to unconfirmed transactions, resulting in 700 million USD stalled [113].

Other attacks reported in 2015 include a malleability-based attack [114] and a *money drop*, where a number of

[6]Miners are known to prioritize transactions paying higher fees. Thus, such transactions get confirmed faster.

private keys were released to the public to trigger a race to spend them, generating multiple double-spending transactions [104].

d: MITIGATION

To prevent spamming attacks, Bitcoin implements several restrictions to transaction relay. For instance, double-spending transactions are not forwarded: in case of conflict, only the first-seen transaction is processed.

Non-standard transactions are also prohibited. In particular, a transaction is considered as *non-standard* if:

- it is larger than 100 kB;
- it contains dust outputs (i.e., outputs below 546 satoshis);
- it has multiple outputs with the same destination address;
- it contains more than 80 bytes of arbitrary data (using the `OP_RETURN` field).

As for memory bloating, Bitcoin limits the size of the mempool (default size is 300MB). When the limit is reached, transactions with the lowest feerate are evicted. Similarly, the number of orphan transactions stored in memory is also limited (see § IV-B1.c).

4) MEMPOOL FLOODING: SAAD *et al.*'s ATTACK

Mempool-flooding attacks exploit the inherent limitation of Bitcoin transaction processing. In fact, given the low rate block production (approximately 1 block every 10 minutes) and the limited size of blocks, the average throughput in Bitcoin is as low as 3-4 transactions per second [115]. When transactions are produced at a faster rate, confirmation rate decreases, sometimes causing low-value transactions to stall.

In this attack, described in 2018 by Saad *et al.* [44], [116], [117], $\mathcal{A}$ aims at bloating the mempool for a long time by generating a large number of low-value transactions, which are both cheap and have longer confirmation times.

a: THREAT MODEL

- $\mathcal{T}=Net$: the attack targets the whole network.
- $\mathcal{A}=\mathcal{A}^I(U)$: the adversary use *UTXO*s to create and inject transactions.

b: PROCEDURE

1) Divide confirmed *UTXOs* into various self-paying transactions.
2) Use the newly-generated unconfirmed outputs to broadcast new self-paying transactions; as these transactions spend unconfirmed outputs, their confirmation time will be longer, causing transaction backlog and increasing the mempool size.

To minimize the probability of her transactions being mined, $\mathcal{A}$ only pay the minimum relay fee (MRF) for each transaction.

c: IMPACT

The cost of this attack is only as high as the MRFs $\mathcal{A}$ will pay if all transactions get mined.

d: MITIGATION

As shown in [117], the increase in the block size introduced by SegWit [38] reduces the ability to bloat the mempool by increasing the probability of transactions being mined. However, it is worth noting that this increase is only valid for SegWit transactions, potentially leaving space for similar attacks.

B. DOUBLE SPENDING

In this type of attack, the adversary $\mathcal{A}$ deceives a victim $\mathcal{V}$ (usually a merchant) by tricking him to accept a transaction in exchange for a service or good, while having a double-spending one (typically sending the coins back to $\mathcal{A}$) accepted by the rest of the network and mined into the blockchain (thus invalidating the payment). To this purpose, $\mathcal{A}$ targets $\mathcal{V}$'s node, whose IP address must be public and known to $\mathcal{A}$, and manipulates its view of transactions or blocks. The attack occurs in a limited time frame, in which $\mathcal{A}$ obtains the desired service or good from $\mathcal{V}$ before he discovers the scam.

Double-spending attacks can use unconfirmed transactions (*0-confirmation* double-spending) or confirmed transactions (*n-confirmation* double-spending, with $n \geq 1$).

In the rest of this section, we will denote the paying transaction with $tx_{\mathcal{V}}$, and the double-spending one with $tx_{\mathcal{A}}$.

1) RACE ATTACK: DOUBLE SPENDING IN FAST PAYMENTS

In this attack, first described by Karame *et al.* in 2012 [118], $\mathcal{A}$ takes advantage of a scenario in which $\mathcal{V}$ cannot wait for the paying transaction to be mined, due to the long confirmation times (around 20 minutes on average [119], [120]).

In this situation, $\mathcal{V}$ relies on receiving the paying transaction from the network, which supposedly prove the expected payment has been broadcast, and is considered sufficient to believe it will be confirmed. However, $\mathcal{A}$ can deceive $\mathcal{V}$ by sending him the paying transaction while broadcasting a double-spending one to the rest of the network.

a: THREAT MODEL

- $\mathcal{V}=\mathcal{S}^0$, $\mathcal{T}=N^R$: the attack targets a user or merchant accepting zero-confirmation transactions, and connecting through a reachable node.
- $\mathcal{A}=\mathcal{A}^I(B,\mathcal{T}) \triangleright \mathcal{T}$: the adversary knows $\mathcal{T}$'s IP, and connects to it directly; additionally, she uses multiple helper nodes.

b: PROCEDURE

Two conditions have to be met for this attack to succeed: (1) $\mathcal{T}$ has to receive $tx_{\mathcal{V}}$ before $tx_{\mathcal{A}}$, and (2) $tx_{\mathcal{A}}$ has to be mined into the blockchain.

For the attack, $\mathcal{A}$ sends $tx_{\mathcal{V}}$ to $\mathcal{T}$, and, after a short time, $tx_{\mathcal{A}}$ to the helper nodes, which broadcast it to their peers.

To ensure $\mathcal{T}$ receives $tx_{\mathcal{A}}$ after $tx_{\mathcal{V}}$, helper nodes do not connect to $\mathcal{T}$. To help $tx_{\mathcal{A}}$ spread faster (which makes it more likely to be mined), $\mathcal{A}$ can increase the number of helper nodes or the number of the connections they establish.

c: IMPACT

Experimental results show that, when the time between the transmission of $tx_{\mathcal{V}}$ and the broadcast of $tx_{\mathcal{A}}$ is 1 second, and $\mathcal{A}$ uses two helper nodes, the attack is almost guaranteed to succeed. However, the probability of success of the attack is inversely proportional to such time. This is because delaying the broadcast of $tx_{\mathcal{A}}$ allows $tx_{\mathcal{V}}$ to reach more nodes, increasing its probability of being mined.

Among the factors that influence the success rate of race attacks, transaction propagation speed is among the most relevant [63], [121].

On the other hand, these attacks can be made more effective by leveraging other attacks, such as Partitioning (§ VIII-D) and Eclipse (§ VIII-C), which allow the attacker to hide information from the victim node.

d: MITIGATION

In v0.10 [122], the relay of double-spending transactions was introduced, which allowed nodes to detect an ongoing attack. However, the change was reverted shortly after due to issues in the implementation and some disagreement over the necessity of this feature [123]. While claiming this feature might be re-enabled in the future, it is currently out of plans.

Generally speaking, the only secure way to avoid double-spending fast payments is to wait for a transaction to be confirmed [14]. However, several strategies and mechanisms have been proposed [118], [124] for vendors to protect from this attack, including:

- adopt a listening period to verify $tx_{\mathcal{V}}$ is received by other peers;
- connect to more nodes, which increase the probability of receiving the double-spending transaction;
- do no accept incoming connections, preventing $\mathcal{A}$ from opening a connection towards $\mathcal{T}$ (this assumes $\mathcal{A}$ is unaware of $\mathcal{T}$'s peers);
- do not relay transactions: this prevents $\mathcal{T}$ from shielding himself from the reception of $tx_{\mathcal{A}}$; in fact, by forwarding $tx_{\mathcal{V}}$ to all its neighbors, none of these will accept the double-spending $tx_{\mathcal{A}}$, which hence will never reach $tx_{\mathcal{T}}$;

By adopting these measures, the success rate of the attack is reduced to just 0.09% [124].

2) FINNEY ATTACK

This attack, suggested by Hal Finney in 2011 [125], exploits *block withholding* to revert a payment transaction.

a: THREAT MODEL

- $\mathcal{V}{=}\mathcal{S}^0$: the attack targets a seller accepting 0-confirmation transactions.
- $\mathcal{A}{=}\mathcal{A}^I(M)$: the adversary is a miner, and injects a block; for the attack she withhold a transaction.

b: PROCEDURE

$\mathcal{A}$ follows these steps:

1) Create a self-paying transaction $tx_{\mathcal{A}}$, without broadcasting it.
2) Start mining a block $B_{\mathcal{A}}$ that includes $tx_{\mathcal{A}}$.
3) When $B_{\mathcal{A}}$ is found, withhold it, and broadcast $tx_{\mathcal{V}}$.
4) After $\mathcal{V}$ accepted the payment, broadcast $B_{\mathcal{A}}$, overriding $tx_{\mathcal{V}}$ with $tx_{\mathcal{A}}$.

c: IMPACT

The attack succeeds if no other block is found between the broadcast of $tx_{\mathcal{V}}$ and the broadcast of $B_{\mathcal{A}}$. On the other hand, if this occurs, $\mathcal{A}$ loses both $tx_{\mathcal{V}}$'s money and $B_{\mathcal{A}}$'s reward. The attack works even if $\mathcal{V}$ waits a few seconds to verify that the network accepts $tx_{\mathcal{V}}$.

Nevertheless, the success probability of this attack heavily depends on the mining power $\mathcal{A}$ controls. In particular, given the current hashrate of the Bitcoin network, the success probability of an adversary that does not control a considerable fraction of the mining power is negligible.

d: MITIGATION

$\mathcal{V}$ can protect from this attack by only accepting confirmed transactions. While this does not completely exclude the risk, it reduces exponentially the probability of success.

3) Vector76 ATTACK

This attack, proposed in 2011 [126], is a combination of the race and Finney attacks, and can be used for 1-confirmation double spending.

a: THREAT MODEL

- $\mathcal{V}{=}\mathcal{P}$, $\mathcal{T}{=}N^R$: the attack targets an exchange accepting 1-confirmation transactions and using a reachable node.
- $\mathcal{A}{=}\mathcal{A}^{I/O}(M, \mathcal{T}){\triangleright}\mathcal{T}, \mathcal{M}$: the adversary is a miner, which knows $\mathcal{T}$'s IP, and connects to it and to other miners; for the attack, she observes and withholds blocks.

b: PROCEDURE

$\mathcal{A}$ follows these steps:

1) Connect to $\mathcal{T}$ and most of the miners.
2) Create $tx_{\mathcal{V}}$ for the deposit, without broadcasting it.
3) Start mining a block $B_{\mathcal{A}}$ containing $tx_{\mathcal{V}}$.
4) When the block is found, withhold it.
5) As soon as a new block is mined in the network, send $B_{\mathcal{A}}$ to $\mathcal{T}$, which will then believe $tx_{\mathcal{V}}$ has one confirmation.
6) Request a withdraw, create $tx_{\mathcal{A}}$, and broadcast it.

If $B_{\mathcal{A}}$ is accepted into the blockchain, then $\mathcal{A}$ has simply made a deposit and a withdrawal, thus losing nothing. Otherwise, if the withdrawal did not use $tx_{\mathcal{V}}$'s outputs, then $\mathcal{A}$ gains the amount of the deposit.

c: IMPACT

As the attack only lasts around 10 minutes, it can only be successful against exchanges with automated deposits/withdrawals and accepting 1-confirmation transfers, making the attack unlikely to happen in reality [127]. Moreover, in a successful Vector76 attack, the adversary renounces to the reward of the block she produces, which makes it profitable only if the deposit is larger than the block reward.

d: MITIGATION

$\mathcal{V}$ can protect himself by disabling inbound connections, and using trusted outbound peers. Besides this, the most secure protection is to wait for more confirmations.

4) GERVAIS *et al.*'s ATTACK: DELAY-BASED DOUBLE SPENDING

In this attack, described by Gervais *et al.* in 2015 [85], $\mathcal{A}$ use Delay attacks (§ VIII-E) attack to double spend 0- and 1-confirmation transactions.

a: THREAT MODEL

- $\mathcal{V}$=$\mathcal{S}^{0/1}$, $\mathcal{T}$=N^R: the attack targets a seller using a reachable node.
- $\mathcal{A}$=$\mathcal{A}^{I/O}(\mathcal{T}) \triangleright \mathcal{T}, \mathcal{M}$: the adversary has knowledge of $\mathcal{T}$'s IP, and connects to both $\mathcal{T}$ and a miner; for the attack, she observes blocks and delays transactions.

b: PROCEDURE

To double-spend a 0-confirmation transaction, $\mathcal{A}$ follows these steps:

1) Create $tx_{\mathcal{V}}$ and $tx_{\mathcal{A}}$.
2) Broadcast $tx_{\mathcal{A}}$ and use transaction-delay (e.g. § VIII-E2) to prevent $\mathcal{T}$ from receiving it.
3) Meanwhile, forward $tx_{\mathcal{V}}$ to $\mathcal{T}$.
4) If $tx_{\mathcal{A}}$ is included in a block, use block-delay (e.g. § VIII-E3) to retain it from $\mathcal{T}$.

To double-spend a 1-confirmation transaction, $\mathcal{A}$ follows these steps:

1) Connected to a miner $\mathcal{M}$;
2) Send $tx_{\mathcal{V}}$ to $\mathcal{T}$ and $\mathcal{M}$;
3) Broadcast $tx_{\mathcal{A}}$ to the network, and use transaction-delay to prevent $\mathcal{T}$ and $\mathcal{M}$ from receiving it;
4) When $tx_{\mathcal{A}}$ is added to a block B, use block-delay to prevent B from being received by $\mathcal{T}$ and $\mathcal{M}$;
5) Wait for $\mathcal{M}$ to mine a block $B_{\mathcal{M}}$ containing $tx_{\mathcal{V}}$;
6) Deliver $B_{\mathcal{M}}$ to $\mathcal{T}$, making $\mathcal{V}$ believe $tx_{\mathcal{V}}$ has been confirmed.

c: MITIGATION

This attack is mitigated by countermeasures against Delay attacks (§ VIII-E).

5) BALANCE ATTACK: PARTITIONING-BASED DOUBLE SPENDING

In this attack, described by Natoli *et al.* in 2017 [128], $\mathcal{A}$ aims at double spending by partitioning the mining power and delaying the delivery of blocks among subgroups.

a: THREAT MODEL

- $\mathcal{V}$=$\mathcal{S}^1$, $\mathcal{T}$=N^R: the attack targets merchants accepting 1-confirmation payments and connecting with a reachable node.
- $\mathcal{A}$=$\mathcal{A}^{I/O}(M, Net^R, \mathcal{T})$: the adversary is a miner, who knows both the network topology and $\mathcal{T}$'s IP; for the attack, she observes and injects blocks.

b: PROCEDURE

$\mathcal{A}$ performs the following steps:

1) Partition the network in two or more subgroups with balanced hash power.
2) Isolate these subgroups by disrupting the delivery of blocks during a time τ.
3) As soon as a block B is mined in a subgroup G_i, send $tx_{\mathcal{V}}$ to $\mathcal{T}$ in a different group G_j: this way $tx_{\mathcal{V}}$ will be confirmed in G_j's chain.
4) Start mining a private chain $C_{\mathcal{A}}$ on top of B (i.e., the longest chain).
5) Before τ expires, publish $C_{\mathcal{A}}$ to G_i.
6) When $C_{\mathcal{A}}$ is accepted as the main chain, issue $tx_{\mathcal{A}}$, reverting $tx_{\mathcal{V}}$.

c: IMPACT

This attack allows $\mathcal{A}$ to revert a confirmed transaction with high probability. However, this attack only works if miners connect exclusively through the Bitcoin network, i.e., without using relay networks. Moreover, as $\mathcal{A}$ needs to gather knowledge of the hashing power of different miners, of the network topology, and of $\mathcal{T}$'s IP, this attack is extremely hard to perform.

6) ZHANG *et al.*'s ATTACK: SYBIL-BASED DOUBLE SPENDING

In this attack, described in 2019 by Zhang *et al.* [129], $\mathcal{A}$ perform double spending by delaying block propagation.

a: THREAT MODEL

- $\mathcal{V}$=$\mathcal{S}^n$: the attack targets merchants accepting n-confirmation transactions.
- $\mathcal{A}$=$\mathcal{A}^{I/O}(M, B)$: the adversary is a miner controlling multiple non-mining nodes; for the attack she observes and injects blocks.

b: PROCEDURE

The attack aims at reverting a confirmed transactions by mining a private chain. To do so, $\mathcal{A}$ follows these steps:

1) Create $tx_{\mathcal{V}}$ and broadcast it.
2) Start mining a secret chain $C_{\mathcal{A}}$ including a block including the double-spending $tx_{\mathcal{A}}$.
3) Make Sybil nodes delay the delivery of new blocks to all honest peers (see § VIII-E), slowing down the growth of the honest chain: this allows $C_{\mathcal{A}}$ to catch up with the main chain.
4) Wait until the private chain exceeds the main one: when this occurs, publish $C_{\mathcal{A}}$, overwriting $tx_{\mathcal{V}}$ with $tx_{\mathcal{A}}$.

c: IMPACT

The success probability p depends on the $\mathcal{A}$'s share of mining power q and the portion μ of Sybil nodes in the network. Analysis results show that the use of Sybil-based delay significantly improves the chances of double spending: for instance, with q=0.25, $\mathcal{A}$ can revert 10 blocks of the main chain with p=20% when μ=16% and close to p=50% when μ=20%. Moreover, $q \geq 0.32$ $\mathcal{A}$ has 100% probability of a successful attack when μ=20%.

d: MITIGATION

As shown in § VIII-E, recent changes to the block-propagation protocol made Delay attacks harder to implement, as nodes can discover new blocks quickly, even with only a single honest peer connected.

Moreover, the use of relay networks make the attack even harder.

7) MALLEABILITY-BASED DOUBLE SPENDING

In this attack, simply known as *malleability attack*, $\mathcal{A}$ leverages transaction malleability (§ VII-A) to trick a victim $\mathcal{V}$ into sending a payment twice.

Different from other Double Spending attacks $\mathcal{A}$ is not the issuer of a payment but its recipient.

a: THREAT MODEL

- $\mathcal{V}$=$\mathcal{P}$: the attack targets payers or exchanges.
- $\mathcal{A}$=$\mathcal{A}^{I/O}$: the adversary observes a transaction and injects a copy.

b: PROCEDURE

$\mathcal{A}$ follows these steps:

1) When receiving a transaction $tx_{\mathcal{V}}$ from $\mathcal{V}$, create a copy $tx_{\mathcal{A}}$ with a different *TXID*, and broadcast it.
2) Wait for either $tx_{\mathcal{V}}$ or $tx_{\mathcal{A}}$ to be mined (this ensures $\mathcal{A}$ receives the coins).
3) If $tx_{\mathcal{A}}$ gets accepted, complain with $\mathcal{V}$ about $tx_{\mathcal{V}}$ not having been received: if $\mathcal{V}$ checks $tx_{\mathcal{V}}$ by its *TXID*, he will not find any confirmation and thus send the payment again.

The attack succeeds if $\mathcal{V}$ makes use of different coins for the new payment.

c: IMPACT

Andrychowicz *et al.* [49] experimentally showed it was relatively easy to implement this attack, and that most clients were not properly protected.

Malleability attacks in Bitcoin became particularly relevant after the Mt. Gox exchange claimed, in 2014, that around 850,000 BTC were stolen using this technique. Although these claims were not confirmed by later analyses [47], they raised awareness on the problem, and likely caused similar attacks against other exchange platforms. A study by Decker *et al.* [47] showed that more than 28 thousands malleability attacks occurred after the Mt. Gox incident, of which 20% were deemed successful, potentially accounting for more than 60k BTC.

Malleability attacks targeting exchanges are particularly worrying due to the large amount of coins they handle and the high probability of the service using different coins to send repeated payments.

d: MITIGATION

In most cases, protecting from malleability attacks is just a matter of best practices [47], [130], such as checking confirmed transactions outputs instead of relying on *TXIDs* [14].

Morover, Bitcoin Core protects users from this attack by checking transactions by their outputs instead of their *TXID* [14].

C. DEANONYMIZATION ATTACKS

At the network level, a Bitcoin transaction can be deanonymized by identifying the node that generated it, or, more precisely to its IP address. This strategy is based on the assumption that (autonomous) users broadcast their transactions from their own devices. As such, all transactions generated from a given device are likely to belong to the same owner. In other words, a transaction *tx* belongs to a user U if its propagation (via `INV`) started from U's node.

Based on this intuition, the network adversary $\mathcal{A}$ can aim to deanonymize a transaction by determining its source in the network. To do so, $\mathcal{A}$ observes how transactions spread through the network. In this respect, $\mathcal{A}$ has to be connected to the network when the transaction is generated, making the attack infeasible for past transactions. At the same time, once the link between a transaction and a user has been established, it is possible to determine most of his transaction history by analyzing the blockchain [8], [131].

Network-level deanonymization attacks can be directed at a single node (N) or at the whole reachable network (Net^R). In the first case, $\mathcal{A}$ aims at determining which transactions have been created by the target N. For instance, this can be done using Eclipse (§ VIII-C) or MitM (§ VII-B) attacks to intercept all the transactions sent and received by N. On the other hand, network-wide deanonymization aims at detecting the source of all new transactions broadcasted in the network during a period of time, which requires more complex strategies. In this section, we review known network-wide attacks.

1) KAMINSKY ATTACK: NETWORK-WIDE DEANONYMIZATION

Network-wide deanonymization was first proposed by Dan Kaminsky in 2011 [132]. This attack is based on the observation that the node that creates a transaction will be the first one in the network to announce it.

a: THREAT MODEL

- $\mathcal{T}$=Net^R: the attack targets all reachable nodes.
- $\mathcal{A}$=$\mathcal{A}^O \triangleright Net^R$: the adversary connects to all reachable nodes, and observes transactions.

b: PROCEDURE

In the basic attack, $\mathcal{A}$ follows these steps:

1) Connect to all nodes in $\mathcal{T}$.
2) Observe and log `INV` messages announcing transactions.
3) Link each transaction *tx* to the first node sending `INV`(*tx*).

The effectiveness of the attack can be improved by using Sybil nodes, which increase the probability of receiving a transaction from its source.

In 2017, Fanti *et al.* [70] also proposed two additional improvements. First, they let $\mathcal{A}$ establish multiple connections to each node, increasing the probability of receiving a transaction from its source.

Second, they have $\mathcal{A}$ use a *maximum-likelihood estimator* function, which, instead of linking a transaction to the first node to announce it (the authors refer to this strategy as *first-timestamp*), leverages *rumor-centrality* [133]: since transactions are spread symmetrically by each node to their peers, nodes that are close to the source of a transaction will relay it before those far from it. Thus, if $\mathcal{A}$ knows the topology of the network graph (see § VIII-B), she can determine the most-likely source of a transaction by observing the order in which nodes announce it.

c: IMPACT

By implementing the basic strategy, Koshy *et al.* [134] were able to deanonymize several hundred transactions. However, in most cases, this was possible thanks to anomalies in the relay pattern. In contrast, normally-relayed transactions proved to be hard to deanonymize.

Neudecker *et al.* [135] combined the attack with address clustering techniques. Nonetheless, their results showed that, for the vast majority of users, this information does not facilitate deanonymization.

On the other hand, Fanti *et al.* [70] showed how Bitcoin propagation protocols (both Trickle and Diffusion) provide poor anonymity guarantees due to the symmetry of the relay pattern.

d: MITIGATION

Since early versions, Bitcoin Core implemented *trickling* (see § IV-B1.a). This affects the ability of $\mathcal{A}$ to detect the origin of transaction with simple timing analyses [73], [84]. After weaknesses in this method were exposed [134], v0.12 replaced this mechanism with Diffusion [37].

After the Diffusion protocol was also shown to provide poor anonymity guarantees [70], an implementation of the Dandelion protocol [136], [137] was proposed [138]. In this protocol, transactions are relayed over a linear path during the first few hops, and then broadcast using Diffusion. This system protects the identity of the origin by breaking the symmetry in the relay pattern. However, the proposal was eventually rejected due to the risk of introducing new flooding (§ IX-A3) vectors [139].

2) BIRYUKOV *et al.*'s ATTACK

In this attack, proposed in 2014 by Biryukov *et al.* [60], $\mathcal{A}$ targets unreachable nodes. To do so, she distinguish them by the set of their *entry nodes*, that is, their reachable peers. In fact, it is likely that each unreachable node has a unique set of peers. Notably, 3 entry nodes is generally sufficient to uniquely identify a client.

a: THREAT MODEL

- $\mathcal{T}{=}Net^U$: the attack targets unreachable nodes.
- $\mathcal{A}{=}\mathcal{A}^O{\triangleright}Net^R$: the adversary connects to $\mathcal{T}$, and observes transactions.

b: PROCEDURE

$\mathcal{A}$ performs the following steps:

1) Compose a list of targets: this can be based on the addresses advertised through the network, or on known ISP addresses (for NAT nodes).
2) For each target $N^i_{\mathcal{T}}$, use the Topology Inference attack in § VIII-B1 to infer a subset E^i of $N^i_{\mathcal{T}}$'s entry nodes.
3) For each new transaction *tx*:
   - Log the set R_{tx} of the first q nodes to forward `INV`(*tx*);
   - Compare R_{tx} with known sets of entry nodes: if R_{tx} matches the set E^i, link *tx* to $N^i_{\mathcal{T}}$.

c: IMPACT

The success rate of this attack strongly depends on the percentage of connections controlled by $\mathcal{A}$ for each reachable node.

By establishing 50 connections to each server, $\mathcal{A}$ can deanonymize 11% of all new transactions. This rate can reach 60%, if $\mathcal{A}$ first saturates nodes' connections and then gradually replaces them with her own.

d: MITIGATION

The switch from Trickle to Diffusion [37] should have improved the protection against this attack. However, the effectiveness of such measure is unclear [70].

To prevent the fingerprinting method, a periodic randomization of the outbound peers was proposed in 2014 [140]. However, this was eventually discarded due to possible new vectors for Partitioning attacks.

3) WANG *et al.*'s ATTACK

In this attack, described in 2017 by Wang *et al.* [25], $\mathcal{A}$ deanonymize unreachable nodes by using separate monitoring nodes.

a: THREAT MODEL

- $\mathcal{T}{=}N^U$: the attack targets a single unreachable node.
- $\mathcal{A}{=}\mathcal{A}^O(B){\triangleright}\mathcal{T}, Net^R$: the adversary use a monitoring node $N^M_{\mathcal{A}}$ connected to $\mathcal{T}$ only and observing transactions; additionally, she uses a listener node $N^L_{\mathcal{A}}$ connected to all reachable nodes.

b: PROCEDURE

The attack follows these steps:

1) Both $N^M_{\mathcal{A}}$ and $N^L_{\mathcal{A}}$ log INV messages.
2) When a transaction *tx* is received from $\mathcal{T}$, use the following criteria: if $N^M_{\mathcal{A}}$ received *tx* before $N^L_{\mathcal{A}}$, than link *tx* to $\mathcal{T}$; otherwise $\mathcal{T}$ is a relayer.

c: IMPACT

Experimental results show this attack has, on average, 40% of true positives and less then 0.1% false negatives, proving to be a very effective method.

4) PERIMETER: AS-LEVEL DEANONYMIZATION

In this attack, described in 2021 by Apostolaki *et al.* [141], $\mathcal{A}$ makes use of AS-level privileges (§ VII-E2) to deanonymize the transactions of a node by eavesdropping a portion of its connections.

The attack leverages the fact that nodes have a different communication pattern for own transactions and relayed ones. For instance, an own transaction is transmitted (GETDATA/TX) many more times than a relayed one.

a: THREAT MODEL

- $\mathcal{T}=N$: the attack targets a generic node.
- $\mathcal{A}=\mathcal{A}^O_{AS}(\mathcal{T})$: the adversary knows $\mathcal{T}$'s IP and controls an Autonomous System $AS_{\mathcal{A}}$ used by a portion of $\mathcal{T}$'s connections; for the attack, she observes transaction-related messages.

b: PROCEDURE

For the attack, $\mathcal{A}$ performs the following steps:

1) Observe INV, GETDATA, and TX messages exchanged by $\mathcal{T}$.
2) Train an *unsupervised anomaly detection* algorithm, such as an Isolation Forest [142], on the observed traffic.
3) Use the trained algorithm to distinguish $\mathcal{T}$'s own transactions.

c: IMPACT

By intercepting 30% of $\mathcal{T}$'s connections, $\mathcal{A}$ can deanonymize transactions with at least 70% accuracy. Notably, for 50% of Bitcoin nodes, there are at least four ASs with this potential. Moreover, by intercepting 50% of a node's connections, $\mathcal{A}$ is able to deanonymize 90% of new transactions.

The PERIMETER attack is also effective against unreachable nodes, although it is not able to distinguish nodes sharing the same IP address (i.e., behind the same NAT). Similarly, the attack cannot target nodes connecting through Tor.

D. UNFAIR REVENUE

In the PoW consensus model used in Bitcoin, miners earn an average amount of revenues according to the computation effort they contribute to the network. In other words, the more the computing power dedicated to create blocks, the more coins are earned.

However, it has been shown how miners, or mining pools, can obtain an unfair share of revenues, at the expense of their competitors, by breaking the rules. To that purpose, a dishonest miner can follow different strategies, commonly known as *Selfish Mining*.[7]

In this section, we review the basic version of the attack. However, several variations can be found in the literature [67], [144], which aim at improving the effectiveness of the basic approach.

1) SELFISH MINING

In this attack, $\mathcal{A}$ withholds new blocks to gain an advantage over competing miners.

While known since 2010 [145], this attack was formally described in 2014 by Eyal and Sirer [66]. A similar work was also published in 2013 by Bahack [146], who describe it as *block discarding attack*.

a: THREAT MODEL

- $\mathcal{V}=\mathcal{M}$: the attack affects other miners in the network.
- $\mathcal{A}=\mathcal{A}^{I/O}(M)$: the adversary is a miner or mining pool, which produces and withholds blocks.

b: PROCEDURE

$\mathcal{A}$ follows this strategy:

1) Initially mine on the tip B_T of the main chain C_H.
2) If a new block $B_{\mathcal{A}}$ is found before the rest of the network, withhold it, creating a private chain $C_{\mathcal{A}}$.
3) Keep working on top of $B_{\mathcal{A}}$, while honest miners work on B_T.
4) At this point, two cases are possible:
    - If a new block $B2_{\mathcal{A}}$ is found ($C_{\mathcal{A}}$ is two blocks longer than C_H):
        5. Keep mining on $C_{\mathcal{A}}$, publishing a block every time a new block is found by honest miners.
        6. When the distance between $C_{\mathcal{A}}$ and C_H gets back to one block, publish $C_{\mathcal{A}}$: as $C_{\mathcal{A}}$ is longer than C_H, all nodes switch to it, replacing all honest blocks produced in the meantime. In this case, $\mathcal{A}$ gains the revenues of all blocks in $C_{\mathcal{A}}$.
    - If honest miners find a new block B_H:
        5. Immediately publish $B_{\mathcal{A}}$, which will compete against B_H as the new tip: honest miners receiving $B_{\mathcal{A}}$ before B_H start mining on.
        6. At this point, three cases are possible:
            (a) If a new block $B2_{\mathcal{A}}$ is found, $\mathcal{A}$ publishes it immediately, gaining the rewards of both $B_{\mathcal{A}}$ and $B2_{\mathcal{A}}$.

[7]Note that some works also indicate *Block Withholding* (BW) [143] as a form, or even a synonym, of Selfish Mining. Although similar, these attacks are very different in their context and goal. In particular, a BW attack occurs inside an open pool and has such pool as target. Moreover, this attack has no relation with network activity, which is why we do not include it in this survey.

(b) If honest miners find a new block $B2_H$ on top of $B_\mathcal{A}$, $\mathcal{A}$ only gains the reward of $B_\mathcal{A}$.

(c) If $B2_H$ is mined on top of B_H, $\mathcal{A}$ gains nothing.

c: IMPACT

According to Eyal and Sirer [66], the profitability of the attack depends on the portion of mining power controlled by $\mathcal{A}$ (α) and the portion of honest miners which choose to work on the $B_\mathcal{A}$ in the block race against B_H (γ). In particular, when γ=0.99, the threshold for a profitable attack is α>0.009. Therefore, any mining pool, regardless of its size, can profit from this attack if the block race is favorable. At the same time, when $\alpha \geq 0.3$ ($\frac{1}{3}$ of the mining power), $\mathcal{A}$ will always be able to gain revenues exceeding its proportion, even when losing every block race (i.e., γ=0).

According to Bahack [146], when $\mathcal{A}$ controls a fraction p of the mining power, she can increase profits from p (the fair share) to $\frac{p}{1-p}$. This analysis also confirms that, when $p>\frac{1}{3}$, the attack is profitable even when honest nodes never mine on top of the adversarial block.

Bahack also highlights the importance of the so-called *network superiority* to increase the chances of winning a block race. This superiority is obtained by using highly-connected Sybil nodes, which allow $\mathcal{A}$ to both receive and propagate blocks faster than other miners. With this advantage, $\mathcal{A}$ can always profit by withholding new blocks until a competing one is discovered. In fact, with network superiority, $B_\mathcal{A}$ is always likely to win the race against B_H. At the same time, she gains advantage over other miners by starting to mine earlier on the private block.

In that respect, Auxiliary attacks, such as Eclipse (§ VIII-C), Delay (§ VIII-E), and Topology Inference (§ VIII-B) allow to further improve the effectiveness of Selfish Mining. In particular, an Eclipse attack can sensibly increase $\mathcal{A}$'s revenues when implementing a Selfish Mining strategy [67]. Delay attacks could also lower the 33% bound ($\alpha \geq 0.3$) if a considerable fraction of miners only rely on the Bitcoin P2P network (i.e., they do not use Relay networks); for instance, by preventing the delivery of 2 consecutive blocks from 50% of the network, $\mathcal{A}$ can profit from Selfish Mining even with $\alpha \geq 0.26$. Even worse, an adversary controlling less than 34% of the computing power could effectively sustain the longest block chain and control the network (i.e., the equivalent of a 50% attack).

d: MITIGATION

To date, there are no explicit measures deployed in Bitcoin to counter Selfish Mining. This is likely due to two main factors: first, the attack is somewhat complex, risky, and requires controlling a large share of mining computation to be effective; second, most proposed countermeasures [66], [147], [148] imply a change in the consensus protocol, which might require a hard fork to be implemented.

Additionally, the use of relay networks greatly reduces the ability to improve Selfish Mining using Auxiliary attacks.

X. DISCUSSION

In the previous sections, we reviewed and classified network-level attacks against the Bitcoin P2P protocol. In particular, we organized available information in a structured format, based on a generalized Network Adversary model and an objective-focused attack taxonomy.

In this section, we concisely summarize such information, and take a broad-spectrum look over it to infer useful insights on the Bitcoin network security.

A. ATTACKS OVERVIEW

In Table 1, we summarize all the attacks reviewed in this paper (except for Infrastructure attacks, which operate outside the Bitcoin protocol). For each attack, we specify target and adversary model, the Auxiliary attacks it depends on or can helped by, the type of weakness exploited, and the type of mitigation.

By looking at this table, it is possible to infer common characteristics of network-level attacks, such as their typical setting (adversary and targets) and which components of the protocol are involved (weaknesses and mitigation). In the following, we report our observations.

e: ADVERSARIES AND TARGETS

Most attacks require the adversary to be connected to the target. For this reason, the target is often a reachable node (or multiple ones), whose IP address is known. Similarly, the adversary often connects to all reachable nodes to observe or manipulate the propagation of data. To that purpose, she typically deploys multiple nodes, through which she actively interacts by injecting, modifying, or dropping messages. Less commonly, the adversary targets unreachable nodes and miners. Unreachable nodes are both more vulnerable, when connected to an adversary, and less exposed, because she cannot ensure such a connection. On the other hand, miners are typically protected by the use of relay networks. Attacks targeting the whole Bitcoin network are rare and typically limited to DoS and Partitioning attacks.

While the above-mentioned characteristics give a generic overview of network-level attacks, it is interesting to also study the threat model in the different attack categories. In particular, by looking at the table, we can observe the following settings:

- Direct attacks:
  - DoS: in these attacks, the adversary typically targets the whole network by injecting transactions; less commonly, she targets specific nodes, whose IP is known.
  - DS: in these attacks, the adversary is typically a miner, injecting blocks, and targeting a merchant connected with a reachable node, whose IP is known.
  - D: in these attacks, the adversary typically connects to the target and all other reachable nodes, and observes transactions.

TABLE 1. Network-Level Attacks on the Bitcoin P2P Network: for each attack, we report adversary and target according to our model (§ V), use of Auxiliary attacks (the asterisk indicates such use is optional and only required to improve the attack), the component of the protocol exploited for the attack, and the type of countermeasure implemented in the protocol (*Tx*, *Blk*, and *Addr* denote transaction, block, and address, respectively).

	Attack	Target	Adversary	Dependencies	Weakness	Mitigation
	Sybil	N,Net	$\mathcal{A}(B) \triangleright \mathcal{T}$	/	Peering	Improve Peering, Fix Addr Storage
Topology Inference	Biryukov et al. (Unreachable)	N^U	$\mathcal{A}^O \triangleright Net^R$	Sybil (opt)	Addr Propagation	None
	Biryukov et al. (Reachable)	N^R	$\mathcal{A}^{I/O}(\mathcal{T}) \triangleright \mathcal{T}$	Sybil (∗)	Addr Propagation	None
	AddressProbe	Net^R	$\mathcal{A}^O \triangleright Net^R$	/	Addr Propagation	Fix Addr Storage, Limit Propagation
	Neudecker et al.	Net^R	$\mathcal{A}^{I/O}(B) \triangleright Net^R$	Sybil, Deanon. (∗)	Tx Propagation	Limit Propagation
	Grundmann et al. (Tx-Acc)	Net^R	$\mathcal{A}^{I/O}(B) \triangleright Net^R$	Sybil	Tx Propagation	None
	Grundmann et al. (DS)	N^R	$\mathcal{A}^{I/O}(\mathcal{T}) \triangleright Net^R$	/	Tx Propagation	None
	TxProbe	Net^R	$\mathcal{A}^{I/O} \triangleright Net^R$	InvBlock	Tx Propagation, Tx Storage	Fix Tx-DB, →*InvBlock*
Eclipse	Heilman et al.	N^R	$\mathcal{A}^I(B,\mathcal{T}) \triangleright \mathcal{T}$	Sybil	Addr Storage	Fix Addr-DB, Improve Peering
	EREBUS	N^R	$\mathcal{A}^I_{AS}(\mathcal{T}) \triangleright \mathcal{T}$	MitM	Addr Storage, Peering	Improve Peering
Partition.	Neudecker et al.	Net^R	$\mathcal{A}^I(B, Net^R)$	Sybil, Top. Inf.	Peering	→ Top. Inf.
	Apostolaki et al.	Net^P	$\mathcal{A}_{AS}(\mathcal{T})$	BGP Hijack	Peering	Improve Peering
Delay	Timejacking	N^R	$\mathcal{A}^I(M,B,\mathcal{T}) \triangleright \mathcal{T}$	Sybil	Blk Propagation	Use Outbound
	InvBlock (Tx)	N	$\mathcal{A}^I \triangleright \mathcal{T}$	/	Tx/Blk Propagation	Fix Relay, Use Outbound
	InvBlock (Blk)	N	$\mathcal{A}^I \triangleright \mathcal{T}$	/	Tx/Blk Propagation	Improve Relay
	Apostolaki et al.	N	$\mathcal{A}^I_{AS}(\mathcal{T})$	MitM, BGP Hijack (∗)	Blk Propagation	Improve Relay
	TendrilStaller	N^R	$\mathcal{A}(B) \triangleright \mathcal{T}$	Sybil	Blk Propagation	Use Outbound
Fingerprint.	Stale-Block	N	$\mathcal{A}^{I/O}(M) \triangleright \mathcal{T}$	/	Blk Storage	Limit Relay
	Address-Cookie	N^U	$\mathcal{A}^{I/O} \triangleright \mathcal{T}$	/	Addr Storage	Limit Relay
	Block-Sync	Net^P	$\mathcal{A}^O \triangleright \mathcal{T}$	/	Blk Propagation	None
DoS	Tapsell et al.	N^R	$\mathcal{A}^I(\mathcal{T})$	Spoofing	Tx Processing, Blk Propagation	None
	Flooding (block-fill)	Net	$\mathcal{A}^I(U)$	/	Blk Processing	Limit Relay
	Flooding (resource-waste)	Net	$\mathcal{A}^I(U)$	/	Tx Processing	Limit Relay, Limit Tx-DB
	Saad et al.	Net	$\mathcal{A}^I(U)$	/	Tx Processing	Improve Processing
DS	Race	$\mathcal{S}^0\ (N^R)$	$\mathcal{A}^I(B,\mathcal{T}) \triangleright \mathcal{T}$	Sybil	Tx Propagation	None
	Finney	$\mathcal{S}^0$	$\mathcal{A}^I(M)$	/	Blk Propagation	None
	Vector76	$\mathcal{S}^1\ (N^R)$	$\mathcal{A}^{I/O}(M,\mathcal{T}) \triangleright \mathcal{T},\mathcal{M}$	/	Blk Propagation	None
	Gervais et al.	$\mathcal{S}^{0/1}\ (N^R)$	$\mathcal{A}^{I/O}(\mathcal{T}) \triangleright \mathcal{T},\mathcal{M}$	Delay	Tx/Blk Propagation	→Delay
	Balance	$\mathcal{S}^1\ (N^R)$	$\mathcal{A}^{I/O}(M, Net^R, \mathcal{T})$	Partition, Delay	Blk Propagation	→Partition, →Delay
	Zhang et al.	$\mathcal{S}^n$	$\mathcal{A}^{I/O}(M,B)$	Sybil Delay	Tx/Blk Propagation	→Delay
	Malleability	$\mathcal{P}$	$\mathcal{A}^{I/O}$	Tx Malleability	Tx Processing	Improve Processing
D	Kaminsky	Net^R	$\mathcal{A}^O \triangleright Net^R$	Top. Inf. (∗), Sybil (∗)	Tx Propagation	Limit Relay
	Biryukov et al.	Net^U	$\mathcal{A}^O \triangleright Net^R$	Top. Inf.	Tx Propagation	Limit Relay
	Wang et al.	N^U	$\mathcal{A}^O(B) \triangleright \mathcal{T}, Net^R$	Sybil	Tx Propagation	None
	PERIMETER	N	$\mathcal{A}^O_{AS}(\mathcal{T})$	MitM, BGP Hijack (∗)	Tx Propagation	None
UR	Selfish Mining	$\mathcal{M}$	$\mathcal{A}^{I/O}(M)$	Eclipse (∗), Delay (∗) Top. Inf. (∗)	Blk Propagation	None

    – UR: in these attacks, the adversary is a miner withholding blocks from other miners to gain profits.
- Auxiliary attacks:
    – Sybil: in these attacks, the adversary uses multiple nodes to establish connections to the target.
    – Topology Inference: in these attacks, the adversary typically connects to the whole reachable network, and injects addresses and transactions; less commonly, she only observes the propagation of data.
    – Eclipse: in these attacks, the adversary connects to a single node, and injects addresses.
    – Partitioning: in these attacks, the adversary targets a large portion of the network, and prevents the propagation of data.
    – Delay: in these attacks, the adversary targets a single node, from which she withholds transactions and blocks.
    – Fingerprinting: in these attacks, the adversary connects to a single unreachable node, and injects addresses or blocks.

f: WEAKNESSES AND MITIGATION

Network-level attacks essentially exploit weaknesses in the management of data, that is the propagation, processing, and storage of:

- Transactions: flaws in their propagation, verification, and mining, enable DoS, Double Spending, Deanonymization, Topology Inference, and Delay attacks;
- Blocks: flaws in their propagation enable DoS, Delay, and Fingerprinting attacks;
- Addresses: flaws in their propagation and storage enable Topology Inference, Eclipse, Partitioning, and Fingerprinting attacks.

Additionally, *time* management also shows to be relevant. This can be primarily seen in Delay attacks, but is also evident in features such as timeouts, timestamps, or trickling relay.

From a different perspective, network-level attacks involve the following protocol functionalities:

- *Connection Management*: issues in this functionality can lead to Sybil, Eclipse, and Partitioning attacks; in turn, related countermeasures include limitations and improvements to connectivity, such as subnet-/AS-aware peer selection, eviction mechanisms, block-relay-only peers, and anchoring.
- *Relay/Processing*: issues in these functionalities can lead to Topology Inference, Delay, DoS, Double Spending, Deanonymization, and Selfish Mining attacks; in turn, related countermeasures include limitations and improvements to propagation, such as timeouts, minimum fees, Trickling/Diffusion, direct headers announcement, compact blocks, and the preferential/exclusive use of outbound peers (see § IV-A3.a).
- *Storage*: issues in this functionality can lead to Eclipse and Fingerprinting attacks; in turn, related countermeasures include limitations and improvements to local databases, such as timestamps, bounded block age, bounded pool sizes, eviction mechanisms, and limited querying.

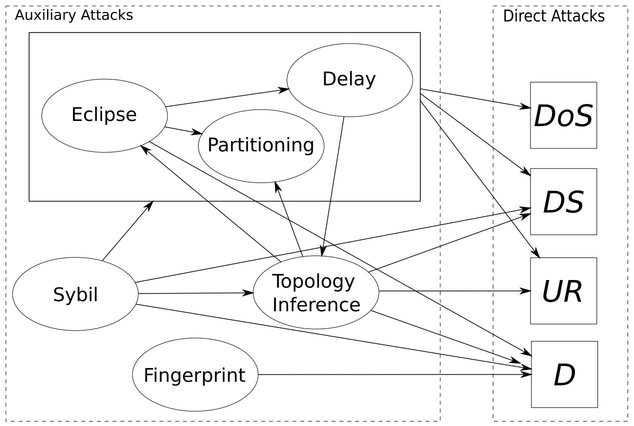

FIGURE 4. Relation between network-level attacks: squares and ellipses represent primary and Auxiliary attacks respectively. An arrow from A to B indicates that the attacks of type A influences attacks of type B.

Based on these observations, we identify two core protocol components whose design is of paramount importance for the security of the Bitcoin P2P network:

- *Peering*: this component determines how each node connects to other peers, and includes functionalities like propagation and storage of addresses, selection of outbound peers, and eviction mechanisms (peers and addresses);
- *Sharing*: this component determines how data spreads through the network, and includes functionalities like propagation, verification, and storage of transactions and blocks.

B. AUXILIARY ATTACKS RELATIONSHIP

Auxiliary attacks are a prominent threat for the Bitcoin network, as they enable the adversary to achieve one or more primary goals, or to improve the effectiveness of other attacks. In this section, we study how Auxiliary attacks relate to each other and to Direct attacks. We summarize this relationship in Table 2 and graphically represent it in Figure 4.

These resources allow us to infer interesting information about Auxiliary attacks. First of all, it is clear how the Sybil attack is the most influential, enabling all other Auxiliary attacks (except Fingerprinting), and facilitating Double Spending and Deanonymization. The Eclipse attack is arguably the most powerful, enabling all Direct attacks as well as Delay and Partitioning. Topology Inference is also relevant, influencing Double Spending, Deanonymization, and Unfair Revenue, while also facilitating Eclipse and Partitioning attacks. On the other hand, Fingerprinting is only related to Deanonymization.

Another interesting observation is the fact that Eclipse, Partitioning, and Delay attacks have an almost overlapping

TABLE 2. Relationship between network-level attacks: an *x* in a cell indicates the attack in the corresponding row influences (i.e., enables, facilitates, or improves) the attack in the corresponding column.

	DoS	DS	UR	D	Sybil	Top. Inf.	Eclipse	Part.	Delay	Fing.
Sybil		x		x	N/A	x	x	x	x	
Top. Inf.		x	x	x		N/A	x	x		
Eclipse	x	x	x	x			N/A	x	x	
Part.	x	x	x					N/A		
Delay	x	x	x			x			N/A	
Fing.				x						N/A

set of influence relationships, indicating they are strictly related to each other. In fact, these attacks can be conceptually seen as variations of a *MitM*-like attack where the adversary affects the communications between a target and the rest of the network. In Figure 4, we reflect this commonality by grouping the three attacks in a single block. In addition, we introduce the term *EPD attacks* (from the initials of their names) to refer to these attacks collectively.

Similarly, for Direct attacks, we can observe a close relation between Double Spending and Unfair Revenue, which share most of the influences, suggesting that they might belong to a common threat category. On the other hand, using the same criteria, DoS and Deanonymization seem to belong to a separate category.

XI. STATUS AND RESEARCH DIRECTIONS

As evidenced from our references, most Bitcoin network issues have been exposed, and in many cases addressed, within the first years following its launch. This shows how the Bitcoin protocol benefited from the enthusiasm, and contribution, of both its user base and related academic research. Thanks to these early efforts, the Bitcoin P2P protocol is arguably among the most complete and secure among blockchain networks today.

Nevertheless, several issues still exist, which should be addressed. In the following, we summarize the current state of Bitcoin network security.

A. DIRECT ATTACKS

Direct attacks have been extensively addressed in the Bitcoin protocol, and are arguably the least dangerous nowadays.

g: DoS

Network-scale denial-of-service attacks on the Bitcoin network are theoretically possible but hard to implement. There are a number of measures in place in the protocol to prevent malicious users from clogging up the network. Currently, mempool attacks [116] are the most realistic strategy for a DoS attack. This risk has been mitigated by the introduction of bigger blocks, although this is conditional to the use of SegWit transactions. For single nodes, spoofing-based attacks might still be a concern [50].

h: DOUBLE SPENDING

Double-spending attacks have been extensively addressed in the Bitcoin protocol, and are currently unlikely to be performed successfully. To achieve this objective in a direct way, the adversary is required to control a large portion of the network nodes and to have powerful mining resources.

The Balance attack [128] has not explicit mitigation in the protocol. However, its implementation is very complex.

i: DEANONYMIZATION

Transaction deanonymization is likely the most realistic issues in the current Bitcoin protocol. In fact, it has been shown that the current relay protocol offers very poor anonymity properties for reachable nodes [45]. While some solutions have been proposed in research, no effective measure has been implemented so far. Unreachable nodes can also be easily deanonymized [25].

Furthermore, the PERIMETER attack [141] can be used against all nodes in the network. No mitigation has been yet introduced in the protocol.

j: UNFAIR REVENUE

Selfish-mining [66] and other akin strategies [67], [144] are currently another open problem and a prominent topic in Bitcoin security research. However, very few real-world cases have been reported so far, probably due to the resources needed for the attack, the implementation complexity of the strategies, and the relatively-low profits for the attacker.

At the same time, not much can be done at the network level without introducing changes to prevent block-withholding and similar malicious behaviors.

B. INFRASTRUCTURE ATTACKS

Many of these attacks have been exposed during the early stages of the Bitcoin network. However, due to the difficulty in finding proper solutions, and partly to lack of encrypted communications, most of them are still a realistic threat.

k: TRANSACTION MALLEABILITY

While transaction malleability has been fixed in the current protocol, it is still possible to use this vulnerability in legacy transactions to perform spamming attacks.

l: TCP/IP
Due the lack of encryption, TCP/IP attacks like MitM and spoofing are still an open problem in Bitcoin.

m: DNS
Due to the inherent centralization of this service, DNS-related attacks are still a risk when instantiating a new network node. Nevertheless, the slow transition to DNSSEC and the parallel use of multiple servers minimize the threat.

n: AS
AS-level attacks are currently a prominent threat for Bitcoin communications, for which few countermeasures currently exist. However, these attacks require a highly-privileged adversary (e.g. an ISP), which can be potentially exposed by the attack itself. This acts as a potential deterrent for the attacker, making them less likely to occur.

o: TOR
Tor-related attacks mostly depended on a specific anti-DoS feature of the Bitcoin protocol, and should not be considered as a threat today.

C. AUXILIARY ATTACKS
This type of attacks has received a lot of attention from the Bitcoin community, due to the disruptive potential the adversary obtains when they are successful. Thanks to these efforts, most attacks have been mitigated in the current protocol, with few exceptions. At the same time, new attack vectors are regularly discovered by researchers, showing the network is not fully protected from these threats.

p: SYBIL
Sybil attacks are inherent to permissionless networks are nearly impossible to prevent. The Bitcoin protocol deals with this issue by assuming a Sybil attacker can always exist. Based on that assumption, nodes diversify their connections to maximize the chance to connect to at least one honest node.

q: TOPOLOGY INFERENCE
This category is one of the most debated threat in Bitcoin, with numerous techniques disclosed, and promptly fixed, over the years. Currently, no explicit mitigation has been introduced for techniques in [60] and [74].

The need of topology obfuscation has been questioned by some authors, who openly advocate for an open topology strategy [65].

r: ECLIPSE
Eclipse attacks are a major threat for Bitcoin nodes, which rely on their peers to keep their status up-to-date. While numerous countermeasures have been adopted in the Bitcoin protocol, it is possible to that other vectors might still exist. Current countermeasures against the EREBUS attack [82] only protect from block eclipsing, leaving the nodes exposed to transaction eclipsing.

s: PARTITIONING
Partitioning attacks are currently possible in the Bitcoin network at both node level [84] and AS level [58]. This risk is currently mitigated by obfuscating the network topology and by diversifying nodes connections.

t: DELAY
Except for timejacking [86], most of these attacks have been mitigated in the Bitcoin protocol, which currently implement privileged channels to rapidly receive blocks from the network.

u: FINGERPRINTING
Currently, the Block-Sync attack [100] has still no mitigation in the protocol. This type of attacks can be a threat when combined with deanonymization techniques.

D. RESEARCH DIRECTIONS
This paper provides numerous sources of information for researchers willing to investigate open problems in the Bitcoin P2P network security.

First of all, the summary provided in this section indicates all the attacks or categories for which no definitive solution has been found or implemented.

Furthermore, by observing Table 2 and Figure 4, it is possible to identify potentially missing relations between attacks. For instance, possible research questions might be: can Partitioning and Delay attacks improve Deanonymization? can Topology Inference influence Delay attacks? can Sybil attacks allow DoS or Fingerprinting?

At a more abstract level, it would also be interesting to study the intrinsic relation between Eclipse, Delay, and Partitioning attacks: is it possible to entail the three attacks under a single category? Similarly, the relation between these and Sybil and Topology Inference could also be analyzed.

Another important research direction is towards a formalization of the Bitcoin P2P protocol and its security. The information provided in this paper could be used as a basis for creation of a network model and to define the fundamental threats to the security of its nodes and users.

To that respect, another useful task for future research would be to gather and organize information on the Bitcoin P2P protocol, with both high-level documentation and low-level details. Today, both new and experienced researchers have to face the challenge of learning how the protocol actually works. Having a single, complete source of information would be immensely beneficial for the community, and eventually strengthening Bitcoin security.

XII. CONCLUSION
Bitcoin network-level security is a complex subject. Lack of official protocol documentation and scattered information on relative threats and solutions make it hard to properly study the subject. In turn, this hampers the identification of

structural issues in the protocol, and the design of long-term solutions.

In an effort to help the community fill this gap, and move towards a more formal network security model, we provided a detailed compendium of known threats and solutions in the Bitcoin network.

In particular, we described the most recent version of the Bitcoin P2P protocol, and systematically reviewed all network-level attacks, along with implemented countermeasures. We then proposed a new security framework composed of a generic network-level adversary and an objective-based attack taxonomy. This framework allows to quickly identify the main characteristics of each attack, and understand its implications with respect to the general setting of the Bitcoin network. Moreover, it enables a systematic comparison with other network-level attacks.

Based on this framework, we extracted useful information on common attack settings, inferred structural weaknesses and solutions, and identified the core protocol components that relate to Bitcoin security.

While the protocol definition may become obsolete over time, the adversary model and the attack taxonomy can serve as a basis for future research and development. Furthermore, our framework can be applied to other cryptocurrency networks that share a similar network model and protocol.

We believe this work can help researchers and developers better understand the Bitcoin P2P protocol and eventually pursue a more formally-secure blockchain network.

REFERENCES

[1] CoinMarketCap. *Total Cryptocurrency Market Cap*. Accessed: Oct. 14, 2021. [Online]. Available: https://coinmarketcap.com/charts/

[2] CoinMarketCap. *Bitcoin Price Today, Market Cap and Chart*. Accessed: Oct. 14, 2021. [Online]. Available: https://coinmarketcap.com/currencies/bitcoin/

[3] (2020). *Bitnodes Global Bitcoin Nodes Distribution*. Accessed: Feb. 22, 2022. [Online]. Available: https://bitnodes.io

[4] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Advances in Cryptology EUROCRYPT*. Berlin, Germany: Springer, 2015, pp. 281–310.

[5] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 3–16.

[6] T. Volety, S. Saini, T. McGhin, C. Z. Liu, and K.-K. R. Choo, "Cracking bitcoin wallets: I want what you have in the wallets," *Future Gener. Comput. Syst.*, vol. 91, pp. 136–143, Feb. 2019.

[7] P. K. Kaushal, A. Bagga, and R. Sobti, "Evolution of bitcoin and security risk in bitcoin wallets," in *Proc. Int. Conf. Comput., Commun. Electron. (Comptelix)*, Jul. 2017, pp. 172–177.

[8] D. Ron and A. Shamir, "Quantitative analysis of the full bitcoin transaction graph," in *Financial Cryptography and Data Security*. Berlin, Germany: Springer, 2013, pp. 6–24.

[9] M. Saad, J. Spaulding, L. Njilla, C. Kamhoua, S. Shetty, D. Nyang, and D. Mohaisen, "Exploring the attack surface of blockchain: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 1977–2008, 2020, doi: 10.1109/COMST.2020.2975999.

[10] Y. Wen, F. Lu, Y. Liu, and X. Huang, "Attacks and countermeasures on blockchains: A survey from layering perspective," *Comput. Netw.*, vol. 191, May 2021, Art. no. 107978.

[11] A. Hassan, "A systematic literature review on the security and privacy of the blockchain and cryptocurrency," *OIC-CERT J. Cyber Secur.*, vol. 2, no. 1, pp. 1–17, 2020. [Online]. Available: https://www.oic-cert.org/en/journal/pdf/2/1/211.pdf

[12] C. A. Vyas and M. Lunagaria, "Security concerns and issues for bitcoin," *Int. J. Comput. Appl.*, pp. 10–12, May 2014. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.667.5197&rep=%rep1&type=pdf

[13] S. Shalini and H. Santhi, "A survey on various attacks in bitcoin and cryptocurrency," in *Proc. Int. Conf. Commun. Signal Process. (ICCSP)*, Apr. 2019, pp. 220–224.

[14] M. Conti, S. Kumar, C. Lal, and S. Ruj, "A survey on security and privacy issues of bitcoin," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 3416–3452, 4th Quart., 2018.

[15] T. Cao, J. Yu, J. Decouchant, and P. Verissimo. (2018). *Revisiting Network-Level Attacks on Blockchain Network*. [Online]. Available: https://orbilu.uni.lu/bitstream/10993/38142/1/bcrb18-cao.pdf

[16] M. Mostafa, "Bitcoin's blockchain peer-to-peer network security attacks and countermeasures," *Indian J. Sci. Technol.*, vol. 13, no. 7, pp. 767–786, Feb. 2020.

[17] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Bus. Rev.*, Oct. 2008, Art. no. 21260. [Online]. Available: https://www.debr.io/article/21260.pdf

[18] B. Wiki. *Clients*. Accessed: Feb. 23, 2022. [Online]. Available: https://en.bitcoin.it/wiki/Clients

[19] (2019). *Fibre Fast Internet BItcoin Relay Engine*. Accessed: Dec. 28, 2020. [Online]. Available: http://bitcoinfibre.org/

[20] (2016). *Falcon—A Fast BItcoin Backbone*. Accessed: Dec. 28, 2022. [Online]. Available: https://falcon-net.org/

[21] *Tor Project Anonymity Online*. Accessed: Nov. 3, 2021. [Online]. Available: https://www.torproject.org

[22] *I2p Anonymous Network*. Accessed: Feb. 22, 2022. [Online]. Available: https://geti2p.net

[23] *The Number of Bitcoin Network Nodes Has Reached an All-Time High*. Accessed: Feb. 28, 2022. [Online]. Available: https://www.kogocrypto.com/the-number-of-bitcoin-network-nodes-has-reac%hed-an-all-time-high

[24] P. Srisuresh and M. Holdrege, *IP Network Address Translator (NAT) Terminology and Considerations*, document RFC 6528, The Internet Society, Aug. 1999. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2663

[25] L. Wang and I. Pustogarov, "Towards better understanding of bitcoin unreachable peers," 2017, *arXiv:1709.06837*.

[26] A. Miller, J. Litton, A. Pachulski, N. Gupta, D. Levin, N. Spring, and B. Bhattacharjee. (2015). *Discovering Bitcoin's Public Topology and Influential Nodes*. [Online]. Available: https://www.cs.umd.edu/projects/coinscope/coinscope.pdf

[27] S. Delgado-Segura, S. Bakshi, C. Pérez-Solà, J. Litton, A. Pachulski, A. Miller, and B. Bhattacharjee, "TxProbe: Discovering bitcoin's network topology using orphan transactions," 2018, *arXiv:1812.00942*.

[28] (2019). *Bitcoin Wiki Protocol Documentation*. Accessed: Dec. 3, 2019. [Online]. Available: https://en.bitcoin.it/wiki/Protocol_documentation

[29] A. M. Antonopoulos, *Mastering Bitcoin: Unlocking Digital Cryptocurrencies*. Sebastopol, CA, USA: O'Reilly Media, 2014.

[30] *Bitcoin Core Integration/Staging Tree*. Accessed: Feb. 23, 2022. [Online]. Available: https://github.com/bitcoin/bitcoin

[31] S. Daftuar. (Sep. 2019). *Pr 15759: Add 2 Outbound Block-Relay-Only Connections*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/15759

[32] H. Stepanov. (Nov. 2019). *Pr 17428: Try to Preserve Outbound Block-Relay-Only Connections During Restart*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/17428

[33] P. Strateman. (Jul. 2015). *Pr 6374: Connection Slot Exhaustion Dos Mitigation*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/6374

[34] G. Naumenko. (May 2020). *Pr 18991: Cache Responses to Getaddr to Prevent Topology Leaks*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/18991

[35] G. Maxwell. (Jan. 2019). *Pr 14929: Allow Connections From Misbehavior Banned Peers*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/14929

[36] Wikipedia. (2020). *Poisson Distribution*. Accessed: Dec. 31, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Poisson_distribution

[37] P. Wuille. (Nov. 2015). *Pr 7125: Replace Global Trickle Node With Random Delays*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/7125

[38] E. Lombrozo, J. Lau, and P. Wuille. (2015). *Bip141: Segregated Witness (Consensus Layer)*. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki

[39] P. Wuille. (Apr. 2016). *Pr 7910: Segregated Witness*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/7910

[40] S. Daftuar. (2016). *Bip130: Sendheaders Message*. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0130.mediawiki

[41] S. Daftuar. (2015). *Pr 6494: Allow Block Announcements With Headers*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/6494

[42] M. Corallo. (2016). *Bip152: Compact Block Relay*. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki

[43] M. Corallo. (May 2016). *Pr 8068: Compact blocks*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/8068

[44] M. Saad, L. Njilla, C. Kamhoua, J. Kim, D. Nyang, and A. Mohaisen, "Mempool optimization for defending against DDoS attacks in PoW-based blockchain systems," in *Proc. IEEE Int. Conf. Blockchain Cryptocurrency (ICBC)*, May 2019, pp. 285–292.

[45] G. Fanti and P. Viswanath, "Deanonymization in the bitcoin P2P network," in *Proc. Adv. Neural Inf. Process. Syst.* Red Hook, NY, USA: Curran Associates, 2017, pp. 1364–1373. [Online]. Available: http://papers.nips.cc/paper/6735-deanonymization-in-the-bitcoin-p2p-net%work.pdf

[46] Q. Do, B. Martini, and K.-K. R. Choo, "The role of the adversary model in applied security research," *Comput. Secur.*, vol. 81, pp. 156–181, Mar. 2019.

[47] C. Decker and R. Wattenhofer, "Bitcoin transaction malleability and MtGox," in *Computer Security ESORICS 2014*. Cham, Switzerland: Springer, 2014, pp. 313–326.

[48] P. Wuille. (2014). *Bip62: Dealing With Malleability*. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0062.mediawiki

[49] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek, "On the malleability of bitcoin transactions," in *Financial Cryptography and Data Security*. Berlin, Germany: Springer, 2015, pp. 1–18, doi: 10.1007/978-3-662-48051-9_1.

[50] J. Tapsell, R. N. Akram, and K. Markantonakis, "An evaluation of the security of the bitcoin peer-to-peer network," in *Proc. IEEE Int. Conf. Internet Things (iThings) IEEE Green Comput. Commun. (GreenCom) IEEE Cyber, Phys. Social Comput. (CPSCom) IEEE Smart Data (SmartData)*, Jul. 2018, pp. 1057–1062.

[51] J. Schnelli. (2016). *Bip151: Peer-to-Peer Communication Encryption*. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0151.mediawiki

[52] N. E. Hastings and P. A. McLean, "TCP/IP spoofing fundamentals," in *Proc. Conf. IEEE 15th Annu. Int. Phoenix Conf. Comput. Commun.*, Mar. 1996, pp. 218–224.

[53] F. Gont and S. Bellovin, *Defending Against Sequence Number Attacks*, document RFC 6528, Internet Engineering Task Force (IETF), Feb. 2012. [Online]. Available: https://www.hjp.at/doc/rfc/rfc6528.html

[54] P. Vixie, "DNS and BIND security issues," in *Proc. Usenix Secur. Symp.*, 1995, pp. 1–9. [Online]. Available: https://www.usenix.org/legacy/publications/library/proceedings/security%95/full_papers/vixie.pdf

[55] Wiz. (2020). *Issue 19714: Enable DNSSEC on All Bitcoin DNS Seed Domain Names*. [Online]. Available: https://github.com/bitcoin/bitcoin/issues/19714

[56] Y. Rekhter. (1994). *A Border Gateway Protocol 4 (BGP-4)*. [Online]. Available: http://www.hjp.at/doc/rfc/rfc4271.html

[57] A. Pilosov and T. Kapela, "Stealing the internet: An internet-scale man in the middle attack," in *Proc. NANOG*, Oct. 2008, pp. 12–15. [Online]. Available: https://defcon.org/images/defcon-16/dc16-presentations/defcon-16-piloso%v-kapela.pdf

[58] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking bitcoin: Routing attacks on cryptocurrencies," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 375–392.

[59] M. Saad, V. Cook, L. Nguyen, M. T. Thai, and A. Mohaisen, "Partitioning attacks on bitcoin: Colliding space, time, and logic," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2019, pp. 1175–1187.

[60] A. Biryukov, D. Khovratovich, and I. Pustogarov, "Deanonymisation of clients in bitcoin P2P network," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2014, pp. 15–29.

[61] A. Biryukov and I. Pustogarov, "Bitcoin over tor isn't a good idea," in *Proc. IEEE Symp. Secur. Privacy*, May 2015, pp. 122–134.

[62] J. R. Douceur, "The Sybil attack," in *Peer-to- Peer Systems*. Berlin, Germany: Springer, 2002, pp. 251–260, doi: 10.1007/3-540-45748-8_24.

[63] T. Neudecker and H. Hartenstein, "Network layer aspects of permissionless blockchains," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 838–857, 1st Quart., 2018.

[64] P. Wuille. (Oct. 2021). *Pr 23306: Make Addrman Support Multiple Ports Per IP*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/23306

[65] F. Franzoni, X. Salleras, and V. Daza, "AToM: Active topology monitoring for the bitcoin peer-to-peer network," *Peer Peer Netw. Appl.*, vol. 15, no. 1, pp. 408–425, Jan. 2022.

[66] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *Financial Cryptography and Data Security*. Berlin, Germany: Springer, 2014, pp. 436–454, doi: 10.1007/978-3-662-45472-5_28.

[67] K. Nayak, S. Kumar, A. Miller, and E. Shi, "Stubborn mining: Generalizing selfish mining and combining with an eclipse attack," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Mar. 2016, pp. 305–320.

[68] M. Lei. (2015). *Exploiting Bitcoin's Topology for Double-Spend Attacks*. [Online]. Available: https://pub.tik.ee.ethz.ch/students/2015-FS/BA-2015-10.pdf

[69] J. Nick. (2015). *Guessing Bitcoin's P2P Connections*. Accessed: Jan. 6, 2021. [Online]. Available: https://jonasnick.github.io/blog/2015/03/06/guessing-bitcoins-p2p-conne%ctions/

[70] G. C. Fanti and P. Viswanath, "Anonymity properties of the bitcoin P2P network," 2017, *arXiv:1703.08761*.

[71] P. Wuille. (Mar. 2015). *Pr 5860: Reduce Fingerprinting Through Timestamps in 'Addr' Messages*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/5860

[72] G. Maxwell. (Sep. 2016). *Pr 8661: Do Not Set an Addr Time Penalty When a Peer Advertises Itself*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/8661

[73] T. Neudecker, P. Andelfinger, and H. Hartenstein, "Timing analysis for inferring the topology of the bitcoin peer-to-peer network," in *IEEE UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld*, Jul. 2016, pp. 358–367.

[74] M. Grundmann, T. Neudecker, and H. Hartenstein, "Exploiting transaction accumulation and double spends for topology inference in bitcoin," in *Financial Cryptography and Data Security*. Berlin, Germany: Springer, 2019, pp. 113–126, doi: 10.1007/978-3-662-58820-8_9.

[75] P. Wuille. (Feb. 2018). *Pr 14626: Select Orphan Transaction Uniformly for Eviction*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/14626

[76] G. Naumenko. (Feb. 2019). *Pr 14897: Randomize Getdata(tx) Request Order and Introduce Bias Toward Outbound*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/14897

[77] P. Wuille. (Oct. 2020). *Pr 19988: Overhaul Transaction Request Logic*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/19988

[78] A. Singh, T.-W. Ngan, P. Druschel, and D. S. Wallach, "Eclipse attacks on overlay networks: Threats and defenses," in *Proc. IEEE INFOCOM 25th IEEE Int. Conf. Comput. Commun.*, Apr. 2006, pp. 1–12.

[79] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on Bitcoin's peer-to-peer network," in *Proc. 24th USENIX Secur. Symp. (USENIX Secur.)* Washington, DC, USA: USENIX Association, Aug. 2015, pp. 129–144. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/heilman

[80] P. Wuille. (Apr. 2015). *Pr 5941: Countermeasures Against Eclipse Attacks*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/5941

[81] E. Heilman. (Aug. 2016). *Pr 8282: Feeler Connections to Increase Online Addrs in the Tried Table*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/8282

[82] M. Tran, I. Choi, G. J. Moon, A. V. Vu, and M. S. Kang, "A stealthier partitioning attack against bitcoin Peer-to-Peer network," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 515–530.

[83] G. Naumenko. (Aug. 2019). *Pr 16702: Supplying and Using Asmap to Improve IP Bucketing in Addrman*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/15759

[84] T. Neudecker, P. Andelfinger, and H. Hartenstein, "A simulation model for analysis of attacks on the bitcoin peer-to-peer network," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, May 2015, pp. 1327–1332.

[85] A. Gervais, H. Ritzdorf, G. O. Karame, and S. Capkun, "Tampering with the delivery of blocks and transactions in bitcoin," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2015, pp. 692–705.

[86] A. Boverman. (2011). *Timejacking & Bitcoin*. [Online]. Available: https://culubas.blogspot.com/2011/05/timejacking-bitcoin_802.html

[87] Theymos. (May 2011). *Issue 279: Time Adjustment Limit Should be Reduced to 40*. [Online]. Available: https://github.com/bitcoin/bitcoin/issues/279

[88] Mruddy. (Feb. 2016). *Pr 7573: Add Maxtimeadjustment Command Line Option*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/7573

[89] M. Zumsande. (Nov. 2021). *Pr 23631: Don't Use Timestamps From Inbound Peers for Adjusted Time*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/23631

[90] G. Maxwell. (Dec. 2015). *Pr 7079: Prevent Peer Flooding INV Request Queue (Redux) (Redux)*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/7079

[91] Laanwj. (Apr. 2016). *Pr 7832: Reduce Block Timeout to 10 Minutes*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/7832

[92] M. Walck, K. Wang, and H. S. Kim, "TendrilStaller: Block delay attack in bitcoin," in *Proc. IEEE Int. Conf. Blockchain (Blockchain)*, Jul. 2019, pp. 1–9.

[93] S. Daftuar. (Jun. 2021). *Pr 22147: Protect Last Outbound HB Compact Block Peer*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/22147

[94] J. Garzig. (Jun. 2013). *Issue 2757: Fingerprint Via Weak-Chain Block Submission*. [Online]. Available: https://github.com/bitcoin/bitcoin/issues/2757

[95] A. Holman. (Jan. 2014). *Pr 2910: Don't Store or Send Blocks Forked Before Last Checkpoint*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/2910

[96] P. Wuille. (Feb. 2015). *Pr 5820: Better Fingerprinting Protection for Non-Main-Chain Getdatas*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/5820

[97] P. Wuille. (Apr. 2015). *Pr 5918: Use Equivalent Pow for Non-Main-Chain Requests*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/5918

[98] J. Posen. (Aug. 2017). *Pr 11113: Ignore Getheaders Requests for Very Old Side Blocks*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/11113

[99] I. Pustogarov. (Dec. 2014). *Pr 5442: Ignore Getaddr Messages on Outbound Connections*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/5442

[100] I. D. Mastan and S. Paul, "A new approach to deanonymization of unreachable bitcoin nodes," in *Cryptology and Network Security*. Cham, Switzerland: Springer, 2018, pp. 277–298.

[101] C. L. Schuba, I. V. Krsul, M. G. Kuhn, E. H. Spafford, A. Sundaram, and D. Zamboni, "Analysis of a denial of service attack on TCP," in *Proc. IEEE Symp. Secur. Privacy*, May 1997, pp. 208–223.

[102] B. Fuller and J. Khan. (2018). *CVE-2018-17145: Bitcoin Inventory Out-of-Memory Denial-of-Service Attack*. [Online]. Available: https://invdos.net/paper/CVE-2018-17145.pdf

[103] B. Core. *CVE-2018-17144 Full Disclosure*. Accessed: Feb. 22, 2022. [Online]. Available: https://bitcoincore.org/en/2018/09/20/notice/

[104] K. Baqer, D. Y. Huang, D. McCoy, and N. Weaver, "Stressing out: Bitcoin 'stress testing,'" in *Financial Cryptography and Data Security*. Berlin, Germany: Springer, 2016, pp. 3–18.

[105] Z. Huffman. (Jul. 2015). *Bitcoin Under Denial of Service Attack*. Accessed: Sep. 3, 2021. [Online]. Available: https://themerkle.com/bitcoin-under-denial-of-service-attack/

[106] (Jul. 2015). *80,000 Unconfirmed Transactions Right Now*. [Online]. Available: https://www.reddit.com/r/Bitcoin/comments/3ck5z9/80000_unconfirmed_tran%sactions_right_now/

[107] J. Pearson. (Jul. 2015). *Wikileaks is Now a Target in the Massive Spam Attack on Bitcoin*. Accessed: Sep. 3, 2021. [Online]. Available: https://www.vice.com/en/article/ezvw7z/wikileaks-is-now-a-target-in-the%-massive-spam-attack-on-bitcoin

[108] WellsHunter. (Jul. 2015). *14 Hours Without Confirmation*. Accessed: Sep. 3, 2021. [Online]. Available: https://www.reddit.com/r/Bitcoin/comments/3cjcxp/14_hours_without_confi%rmation/

[109] J. Pearson. (Jul. 2015). *The Mystery Behind the Biggest Bitcoin Transaction Ever Made*. Accessed: Sep. 3, 2021. [Online]. Available: https://www.vice.com/en/article/d73b8k/the-mystery-behind-the-biggest-b%itcoin-transaction-ever-made

[110] laanwj. (Oct. 2015). *Pr 6793: Bump Minrelaytxfee Default*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/6793

[111] S. Delgado-Segura, C. Pérez-Solá, G. Navarro-Arribas, and J. Herrera-Joancomarti, "Analysis of the bitcoin UTXO set," in *Financial Cryptography Data Security*. Berlin, Germany: Springer, 2019, pp. 78–91.

[112] (Oct. 2015). *With a 1GB Mempool, 1000 Nodes are Now Down (Compared to Yesterday)*. [Online]. Available: https://www.reddit.com/r/Bitcoin/comments/3ny3tw/with_a_1gb_mempool_100%0_nodes_are_now_down/

[113] F. Memoria. (Nov. 2017). *700 Million Stuck in 115,000 Unconfirmed Bitcoin Transactions*. [Online]. Available: https://www.ccn.com/700-million-stuck-115000-unconfirmed-bitcoin-transa%ctions/

[114] (Oct. 2015). *New Transaction Malleability Attack Wave? Another Stresstest*. [Online]. Available: https://bitcointalk.org/index.php?topic=1198032.msg12579271

[115] B. Visuals. *Transactions Per Second*. Accessed: Sep. 1, 2021. [Online]. Available: https://bitcoinvisuals.com/chain-tx-second

[116] M. Saad, M. T. Thai, and A. Mohaisen, "POSTER: Deterring DDoS attacks on blockchain-based cryptocurrencies through mempool optimization," in *Proc. Asia Conf. Comput. Commun. Secur.*, May 2018, pp. 809–811.

[117] M. Saad, J. Kim, D. Nyang, and D. Mohaisen, "Contra-: Mechanisms for countering spam attacks on blockchain's memory pools," *J. Netw. Comput. Appl.*, vol. 179, Apr. 2021, Art. no. 102971.

[118] G. O. Karame, E. Androulaki, and S. Capkun, "Double-spending fast payments in bitcoin," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2012, pp. 906–917.

[119] G. O. Karame, E. Androulaki, M. Roeschlin, A. Gervais, and S. Capkun, "Misbehavior in bitcoin: A study of double-spending and accountability," *ACM Trans. Inf. Syst. Secur.*, vol. 18, no. 1, pp. 1–32, Jun. 2015.

[120] *Bitcoin Average Confirmation Time*. Accessed: Dec. 14, 2021. [Online]. Available: https://ycharts.com/indicators/bitcoin_average_confirmation_time

[121] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *Proc. IEEE P2P*, Sep. 2013, pp. 1–10.

[122] T. Harding. (Mar. 2014). *Pr 3883: Relay and Alert User to Double Spends*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/3883

[123] J. Garzig. (Jul. 2014). *Issue 4550: Reverted Double Spend Relaying (for Now)*. [Online]. Available: https://github.com/bitcoin/bitcoin/issues/4550

[124] T. Bamert, C. Decker, L. Elsen, R. Wattenhofer, and S. Welten, "Have a snack, pay with bitcoins," in *Proc. IEEE P2P*, Sep. 2013, pp. 1–5.

[125] H. Finney. (2011). *Re: Best Practice for Fast Transaction Acceptance—How High is the Risk*. [Online]. Available: https://bitcointalk.org/index.php?topic=3441.msg48384#msg48384

[126] Vector76. (2011). *Re: Fake Bitcoins*. [Online]. Available: https://bitcointalk.org/index.php?topic=36788.msg463391#msg463391

[127] *Vector76 Double Spend Attack*, Reddit, San Francisco, CA, USA, 2014.

[128] C. Natoli and V. Gramoli, "The balance attack or why forkable blockchains are ill-suited for consortium," in *Proc. 47th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2017, pp. 579–590.

[129] S. Zhang and J.-H. Lee, "Double-spending with a Sybil attack in the bitcoin decentralized network," *IEEE Trans. Ind. Informat.*, vol. 15, no. 10, pp. 5715–5722, Oct. 2019.

[130] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek, "Fair two-party computations via bitcoin deposits," in *Financial Cryptography Data Security*. Berlin, Germany: Springer, 2014, pp. 105–121.

[131] F. Reid and M. Harrigan, "An analysis of anonymity in the bitcoin system," in *Security and Privacy in Social Networks*. New York, NY, USA: Springer, 2013, pp. 197–223, doi: 10.1007/978-1-4614-4139-7_10.

[132] D. Kaminsky. (2011). *Black OPS of TCP/IP*. Black Hat USA. [Online]. Available: https://www.slideshare.net/dakami/black-ops-of-tcpip-2011-black-hat-usa% -2011

[133] D. Shah and T. Zaman, "Rumor centrality: A universal source detector," in *Proc. 12th ACM SIGMETRICS/Perform. Joint Int. Conf. Meas. Modeling Comput. Syst.* New York, NY, USA: Association for Computing Machinery, 2012, pp. 199–210.

[134] P. Koshy, D. Koshy, and P. McDaniel, "An analysis of anonymity in bitcoin using P2P network traffic," in *Financial Cryptography Data Security*. Berlin, Germany: Springer, 2014, pp. 469–485.

[135] T. Neudecker and H. Hartenstein, "Could network information facilitate address clustering in bitcoin?" in *Financial Cryptography Data Security*. Cham, Switzerland: Springer, 2017, pp. 155–169, doi: 10.1007/978-3-319-70278-0_9.

[136] S. B. Venkatakrishnan, G. Fanti, and P. Viswanath, "Dandelion: Redesigning the bitcoin network for anonymity," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 1, no. 1, pp. 22:1–22:34, Jun. 2017.

[137] G. Fanti, S. B. Venkatakrishnan, S. Bakshi, B. Denby, S. Bhargava, A. Miller, and P. Viswanath, "Dandelion++: Lightweight cryptocurrency networking with formal anonymity guarantees," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 2, no. 2, pp. 1–35, Jun. 2018.

[138] B. Denby, A. Miller, G. Fanti, S. Bakshi, S. B. Venkatakrishnan, and P. Viswanath. (2017). *Bip156: Dandelion Privacy Enhancing Routing*. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0156.mediawiki

[139] *What is the Tradeoff Between Privacy and Implementation Complexity of Dandelion (BIP156)*. Accessed: Apr. 3, 2022. [Online]. Available: https://bitcoin.stackexchange.com/questions/81503/what-is-the-tradeoff-%between-privacy-and-implementation-complexity-of-dandelion

[140] I. Pustogarov. (Aug. 2014). *Pr 4723: Add Rotation of Outbound Connections*. [Online]. Available: https://github.com/bitcoin/bitcoin/pull/4723

[141] M. Apostolaki, C. Maire, and L. Vanbever, "Perimeter: A network-layer attack on the anonymity of cryptocurrencies," in *Financial Cryptography Data Security*. Berlin, Germany: Springer, 2021, pp. 147–166.

[142] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Proc. 8th IEEE Int. Conf. Data Mining*, Dec. 2008, pp. 413–422.

[143] I. Eyal, "The Miner's dilemma," in *Proc. IEEE Symp. Secur. Privacy*, May 2015, pp. 89–103.

[144] A. Sapirshtein, Y. Sompolinsky, and A. Zohar, "Optimal selfish mining strategies in bitcoin," in *Financial Cryptography Data Security*. Berlin, Germany: Springer, 2017, pp. 515–532.

[145] (Dec. 2010). *Mining Cartel Attack*. Accessed: Dec. 7, 2021. [Online]. Available: https://bitcointalk.org/index.php?topic=2227.msg30083

[146] L. Bahack, "Theoretical bitcoin attacks with less than half of the computational power (draft)," Tech. Rep., 2013.

[147] S. Solat and M. Potop-Butucaru, "Brief announcement: ZeroBlock: Timestamp-free prevention of block-withholding attack in bitcoin," in *Stabilization, Safety, and Security of Distributed Systems. SSS 2017* (Lecture Notes in Computer Science), vol. 10616, P. Spirakis and P. Tsigas, Eds. Cham, Switzerland: Springer, 2017, doi: 10.1007/978-3-319-69084-1_25.

[148] R. Zhang and B. Preneel, "Publish or perish: A backward-compatible defense against selfish mining in bitcoin," in *Topics in Cryptology—CT-RSA 2017* (Lecture Notes in Computer Science), vol. 10159, H. Handschuh, Eds. Cham, Switzerland: Springer, 2017, doi: 10.1007/978-3-319-52153-4_16.

FEDERICO FRANZONI was born in Rome, Italy, in 1985. He received the B.Sc. degree in information technology and the M.Sc. degree in computer science from the Sapienza University of Rome, Rome, in 2009 and 2015, respectively, and the Ph.D. degree in information technologies and communications from Pompeu Fabra University, Barcelona, Spain, in 2021.

Since 2022, he has been a Research Engineer at QPQ. His research interests include IT security, blockchain and P2P networks, operating systems and virtualization, malware and botnets, and memory forensics.

VANESA DAZA was born in Barcelona, Spain. She received the B.Sc. degree in mathematics from the Universitat de Barcelona, Barcelona, Spain, in 1999, and the Ph.D. degree in mathematics from the Universitat Politècnica de Catalunya, Barcelona, in 2004.

She worked as a Researcher in the industry (Scytl, Spain) as well as an Academia (Rovira i Virgili University, Spain). Among other positions serving at UPF, she chaired the Information and Communication Technologies Department, Pompeu Fabra University. She has coauthored more than 30 papers, including international journals and top conferences of cryptography and cybersecurity. Her research interests include the use of distributed cryptographic techniques to enhance security and privacy to secure emerging technologies, with special emphasis on blockchain technology.

Dr. Daza is an Associate Editor of the international journal IEEE Transactions on Dependable and Secure Computing.