2020 IEEE Symposium on Security and Privacy

A Stealthier Partitioning Attack against Bitcoin Peer-to-Peer Network

Muoi Tran*, Inho Choi*, Gi Jun Moon†⋆, Anh V. Vu‡⋆, Min Suk Kang*
*National University of Singapore, {muoitran, inhochoi, kangms}@comp.nus.edu.sg
†Korea University, shangmoon@korea.ac.kr
‡Japan Advanced Institute of Science and Technology, anhvv@jaist.ac.jp

***Abstract*—Network adversaries, such as malicious transit autonomous systems (ASes), have been shown to be capable of partitioning the Bitcoin's peer-to-peer network via routing-level attacks; e.g., a network adversary exploits a BGP vulnerability and performs a prefix hijacking attack (viz. Apostolaki et al. [3]). Due to the nature of BGP operation, such a hijacking is globally observable and thus enables immediate detection of the attack and the identification of the perpetrator. In this paper, we present a stealthier attack, which we call the EREBUS attack, that partitions the Bitcoin network *without* any routing manipulations, which makes the attack undetectable to control-plane and even to data-plane detectors. The novel aspect of EREBUS is that it makes the adversary AS a natural man-in-the-middle network of all the peer connections of one or more targeted Bitcoin nodes by patiently influencing the targeted nodes' peering decision. We show that affecting the peering decision of a Bitcoin node, which is believed to be infeasible after a series of bug patches against the earlier Eclipse attack [29], is possible for the network adversary that can use abundant network address resources (e.g., spoofing millions of IP addresses in many other ASes) reliably for an extended period of time at a negligible cost. The EREBUS attack is readily available for large ASes, such as Tier-1 and large Tier-2 ASes, against the vast majority of 10K public Bitcoin nodes with only about 520 bit/s of attack traffic rate per targeted Bitcoin node and a modest (e.g., 5–6 weeks) attack execution period. The EREBUS attack can be mounted by nation-state adversaries who would be willing to execute sophisticated attack strategies patiently to compromise cryptocurrencies (e.g., control the consensus, take down a cryptocurrency, censor transactions). As the attack exploits the topological advantage of being a network adversary but not the specific vulnerabilities of Bitcoin core, no quick patches seem to be available. We discuss that some naive solutions (e.g., whitelisting, rate-limiting) are ineffective and third-party proxy solutions may worsen the Bitcoin's centralization problem. We provide some suggested modifications to the Bitcoin core and show that they effectively make the EREBUS attack significantly harder; yet, their non-trivial changes to the Bitcoin's network operation (e.g., peering dynamics, propagation delays) should be examined thoroughly before their wide deployment.**

I. Introduction

The robust consensus among large numbers of untrusted nodes is undoubtedly one of the most important technical underpinnings of cryptocurrencies and it can only be achieved with highly dependable peer-to-peer networks. Particularly, the peer-to-peer network of Bitcoin [35], one of the most successful cryptocurrencies, has been an attractive target of attacks in recent years. Notably, the *Bitcoin hijacking* attack [3] demonstrates that a *network adversary* (e.g., a Tier-1 or Tier-2[1] transit autonomous system (AS)) can manipulate the *inter-domain routes* for a set of selected Bitcoin nodes and ultimately partition the Bitcoin's peer-to-peer network. The attack exploits the well-known BGP prefix hijacking vulnerability [8] to redirect all the peer connections of the selected nodes to the adversary AS, effectively controlling all of their peer communications.

The Bitcoin hijacking attack, however, has a serious drawback due to the nature of BGP operation. That is, the *real identity* of the perpetrator (i.e., the malicious AS) is immediately revealed to the public.[2] This can be a critical disadvantage to large ASes with reputation.

In this paper, we present a stealthier Bitcoin attack, which we call the EREBUS[3] attack, that allows a *network adversary* to control the peer connections of a targeted Bitcoin node (i.e., the same attack capabilities and goals as the Bitcoin hijacking attack [3]) *without* any route manipulation, thus leaving *no* control-plane evidence of attacks. Since the attack uses only data-plane attack messages, it is undetected by any control-plane monitoring systems (e.g., public [39], [47] or private BGP monitors [9]); even if data-plane traces of the attack are captured, the perpetrator can easily deny the execution of the attack. The EREBUS attack is shown to be readily available for Tier-1 or large Tier-2 ISPs to target the vast majority of 10K Bitcoin nodes in the system that accept incoming connections from other peers. Thus, nation-state adversaries, who may control large transit ISPs, are able to mount the EREBUS attack.

Figure 1 illustrates the high-level overview of the EREBUS attack, focusing on how an adversary AS network manages to control all (only two shown here) the peer connections of a targeted victim Bitcoin node. Instead of manipulating the underlying routing protocols, the adversary AS changes the existing outgoing peering connections (see the dashed blue arrows) of a victim node (in AS V) to the new connections (see the solid red arrows) with the Bitcoin nodes (in AS C and D) whose victim-to-node inter-domain paths include the adversary AS M. As a result, the adversary AS is eventually placed on the paths of all the peer-to-peer connections of the victim node. It is worth noting that influencing a remote Bitcoin node's peering decision is believed to be infeasible

⋆This research was done while the authors were working as interns at National University of Singapore.

[1] Note that not all networks are eligible for the Bitcoin hijacking attack because some ASes (e.g., single-homed Tier-3 ASes) cannot launch BGP interception attacks; viz. the analysis on BGP hijacking and interception [8].

[2] As an anecdotal evidence, a Canadian ISP hijacked Bitcoin prefixes in 2014 and faced backlash from the media [45].

[3] EREBUS is ancient Greek for "shadow" or "darkness".

DOI 10.1109/SP40000.2020.00027

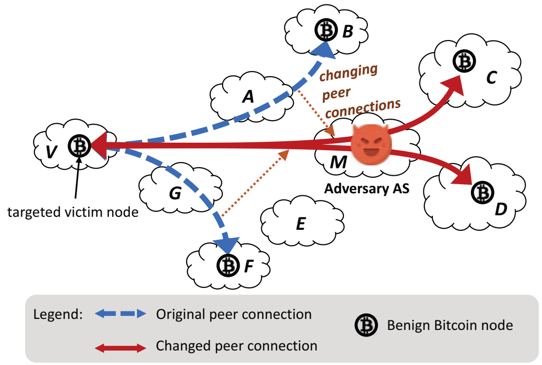


Figure 1: High-level overview of the Erebus attack. A malicious autonomous system (AS) M indirectly switches all the peer connections of a targeted Bitcoin node in the AS V to the other carefully-chosen Bitcoin nodes to locate itself in the middle of all peer connections of the victim node. This attack does *not* require any route manipulation (e.g., hijacking BGP paths), rendering itself undetectable by control-plane anomaly detection systems.

with the up-to-date Bitcoin core (version 0.18.0) [11] because all the related bugs have been fixed after the earlier Eclipse attacks [29]. To be specific, a remote attacker with thousands of bots can no longer force a targeted Bitcoin node to connect to her bots exclusively. We show that, contrary to the common belief, the Erebus network adversary can still affect the peering decision of a Bitcoin node. Our attack is feasible *not* because of any newly discovered bugs in the Bitcoin core implementation but the fundamental *topological advantage* of being a network adversary; that is, our Erebus adversary AS, as a stable man-in-the-middle network, can utilize a large number of network addresses (e.g., impersonating millions or more valid IP addresses) reliably over an extended period of time (e.g., several weeks or more).

Our tests with four-month Bitcoin measurement data show that any Tier-1 or large Tier-2 ASes can control a Bitcoin node with only small bandwidth budget (e.g., 520 bit/s per targeted node) and modest (e.g., 5–6 weeks) attack execution periods, which can be further shortened via an adaptive attack strategy. Note that the Erebus adversary AS can concurrently attack multiple, carefully-chosen Bitcoin nodes (e.g., gateways of popular mining pools) for a strong Bitcoin partitioning attack.

As the Erebus attack does not exploit specific vulnerabilities of the Bitcoin core implementation, no simple, quick patches seem to be available. We first describe naive and ineffective solutions, such as route measurements/simulation for attack detection, or whitelisting or rate-limiting for attack prevention. Then, we discuss third-party proxy based solutions (e.g., SABRE [2]), which may undermine the Bitcoin's philosophy of decentralization. Finally, we suggest several effective countermeasures that require non-trivial changes to the Bitcoin core. We test the effectiveness of the suggested countermeasures and discuss the potential concerns for their wide deployment.

This paper targets Bitcoin as it is one of the most investigated blockchain systems. The Erebus attack can potentially have similar attack effectiveness against many other cryptocurrencies that have peer-to-peer network protocols similar to Bitcoin (e.g., Litecoin, Dash, Zcash). In fact, we found 19 such cryptocurrencies out of the top 50 on the market [17]; see Table I in Appendix A. We leave the evaluation of the Erebus attack on the other cryptocurrencies for future works.

II. Background: Partitioning Bitcoin Nodes

As a background of partitioning attacks against Bitcoin peer-to-peer network, we review the main motivations of these attacks (§II-A). Then, we briefly review the two most notable existing attacks: the Bitcoin hijacking [3] (§II-B) and the Eclipse attack [29] (§II-C).

A. Motivations for Partitioning Bitcoin Network

We list some common objectives for partitioning Bitcoin network:

- *Attacking Bitcoin consensus.* An adversary who partitions a set of miners can waste their mining efforts by stalling the transmission of the latest blockchain state to them. Partitioning a fraction of miners (e.g., 30%) from the rest of the network can make an adversary, who even does not control the majority of the mining power (e.g., 40%), launch the 51% attack [35]. Moreover, the adversary can hijack the computation power of the partitioned miners to mine her blockchain in the selfish mining attacks [19], [24], [43], [36]. Also, the partitioned Bitcoin nodes (e.g., nodes connected to merchants, exchanges or SPV clients) are vulnerable to double-spending attacks [30], even after cautiously waiting for some block confirmations on the blockchain [29].
- *Attacking Bitcoin's off-chain protocols.* Marcus et al. [31] sketch an attack on blockchain layer-two protocols, such as Bitcoin's Lighting Network [38], using the capability of partitioning Bitcoin nodes. The adversary prevents the partitioned victim node from settling the payment channels on the blockchain and steals the victim's funds from her off-chain transactions.
- *Taking down cryptocurrencies.* A powerful adversary, such as a nation-state attacker, may even aim to disrupt a large portion of the underlying peer-to-peer network of a cryptocurrency. At a small scale, the adversary can arbitrarily censor the transactions from the victim.

B. Bitcoin Hijacking and Drawbacks

The Bitcoin hijacking [3] adversary hijacks the traffic toward the most-specific (e.g., up to /24) IP prefixes that include the IP addresses of the targeted Bitcoin nodes using *BGP hijacking* [8]. When the adversary AS hijacks all the inter-domain routes of the nodes, she controls (e.g., inject, drop, modify, delay) the Bitcoin messages of the targeted nodes. Also, the Bitcoin hijacking attack presents several algorithms to find the set of targeted Bitcoin nodes that have to be hijacked together to isolate all of them from the rest of the peer-to-peer network with high probability.

The main drawback. One major drawback of the Bitcoin hijacking attack is that the attacker's identity is revealed to the public immediately (e.g., 5–10 minutes) after the attack is launched. It is well known that BGP hijacking messages for Bitcoin hijacking attacks are propagated to the entire Internet and thus eventually observed by the global route monitoring systems (e.g., RIPE [39], RouteViews [47]); see an AS-level BGP simulation result on this [46]. When the Bitcoin hijacking messages are captured by the route monitoring systems, the perpetrator of an attack is easily identified because its AS number must be included in the hijacking messages.

This is a serious disadvantage to large network adversaries who already have established some reputation in the transit business. Worse yet, if launched by a nation-state attacker (viz., the powerful routing and network capabilities of nation-state attackers [32]), the Bitcoin hijacking attack could create huge undesirable political controversies after the perpetrator is disclosed.

Our BGP measurement study in Appendix B shows that (potentially because of this main drawback) there has been no sign of Bitcoin partitioning attacks in the four-month period in 2018.

C. Eclipse Attacks and Countermeasures

As briefly reviewed earlier, the Eclipse attack [29] directly manipulates the victim Bitcoin node's peer selection decision. A Bitcoin node maintains many connections with other peers via both incoming peer connections (i.e., connections initiated by other peers) and outgoing peer connections (i.e., connections initiated by the node itself). When choosing the outgoing connections, the Bitcoin node chooses the peer IP addresses from its internal database, particularly, the `new` (containing IP addresses learned from its peers) and `tried` (containing IP addresses the node has ever connected to) tables.

Before the Eclipse attack was introduced, the Bitcoin protocol had several vulnerabilities in managing and utilizing the IP addresses in the two tables. Namely, any remote adversaries were able to fill both tables of a targeted node with any arbitrary IP addresses with minimal effort, thus ultimately controlling the peer connection decision of the targeted node.

Deployed countermeasures. The Bitcoin community quickly patched these vulnerabilities. The most notable one among several countermeasures is the ban on direct access to one of the two tables (i.e., `tried` table) from any remote attackers. As a result, an attacker can only directly fill the other table (i.e., `new` table) but not both. Moreover, filling the `new` table has become much harder after the patches. These and several other fixes make the Eclipse attacks infeasible to adversaries with botnets. Interested readers are encouraged to refer to the Eclipse attack paper [29].

III. Overview of the Erebus Attack

In this section, we present an overview of the Erebus attack, beginning with the threat model we consider in this paper (§III-A). We then introduce a *naive* version of the Erebus attack to introduce the main intuition of the attack (§III-B) and then describe its full version (§III-C). Finally, we summarize the three main attack properties that differentiate our attack from previous attacks (§III-D).

A. Threat Model

Similar to the Bitcoin hijacking attack [3], we consider a *network adversary* who has full control of a single AS network, which we call an adversary AS. The adversary may arbitrarily insert/modify/remove/delay any messages traversing her network. Note that typical nation-state adversaries may have such network capability [32]. The adversary's goal is to control *all* the peer connections of a target node in the Bitcoin peer-to-peer network. We target around 10K Bitcoin nodes that accept incoming connections. Bitcoin nodes that do not accept incoming connections due to the lack of public IP interface (e.g., nodes behind network address translations (NATs) or connected via Tor bridges) are out of the scope of our attack.

Moreover, we consider that the targeted Bitcoin nodes have reliable IP addresses during the entire attack execution period. We assume that the main targets of Bitcoin partitioning attacks would be well-known and influential nodes (e.g., gateway nodes of popular mining pools) that would operate reliably with stable IP addresses. Note that even when a targeted node changes its IP address (e.g., DHCP), our adversary can identify the same node with a different IP via the Bitcoin node fingerprinting technique [10].

We assume that a Bitcoin node can be attacked by only one adversary at any given time. When two or more adversaries concurrently attack the same node, the attack success rates may drop due to their competition. We leave the analysis of such attack scenarios for future work.

B. *Naive* Erebus *Attack*

Our network adversary, a malicious AS (e.g., AS M in Figure 1), targets a benign Bitcoin node as a victim (e.g., a node in AS V in Figure 1). Since the adversary may not be on the path of the original peer connections of the victim node (see the dashed blue arrows), the attack goal is to force the victim node to connect to other benign Bitcoin nodes (e.g., nodes in AS C or D) so that the changed peer connections (see the solid red arrows) of the victim node traverse the adversary AS. Note that the new Bitcoin peer nodes for the victim should be chosen carefully so that the victim-to-node route (e.g., V-to-D route) includes the attacker AS M as in Figure 1. The adversary repeats this until it serves *all* the peer connections of the victim node. As a natural man-in-the-middle network of the peer connections of the victim node, the adversary AS now can insert/modify/remove/delay any Bitcoin messages that are delivered to the victim node from the Bitcoin peer nodes of her choice, effectively controlling the victim node; i.e., the attack goal is achieved!

Technical challenges. Though intuitively appealing, implementing the naive Erebus attack in practice is quite challenging due to several limited and unknown attack capabilities. First, there may not exist a sufficient number of benign Bitcoin nodes (such as nodes in AS C or D) whose communication paths to a victim node happen to include the adversary AS. Notice that currently a Bitcoin node may have up to 125 peering connections [11] and, thus, we need 125 or more such Bitcoin peer nodes in the right locations for a reliable attack. Second, even if a sufficient number of such Bitcoin peer nodes exist, it is still hard to influence the peer selection of any Bitcoin node, especially, after several countermeasures have been deployed to thwart the Eclipse attack [29].

C. *Full* EREBUS *Attack*

From the high-level intuition of the naive version, we now describe the full version of the EREBUS attack. Unlike its naive version, the EREBUS adversary uses not only the existing Bitcoin nodes but *any* valid IP addresses whose victim-to-IP-address routes include the adversary AS. For example, the adversary AS M in Figure 1 can enumerate large numbers of valid IP addresses in AS C and D and force the victim node to peer with any of the enumerated IP addresses. We call such adversary-enumerated IP addresses *shadow IPs*. A shadow IP represents a virtual (thus, potentially non-existent) Bitcoin node whose would-be victim-to-node route includes the adversary AS. It is important to note that the shadow IPs are not necessarily used by real Bitcoin nodes or even any host. The shadow IPs are used only to provide a logical view of the peer-to-peer network to the victim and any attempt to connect to them from the victim node is hijacked and completed with normal Bitcoin message exchanges by the adversary AS.

We describe the EREBUS attack in the following two attack phases: a reconnaissance phase (Step I) and an attack execution phase (Step II).

[Step I] Harvesting the shadow IPs (§IV). In this reconnaissance step, the adversary aims to collect as many IP addresses that can be used for the shadow IPs as possible. As illustrated in Figure 2, this step consists of three substeps. In Step I-❶, the adversary evaluates the inter-domain routing state and enumerates all the ASes that may have a node whose victim-to-node packets would traverse its own network M. In Step I-❷, the adversary enumerates all the available IP addresses in the selected ASes and use them for the victim-specific shadow IPs, and inserts them into its database. In Step I-❸, the adversary tests if the packets from the victim node indeed traverse its network towards the chosen shadow IPs. Note that this test step can be integrated into the next attack execution step. We show in Section IV that we can easily obtain tens or hundreds of millions of shadow IPs for a large adversary AS and a typical victim node.

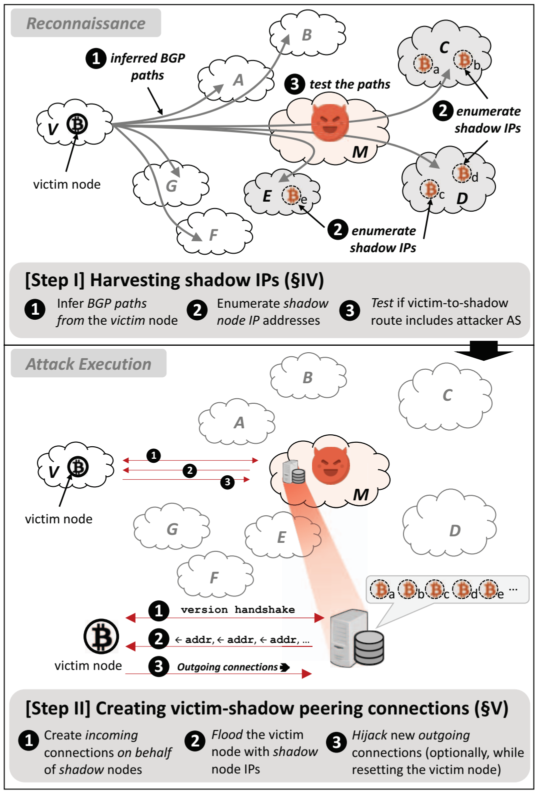

Reconnaissance
1 inferred BGP paths
3 test the paths
2 enumerate shadow IPs
2 enumerate shadow IPs
V
victim node
A
B
C
D
E
F
G
M
a
b
c
d
e
[Step I] Harvesting shadow IPs (§IV)
1 Infer BGP paths from the victim node
2 Enumerate shadow node IP addresses
3 Test if victim-to-shadow route includes attacker AS
Attack Execution
1
2
3
victim node
1 version handshake
2 ← addr, ← addr, ← addr, ...
3 Outgoing connections
...
[Step II] Creating victim-shadow peering connections (§V)
1 Create incoming connections on behalf of shadow nodes
2 Flood the victim node with shadow node IPs
3 Hijack new outgoing connections (optionally, while resetting the victim node)

Figure 2: Two main steps of the EREBUS attack. In the first reconnaissance step, a network adversary collects large numbers of shadow IPs that are used to steer the existing peer connections of the targeted victim Bitcoin node towards. In the second step, the adversary gradually migrates the existing peer connections of the victim node to the shadow IPs by indirectly and patiently influencing the victim node's peer selection algorithm.

[Step II] Creating victim-shadow peering connections (§V). In this attack execution step, the adversary aims to patiently influence the victim node to make peering connections *only* to the shadow IPs it has harvested in Step I. Note that the control of Bitcoin's peering decision with some botnets [29] is no longer possible. Our EREBUS adversary exploits the fundamental advantage of being a network adversary. That is, the adversary AS impersonates the millions or more shadow IPs reliably for several weeks to slowly fill up the internal database of a victim node. In Step II-❶, the adversary creates incoming connections with the victim node *on behalf of* several shadow IPs of its choice. In particular, the adversary initiates the Bitcoin `version` handshake by spoofing shadow IP addresses. Then, in Step II-❷, the adversary floods the victim node's internal IP address tables with a large number of shadow IPs in `addr` messages. After filling the internal tables of the victim node, in Step II-❸, the adversary waits for existing outgoing connections of the victim to be naturally disconnected and replaced by the connections to shadow IPs. To expedite the process, the adversary may trigger the victim node to reset and choose new connections from the internal tables.

D. *Attack Properties*

The EREBUS attack has three properties:

- *Undetectability.* The biggest difference between the Bitcoin hijacking attack [3] and the EREBUS attack is that the former is a control-plane only attack while the latter is a data-plane only attack. The EREBUS adversary does not propagate any control-plane messages (e.g., routing announcements) and thus it is completely invisible to control-plane monitors, such as BGP message collectors and analysis tools (e.g., RIPE [39], RouteView [47], CAIDA BGPStream [37]). As the data-plane traffic is much larger in volume, there exists no public repository of data-plane traffic on the Internet, which makes the public scrutiny for the EREBUS attack nearly impossible. Some cautious and willing ASes may use their deep-packet inspection (DPI) capability to capture all the Bitcoin messages from a suspicious AS and try to identify any suspicious data-plane messages of the EREBUS attack (e.g., Bitcoin messages exchanged in Step II in Figure 2). However, the EREBUS adversary can always deny the execution of attacks because there is no way to identify the actual originator of packets without any accountable Internet architecture; e.g., [1], [48].

- *Immediate availability.* The EREBUS attack is shown to be readily available to *any* Tier-1 ASes against nearly all (99.5%) of the 10K public Bitcoin nodes. Also, many large Tier-2 ASes can target the majority of public Bitcoin nodes in the network; see Section IV-B for our large-scale route evaluations. The network coverage of the adversary ASes matters because the larger the network coverage in the inter-domain topology (e.g., Tier-1 ASes providing connectivity to multiple continents), the more shadow IP addresses are available in general for the EREBUS attack.
- *Lack of trivial countermeasures.* As the EREBUS attack does not exploit any specific protocol vulnerabilities but only the fundamental topological advantage of a network adversary, simple, quick fixes are hard to find. Our investigation in Section VII shows that potentially effective countermeasures are either violating the Bitcoin's philosophy of decentralization (e.g., reliance on third-party proxies) or requiring non-trivial (also not-yet-validated) changes to Bitcoin core.

IV. HARVESTING SHADOW IPS

In the reconnaissance step, an adversary AS enumerates shadow IP addresses for a victim node of its choice. Shadow IPs are *adversary-and-victim specific* and thus each adversary-victim pair has a unique set of shadow IPs that are determined by the topological relationship of the pair (§IV-A). We show that many ASes (e.g., Tier-1 and large Tier-2 ASes) can efficiently enumerate large numbers (e.g., millions or more) of shadow IP addresses (§IV-B). We also show that shadow IPs are geographically well distributed (e.g., shadow IPs are found in all five continents), preventing the victim's connections to shadow IPs from looking suspicious (§IV-C).

A. Enumerating Shadow IPs

The main goal of the attacker in the reconnaissance step is to harvest all available shadow IP addresses. This requires an inter-domain path inference to understand the traffic routes sent from the victim node, the enumeration of all shadow IPs, and an optional verification for those IP addresses.

1) Infer BGP paths from a victim node: The adversary infers the inter-domain routes from the victim node to the rest of the world to see if they include the adversary AS. Notice that the adversary AS is required to be *on* the victim-to-shadow paths but *not* necessarily on shadow-to-victim paths because being on the victim-to-shadow paths is sufficient for impersonating the shadow IPs with IP address spoofing.

To infer the default AS-paths of the traffic routes from the AS hosting the victim node (e.g., AS V in Figure 2), the adversary can simulate the BGP propagation between ASes with a given AS topology and the widely assumed packet forwarding policies of the current Internet; see below in this section. We call the ASes, whose victim-to-themselves paths include the adversary AS, *shadow ASes*; e.g., AS C, D, and E in Figure 2.

2) Collect all shadow IPs: From the inferred shadow ASes, the adversary enumerates all shadow IPs owned by the shadow ASes and stores them into her database.

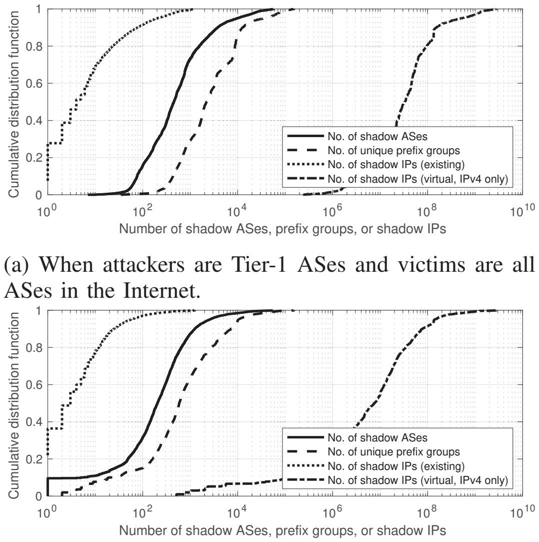
3) Test if shadow node IPs are usable: The collected shadow IPs are inferred results of the BGP route simulations, which may contain some errors. Hence, the attacker may want to verify that the adversary AS is indeed on the traffic routes from the victim node to the shadow node IPs. The adversary can easily test it by establishing a connection (e.g., TCP) with the victim node on behalf of a supposedly shadow IP chosen from a shadow AS. If the adversary does not receive any packet (e.g., SYN/ACK) from the victim, she discards such unusable shadow IPs and their prefixes. This sub-step can be also done during the attack execution phase (see Section V).

B. How Many Shadow IPs are Available?

We evaluate how many shadow IPs are available for various adversary-victim pairs through a comprehensive large-scale BGP route simulations. We also investigate how the shadow IPs are distributed, which is measured by the number of unique prefix *groups* (i.e., the /16 of IPv4 addresses or /32 IPv6 addresses) within the entire shadow IPs pool. The number of prefix groups is important for the EREBUS attack because IPs in the same group can occupy only a small part of the Bitcoin node's peer database.

Evaluation setup. To infer the inter-domain routing between ASes, we start by building the Internet topology of about 60K ASes by using the CAIDA inferred AS business relationship dataset [6], which describes the connectivity among ASes based on their business relationships: provider, customer, peer or sibling. We simulate the propagation of BGP advertisement messages from *all* ASes, allowing all ASes to calculate the default AS paths for the traffic sending to each other. We assume that ASes apply the widely-perceived BGP policies in order [22], [25]: (1) customer links are preferred over peer links and peer links are preferred over provider links; (2) the shortest AS-path length route is preferred; and (3) if multiple best paths exist, an arbitrary method (e.g., AS numbers) is used to break the tie.

We create two sets of attacker-victim ASes pairs for this evaluation. First, we consider the adversaries are Tier-1 ASes and the victim ASes are all ASes on the Internet hosting at least one Bitcoin node [26], making this set contains about 24K of pairs in total. Second, we select the largest 100 ASes



(a) When attackers are Tier-1 ASes and victims are all ASes in the Internet.

(b) When attackers are top-100 ASes and victims are 100 random ASes.

Figure 3: Distributions of the number of shadow ASes, their unique prefix groups (i.e., /16 of IPv4 and /32 of IPv6), and the number of shadow IPs (i.e., legitimate Bitcoin IPs and virtually-created IPs).

in the current Internet ranked by their customer cone size [5] as the attacker ASes, which include all Tier-1 ASes and large Tier-2 ASes. We select 100 random unique ASes from the list of ASes hosting Bitcoin nodes as the victim ASes. We avoid choosing a victim AS that is already chosen as an attacker AS.

Findings. We show in Figure 3 the number of shadow ASes, how diverse their prefixes are, and the number of shadow IPs (including both existing, real Bitcoin IPs and virtual IPs) that can be used for the EREBUS attacks. Figure 3a shows the analysis results when the adversary ASes are Tier-1 and victims are in all the ASes in the Internet. In the majority of the cases (e.g., 85%), there exist more than 100 available shadow ASes; see the solid line. Although only a small number of existing, real Bitcoin nodes can be used as shadow IPs (see the dotted line), almost *all* Tier-1 ASes (e.g., in 99.5% of our tested cases) can target victim nodes in *any* AS with more than 100 unique prefix groups distributed in more than a million of shadow IPs (see the dashed and dash-dot lines).

Figure 3b shows the analysis results when the adversary ASes are top-100 ASes and victims are 100 random ASes. The top-100 ASes (which include many Tier-2 ASes) tend to have less resource (e.g., shadow ASes, prefix groups, shadow IPs), compared to the Tier-1 adversary ASes. Yet, in the majority (e.g., 85%) cases, the adversary ASes still can utilize 100 or more unique prefix groups; see the dashed line. Also, more than a million shadow IP addresses are available in 80% of cases; see the dash-dot line.

C. Geographical Distribution of Shadow IPs

In addition to the distribution in the IP address space, we also investigate how shadow ASes are *geographically* distributed. A cautious Bitcoin node may suspect the EREBUS attack if its peers are located in a restricted geographic area; e.g., it would look suspicious if all outgoing connections are made to one geographic region.

In our case study, we consider Amazon (AS 16509) as the AS hosting the victim node, as cloud providers host the majority Bitcoin nodes and would be common targets of this attack [23]. We choose five largest Tier-2 ASes in each of the five continents, i.e., North America, South America, Europe, Asia-Pacific, and Africa, as our adversary ASes.[4] We show the geographic distribution of the shadow ASes of the five scenarios in Figure 4. Shadow ASes seem to be well distributed globally despite the location of the adversary ASes. In particular, in four out of five tested scenarios, the adversary AS has shadow ASes distributed in all five continents. This result suggests that a strategic adversary can carefully select shadow ASes so that the victim's connections look geographically diverse.

V. CREATING VICTIM-SHADOW PEERING CONNECTIONS

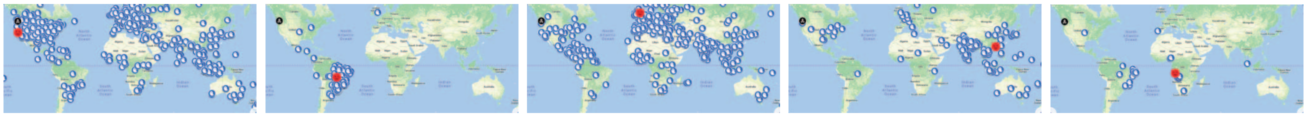
In this section, we describe the attack execution phase, in which the adversary ultimately occupies all the peering connections of a victim node with shadow IP addresses. We begin with a brief overview of the Bitcoin peer-to-peer network according to the most recent (as of June 2019) Bitcoin core v0.18.0 [11], focusing on how a Bitcoin node establishes its incoming and outgoing connections (§V-A). We then present the attack strategies to dominate the two parts of the internal peer database of a victim node with shadow IPs, i.e., the `new` table (§V-B) and the `tried` table (§V-C), which subsequently allow the adversary to occupy all outgoing connections of the victim. We finally describe how an EREBUS adversary occupies the incoming connections of a victim Bitcoin node (§V-D).

A. Bitcoin's Peer Connection Mechanisms

A Bitcoin node with a routable IP address can have several peer connections with other nodes, particularly, up to 8 outgoing peers and 117 incoming peers. Bitcoin nodes accept any incoming connections from other peers with any IP addresses unless the peers have been banned due to sending invalid messages. Outgoing peers, however, are carefully selected from the pool of IP addresses managed by individual Bitcoin nodes. This pool contains IP addresses of other nodes in the network, which Bitcoin nodes learn mainly from the `addr` messages.[5] The learned IP addresses are stored in the hard disks in two tables: a `new` table contains the IPs it has received but yet to connect and a `tried` table contains the IP addresses that it has once successfully made an outgoing connection to. The `new` and `tried` tables have 65,536 and 16,384 slots for IP addresses, respectively. Bitcoin nodes select a random IP

[4] Tier-1 ASes have shadow IPs with much better geographic distribution.

[5] Particularly, each node periodically advertises its IP address via `addr` messages to its peers, which are further relayed to the rest of the network. DNS seeds, which contain a limited number of reliable Bitcoin nodes, can also be another source of IPs when the Bitcoin nodes first join the network.



(a) North America (AS 6939) (b) South America (AS 16735) (c) Europe (AS 1273) (d) Asia-Pacific (AS 4637) (e) Africa (AS 37468)

Figure 4: Maps of *geographic* locations of shadow ASes (blue pins) in five case studies where victim AS (black pin) is Amazon (AS 16509) and the attacker ASes (red pins) are the *largest Tier-2* ASes in five continents.

address from the two tables to make an outgoing connection to until all eight outgoing slots are occupied.

Why is occupying victim's outgoing connections hard? The adversary's goal is to fill the two internal tables of the victim node with shadow IPs as much as possible to maximize the chance of a new outgoing connection made to a shadow IP. This, however, is not trivial, particularly after a series of countermeasures were deployed for Eclipse defense, because IP addresses *cannot* be inserted into the `tried` table directly. The Erebus adversary aims to fill the `new` table first (see Section V-B) and then occupy the `tried` indirectly with a *trickle-down* attack strategy (see Section V-C).

B. *Flooding the* `New` *Table with Shadow IPs*

The Erebus adversary selects a shadow node IP address, say $\mathtt{ip_a}$, from the IPs harvested in the reconnaissance step and makes a peering connection (i.e., a TCP session followed by a `version` handshake) with the victim node on behalf of $\mathtt{ip_a}$ (i.e., the source IP spoofed with $\mathtt{ip_a}$). The replies (e.g., TCP SYN/ACK, Bitcoin `version`/`verack`) from the victim node to $\mathtt{ip_a}$ are captured by the adversary AS because she is on the victim-to-shadow path. The adversary sends `addr` messages, each of which contains up to 1,000 shadow IP addresses, to the victim node and the shadow IPs are inserted into the `new` table of the victim node.

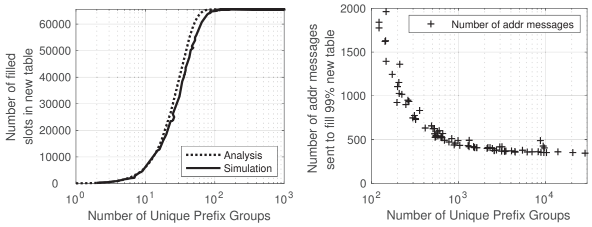
When a Bitcoin node inserts an IP address `ip` to its `new` table, it hashes the IP prefix group (`ip_group`) (i.e., the /16 of IPv4 addresses or /32 IPv6 addresses) and the prefix group of the peer relayed that IP (`peer_group`) to determine the bucket for the IP among 1,024 buckets in total; i.e.,

$$
\begin{aligned}
&\mathtt{h_1\ =\ H(SK,\ ip\_group,\ peer\_group)}\\
&\mathtt{h_2\ =\ H(SK,\ peer\_group,\ h_1\ \%\ 64)}\\
&\mathtt{new\_bucket\ =\ h_2\ \%\ 1024,}
\end{aligned}
$$

where $\mathtt{H}(\cdot)$ is the SHA-256 hash function and `SK` is a secret key of the node. The exact slot for `ip` in the bucket (which contains 64 slots) is determined by hashing the bucket index and the entire IP address; i.e.,

```
new_slot = H(SK, 'N', new_bucket, ip) % 64.
```

If the slot is already occupied, the existing IP address is tested with several checks to consider if it is `terrible` (e.g., the existing IP is more than 30 days old, has failed several connecting attempts). If the existing IP is `terrible`, it is replaced by the new IP that is being inserted; otherwise, the IP being inserted is ignored. Note that IP addresses are stored along with a timestamp. If the IP is already in the `new` table, its timestamp is updated.



(a) (b)

Figure 5: Number of unique prefix groups to fill the `new` table reliably. (a) Number of filled slots in new table versus number of unique prefix groups with results from Equation (1) and Simulation. (b) Number of `addr` messages (each contains 1,000 shadow IPs) sent to fill at least 99% of the `new` table.

Dominating the `new` table with shadow IPs. The adversary repeatedly establishes incoming connections on behalf different shadow IPs and inserts new shadow node IPs into the `new` table at a much higher (e.g., ten times) rate than the rate of incoming non-shadow IPs, which eventually replace most existing IPs in the `new` table.[6] We show that in Section VI that the adversary can easily make the shadow IPs be the vast majority (e.g., 99%) of all the reachable IP addresses in the `new` table in about 30 days.

How many unique prefix groups are needed? The bucket of an inserted shadow IP in the `new` table is determined by its prefix group. Thus, the more unique prefix groups are available in the pool of shadow IPs, the easier to flood all the buckets. We evaluate how many unique prefix groups are enough for the adversary AS to fill all the buckets (and their slots) easily. Suppose that the adversary controls shadow IPs in g unique prefix groups. When sending a shadow IP, says `ip`, to the victim on behalf of a shadow node, says `peer`, the allocated bucket in the `new` table is determined by the prefix groups of two IPs; i.e., (`ip_group`, `peer_group`). The adversary then may have at most g^2 unique pairs of (`ip_group`, `peer_group`). Considering each pair is allocated randomly into a `new` bucket with the probability of $\frac{1}{1024}$ and X is the number of distinct `new` buckets allocated for g^2 pairs, the expected value of X is determined by:

$$E[X] = 1024 \times \left(1 - (1 - \frac{1}{1024})^{g^2}\right). \tag{1}$$

Figure 5a illustrates the relationship between the number of unique prefix groups and the expected number of filled slots of

[6] The attack traffic rate, however, will be only about 1×–2× the rate of normal conditions because the incoming rate of non-shadow IPs is already reduced significantly (e.g., 10%).

a `new` table in Eq. (1) in the dotted line. The solid line denotes the simulation of the IP allocation with the IPs selected from the pool of available shadow IPs in 10K of attack-victim ASes scenarios (enumerated in Section IV). We can see that the analysis result matches well with the simulation result; all the slots in a `new` table can be occupied *reliably* when $g \gtrapprox 100$.

We further test this with an experiment of sending shadow IPs to an actual Bitcoin client running in our lab. As a baseline attack script, we have implemented a rudimentary Bitcoin client that is able to receive and send customized (e.g., source IP spoofing) Bitcoin messages (see Section VI for implementation details). We sample 100 pairs of attacker-victim ASes from the enumerated 10K of pairs (Section IV-B) and use the actual shadow IPs from those pairs in our experiments. In all tests, the victim node is freshly-born with its `new` table initially empty. Our attack script floods the Bitcoin client with `addr` messages with spoofed shadow IPs, each containing 1,000 unique shadow IP addresses; see Appendix C for our efficient shadow IP selection algorithms. Figure 5b illustrates the number of `addr` messages that should be sent in order to have 99% slots of the `new` table occupied by the shadow IPs. Figure 5b shows that, in general, 100 unique prefix groups are sufficient to insert IPs into most of the slots in the `new` table. Moreover, the more diverse the shadow IPs are, the fewer number of IPs are needed. For instance, with 500 or more prefix groups, one can easily occupy most of the `new` table slots with as few as 500 `addr` messages, or 500K shadow IPs. We omit 17 out of 100 cases in which we have less than 100 prefix groups from the Figure 5b because we cannot occupy 99% of the `new` table even after an extremely large number of `addr` messages are sent.

Note that from Section IV we have observed that Tier-1 adversary ASes have at least 100 prefix groups with 99.5% probability. With the above analysis, we confirm that Tier-1 ASes can target nearly all the 10K Bitcoin nodes that accept incoming connections.

C. Trickle-down Migration to Fill the `Tried` *Table*

In the current Bitcoin implementation, the *only* way to insert an IP address into the `tried` table is to move it from the `new` table after a successful outgoing connection made to that IP address. This is to ensure that (1) any remote adversaries cannot directly insert IP addresses into the `tried` table; and (2) the IPs in the `tried` table are likely reachable [33]. Nevertheless, our EREBUS attack indirectly and patiently fills up the `tried` table with the shadow IP addresses by exploiting what we call the *trickle-down effect*.

In particular, there are two scenarios for an IP address to be migrated from `new` table to the `tried` table: (1) an outgoing connection is made to an IP address in the `new` table and the IP address is inserted to the `tried` table; and (2) periodically (every *two* minutes) an IP address is randomly selected from the `new` table and moved to the `tried` table if it is reachable.

First, when a new outgoing connection is made, a Bitcoin node selects either the `new` or `tried` table with equal probability; then, it chooses a random IP address from the selected table. If an IP address is selected from the `new` table and successfully connected, it is moved to the `tried` table while its copies are removed from the `new` table. When inserting the IP to the `tried` table, a Bitcoin node uses the IP's group to determine its bucket and slot indexes; i.e.,

```
h1 = H(SK, ip)
h2 = H(SK, ip_group, h1 % 8)
tried_bucket = h2 % 256
tried_slot = H(SK, 'K', tried_bucket, ip) % 64.
```

Second, a Bitcoin node makes an additional, ephemeral outgoing connection, called a *feeler connection*, every two minutes to test the reachability of a randomly selected IP address from the `new` table [28].[7] If the selected IP from the `new` table is found to be reachable via the feeler connection, it is inserted into the `tried` table. If the IP being inserted collides with an existing IP in the same bucket and slot, the existing one's reachability is tested [27]. If the existing IP is not reachable, the new IP address replaces the existing IP in the `tried` table while the existing IP is inserted back to the `new` table; otherwise, no change is made.

Trickle-down attack strategy. The EREBUS attack first occupies the `new` table slots as much as possible with the shadow IPs and then patiently *waits* for them to be migrated to the `tried` table and ultimately dominate the `tried` table as well. We call this a *trickle-down* attack strategy. We show that in our evaluation in Section VI Tier-1 or large Tier-2 ASes can inject enough numbers of shadow IPs into the `tried` table of a victim node to control all the eight outgoing connections in a few weeks of attack duration.

Adaptive attack strategy. We further propose an optional adaptive attack strategy to speed up the attack execution phase. In the baseline trickle-down attack strategy, an adversary AS would wait until the probability of all the eight outgoing connections made to shadow IPs becomes large enough (e.g., 30% or 50%) and trigger a reboot of the victim node. Note that rebooting a targeted Bitcoin node has been demonstrated with several methods including, but not limited to, denial-of-service or exploiting Bitcoin client's vulnerabilities [29]. In our adaptive attack strategy, an adversary AS keeps tracking of the outgoing connections of the victim node that are *already* made to shadow IPs. When the adversary AS is a large transit AS, the victim node often has some outgoing connections naturally made to shadow IPs even before attacks. Moreover, as the attack progresses, some of existing outgoing connections expire naturally and new outgoing connections can be made to shadow IPs.

The adaptive adversary evaluates if a reboot before the final stage of the attack would be beneficial (e.g., increase the number of outgoing connections to shadow IPs) and triggers a reboot if it would be helpful. We show that this adaptive rebooting strategy indeed can shorten the attack duration to about 40 days only with a couple of more reboots during the attack; see Section VI-D for more details.

[7] To be more specific, a Bitcoin node establishes a feeler connection only when all eight outgoing slots has been occupied and at least *two minutes* has passed since the last feeler connection.

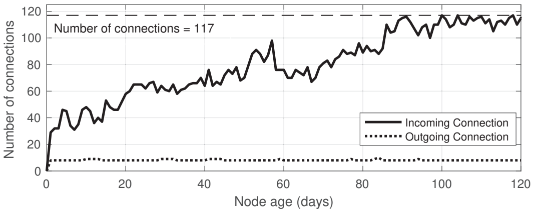


Figure 6: Daily connection snapshots from one of our live nodes. Outgoing slots are almost always fully occupied while incoming connections tend to grow gradually.

D. Occupying Incoming Connections

As long as there exist any unused incoming connection slots, the adversary AS can simply create an incoming connection to the victim node with any of her shadow IPs. Figure 6 shows the number of incoming and outgoing connections established by one of our live Bitcoin nodes. While the node almost always has all eight of its outgoing connections occupied, the number of incoming connections is smaller than the maximum 117 in most cases. Even when all incoming connection slots are full at a moment, the adversary AS can still create an incoming connection because the victim node should evict one existing connection when it has 117 incoming connections, according to a recent change to Bitcoin core [7]. Also, we found that most of the incoming connections are very short-lived (e.g., a couple of minutes), the adversary, thus, can easily occupy most of the incoming connections. For occupying nearly 100% of all the incoming connections, the adversary AS may flood the victim with connection requests from shadow IPs after rebooting the victim node.

VI. Evaluation

We implement the EREBUS attack's execution phase in a Python script and evaluate the attack effectiveness. Our Python attack script is essentially a rudimentary Bitcoin core client that can generate customized Bitcoin messages. We particularly measure the required attack traffic rate and the attack execution duration for various attack configurations.

Attacking a *real* node vs. an *emulated* node. In Section V-B, we evaluate a partial attack functionality (i.e., filling the `new` table) by demonstrating our EREBUS attack script against a real operating Bitcoin client node isolated in a lab. Attacking real nodes, however, is not a feasible option for testing the entire EREBUS attack procedures because each attack may take a few weeks to complete and it is hard to test several attack instances with different configurations and compare them. Worse yet, when attacking real nodes, one cannot test the effectiveness of different combinations of countermeasures in the same condition for a fair comparison. Therefore, we develop and open-source an accurate Bitcoin node emulator [13] that faithfully implements the behaviors of the real Bitcoin nodes, especially IP address management and outgoing connection establishment. For highly accurate experiment results, we use the real Bitcoin `addr` messages collected from our live nodes.

A. Accurate Bitcoin Emulation

Our emulator implements the two components of Bitcoin core version 0.18.0, i.e., the address management and outgoing connections establishment [13]. At a high level, it includes the following Bitcoin nodes' operations: (1) storing IPs into the internal tables; (2) IP allocation; (3) adding IPs to the `new` table; (4) outgoing connections establishment; (5) IPs migration from the `new` to the `tried` table; and (6) feeler connections. For the detailed description of our emulator, we refer the interested readers to Appendix D.

We operate the emulation with the real Bitcoin `addr` messages captured by our live nodes from both incoming and outgoing connections. In particular, we collect Bitcoin messages from six Bitcoin clients version 0.17.0[8] running for 120 days (i.e., from November 18, 2018, to March 18, 2019), in five geographic regions of Amazon EC2 (i.e., US-East, US-West, South America, Europe, and North-Asia) and one in National University of Singapore.

To make the emulation more realistic, the rate of `addr` messages is also adjusted accordingly to the emulated victim's state. Before the attack begins, the emulation is fed with legitimate IPs from DNS seeders and `addr` messages collected via incoming connections. During the attack execution, the number of legitimate `addr` messages received via incoming connection is reduced to 10%.[9] Also, the legitimate `addr` messages received via outgoing connections are set to be proportional to the number of outgoing connections with non-shadow IPs.

To accurately determine whether an IP address is reachable at any given time, our experiments rely on the historical Bitcoin nodes data from Bitnodes [26], which is widely used by existing work [3], [29].

B. Attack Execution

In our attack script, we implement all the attack strategies for creating victim-shadow peering connections (described in Section V). We also implement some additional strategies in the attack script for more realistic attack execution, as follows: (1) the attack script guarantees that shadow IPs are always reachable by replying to all connection attempts from the victim node (i.e., outgoing connections and feeler connections) with the corresponding spoofed source IPs; (2) the script keeps the shadow IPs in the victim's tables fresh by re-advertising a same set of shadow IPs every 30 days; (3) to deploy the adaptive attack strategy, the adversary evaluates the number of outgoing connections it has occupied with the estimated probability of shadow IPs in the two tables. If the actual number of occupied outgoing connections is less than the expected number for a certain threshold (e.g., 2 connections in our experiments), the attack script reboots the victim; and (4) once the estimated probability of all eight

[8] The network implementation of version 0.17.0 is almost identical to the latest version 0.18.0.

[9] Note that 10% is a conservatively chosen value because an adversary in practice can occupy most of the incoming connections and easily reduce the legitimate `addr` message rate much lower than 10%.

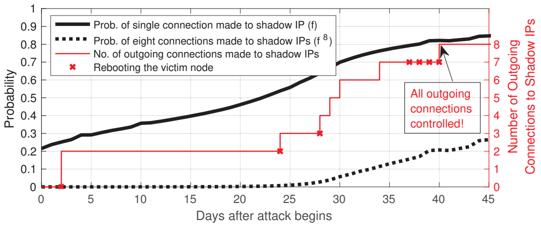


Figure 7: An emulation instance of the adaptive EREBUS attack against a 30-day old node. (Bold lines) Probability of one (f) and all eight (f^8) outgoing connections to be made to shadow IPs. (Normal line) The actual number of outgoing connections under adversary's control.

outgoing connections made to shadow IPs reaches a predefined percentage (e.g., 15%), the attack script reboots the victim every day until all eight outgoing connections are occupied by the adversary.

C. Attack Setup

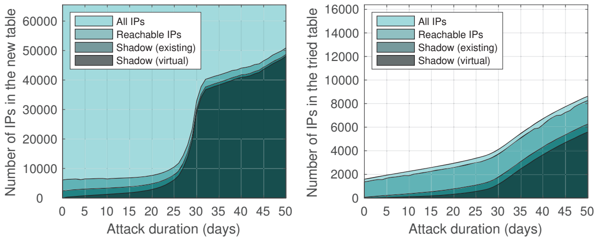
We evaluate the attack effectiveness (e.g., probability of the outgoing connections made to shadow IPs) for varying attack configurations: (1) *adversary's ranking* indicates the index of the adversary among all ASes sorted by their number of available shadow IPs when targeting a specific victim; (2) *victim node's age* is defined as the total lifetime of a victim node since its first operation, and (3) *attack strategy* indicates whether the attack is adaptive or not.

When emulating all three configurations, the victim node is hosted by Amazon (AS 16509). The adversary AS set to Hurricane Electronic (AS 6939), except the experiments with different adversary's rankings, where we select various adversary ASes based on their ranked number of available shadow IPs to be used in the attacks. To analyze the impact of the victim node's age on the attack effectiveness, we run several experiments with the victim's age varying from 10-50 days. In all experiments, the attacks last for 50 days.

D. Experiment Results

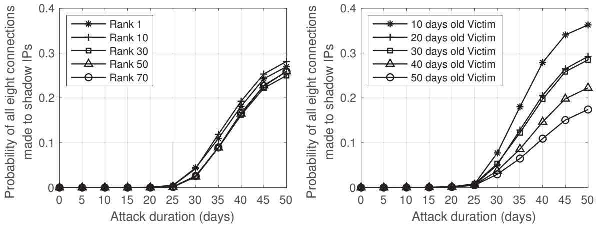
1) Occupying All Outgoing Connections: Figure 7 shows one *instance* of the EREBUS attack, illustrating how the adversary gradually occupies all eight outgoing connections with the adaptive attack strategy. Figure 7 describes the probability of one and eight outgoing connections made to shadow IPs (i.e., f and f^8, respectively) and the actual number of outgoing connections occupied by the adversary. Both probabilities f and f^8 grows as the attack lasts longer; yet, f^8 increases with a much slower rate compared to f and starts acquiring non-negligible probability only after 30 days has passed since the attack starts.

We also observe the number of outgoing connections made to shadow IPs gradually increases and eventually the adversary successfully occupies all victim's outgoing connections on day 40. Moreover, the adversary reboots the victim node only a few times when the adversary expects to occupy more outgoing connections and a couple of more times when the probability of occupying all eight connections reaches 15%.



(a) new table (b) tried table

Figure 8: Dominating the reachable IPs in new and tried tables of a 30 days old node with shadow IPs. Number of shadow IPs surges after attacking for 25–30 days in both tables indicates that at the same time, a huge number of non-shadow IPs become terrible and are evicted.



(a) Adversaries with varying rankings. Victim is hosted at AS 16509. (b) Victim nodes with varying ages. Adversary is AS 6939, victim is hosted at AS 16509.

Figure 9: Probability of making all eight outgoing connections to shadow IPs for varying adversary's rankings and victim's ages. In all experiments, attacks are non-adaptive for fair comparisons.

To better understand the adversary's attack success probability, we illustrate how shadow IPs are inserted into the two tables for the same attack instance in Figure 8. Overall, the number of shadow IPs in both tables tends to increase. Particularly, in the first 25 days of the attack, shadow IPs are inserted with a relatively low rate. After day 25 of the attack, the number of shadow IPs significantly increase as the existing legitimate IPs in the new table become terrible and get evicted; see Figure 8a. Similarly, Figure 8b also shows the number of shadow IPs that are migrated to the tried table increases significantly after attacking for 30 days. Note that although the adversary cannot evict all legitimate IPs to occupy all the slots, shadow IPs still dominates the reachable IPs in both tables, which explains the increasing probability f of a single outgoing connection is made to shadow IP.

2) Attack Effectiveness for Varying Victim's Age and Adversary AS' Ranking: We show the attack effectiveness with various attacker's rankings in Figure 9a and with different victim node's ages in Figure 9b. Figure 9a shows similar growth patterns of f^8 with only minor differences. This suggests that the topological advantages of among ASes even with very different rankings are quite limited. The results from Figure 9b shows that younger victim nodes tend to be more vulnerable to the EREBUS attacks as the adversary's success rates when attacking these nodes are higher. However, the adversary can still achieve a certain success rate (e.g., 18% in our experiments with 50 days old nodes), which requires only a few times of rebooting the victim until all of its outgoing connections are occupied.

3) Required Attack Traffic Rate: We estimate the attack traffic needed in the EREBUS attacks. In 50 days of the attack emulations, our Python script sends shadow IPs to the victim at the rate of 2 IPs per second. This rate is only slightly higher than the IPs inserting rate before the attack starts (about 1.3 IPs per second). Note that the rate of non-shadow IPs is significantly reduced once the attack begins because the attacker occupies most of the incoming connections as well. Thus, the total rate of incoming IPs during the attack is *comparable* to the typical rate of incoming IPs before the attack. Considering every 500 seconds, the adversary needs to send an `addr` message containing 1,000 shadow IPs over a TCP connection made with the victim. The entire connection (i.e., downstream and upstream) includes a TCP handshake (214 bytes), a Bitcoin `version` handshake (614 bytes), an `addr` message (30,093 bytes) and other packets such as TCP ACKs and Bitcoin pings (about 1,500 bytes). Thus, in total, the adversary needs to maintain only the traffic rate of about 520 bit/s to launch the EREBUS attack against one victim node.

4) Attack Scalability: The EREBUS attack is highly scalable. One adversary AS can target multiple Bitcoin nodes at the same time in parallel *without* any extended attack preparation and execution time. This is because an attack execution against each Bitcoin node is independent of each other.

For the concurrent attacks, the adversary AS must send different attack Bitcoin messages to different Bitcoin nodes. This simply requires a linear increase in attack traffic volume as the number of targeted Bitcoin nodes increases. Considering the low attack traffic rate per targeted node, attacking multiple nodes (even tens or hundreds of nodes) still requires negligible total attack traffic rate compared to the multi Tbit/s traffic capacity of large ASes and it can be easily handled by a single or a couple of commodity servers in the adversary AS.

The EREBUS adversary may want to choose the targeted Bitcoin nodes carefully to achieve her ultimate attack goals; e.g., controlling several mining pool gateways, attempting double-spending. Detailed algorithms for determining the set of targeted Bitcoin nodes with adaptive attacker strategies have been discussed by Apostolaki et al. [3].

VII. COUNTERMEASURES

Since the EREBUS attack exploits no design or implementation bugs of Bitcoin core but only the enormous network address resources of network adversaries, the prevention of the attack (e.g., via fixing bugs) is hard. Fortunately, making the EREBUS attack much harder is still possible.

For the sake of argument, we first discuss several naive, ineffective countermeasures that do not work against the EREBUS attack. Then, we discuss a couple of effective countermeasures: the ones that do not change Bitcoin core, and the ones that do require its changes.

A. Ineffective Countermeasures

We list six naive solutions that do *not*, unfortunately, work in practice against the EREBUS attack.

1) Active route measurements: A victim Bitcoin node may actively measure the end-to-end routes from itself to its peers to test whether its connections traverse a common, potentially malicious AS. However, the active route measurement is ineffective because it can be easily spoofed by the EREBUS adversary and the detection of such spoofing is hard. When route probe packets (e.g., `traceroute` probes) traverse through the adversary AS, she can alter the measured routes (e.g., by spoofing IPs of ICMP error messages for `traceroute`) and make the routes look benign (i.e., not traversing the adversary AS). Worse yet, the detection of such spoofing is not straightforward. A victim node may try to detect such spoofing by identifying unusual routes via IP-level route analysis; however, the accuracy of such route manipulation detection is yet unknown and also many false positives may occur whenever router-level paths change.

2) Inter-domain route estimation: A victim node may try to run BGP simulations to estimate the inter-domain routes of its connections (just like the EREBUS adversary does) to detect the EREBUS attack. This approach is, however, unfortunately ineffective because it is hard for a victim node to validate the estimated routes due to the lack of the ground truth route measurement data (see above for why active route measurement is hard). Without the validation of the routes, many false positives and negatives may occur if the estimated routes differ from the real ones. Note that, the EREBUS adversary, in contrast, can accurately validate her route estimation because she sees all the Bitcoin packets from the victim node if she is on the end-to-end routes.

3) Whitelisting IP addresses: One way to prevent the EREBUS attacks would be to maintain a whitelist of real Bitcoin nodes in a central location and inform this to all the Bitcoin nodes so that they can easily ignore shadow IPs. Bitcoin, however, in principle allows any node to join and leave the system at any point in time without any permission. Thus, maintaining such whitelist strongly violates the permissionless blockchain principle. Whitelisting can be centralized or decentralized but either way, it is impractical to operate one. A single (or a small number of) centralized whitelist can be a perfect single point of failure of the system because the compromise of the list can fully control the network graph. A decentralized implementation of a whitelist is not trivial because it requires building consensus on such lists among large numbers of nodes itself.

4) Rate limiting `addr` *messages:* Another candidate solution would be to limit the number of `addr` messages, particularly, the number of IPs received from other Bitcoin nodes to limit the EREBUS adversary's capability. A victim node can enforce a lower limit to the `addr` messages received from incoming connections than the ones from outgoing connections as most of shadow IPs are sent via incoming connections. However, as we observe in our attack emulation in Section VI, the number of IP addresses sent by an attacker

AS is not very different (e.g., only about 1×–2× of the normal rate), making it hard to design a rate limiter in practice.

5) Rate limiting incoming connections: Similarly, one may attempt to limit the number of incoming connections per unit time to make the flooding of new incoming connections to a victim node less effective. However, rate limiting the incoming connections does not decrease the chance of adversary occupying the incoming connections when the adversary's incoming connection requests are indistinguishable from other requests.

6) Network-level data-plane detection: A benign, cautious transit AS may attempt to detect the EREBUS attack launched by a neighbor AS by monitoring any unusually large numbers of Bitcoin messages (e.g., `version`, `verack`, `addr`) to multiple Bitcoin nodes from the single transit AS. However, such data-plane, in-network detection is hard because: (1) attack traffic rate is not significantly higher than the benign cases; and (2) one cannot definitively confirm the originator of any suspicious Bitcoin messages.

B. *Countermeasures* without *Bitcoin Protocol Changes*

To test the reachability of an IP address with an EREBUS-*AS-free* path, a Bitcoin node may rely on third-party proxies (e.g., VPNs, Tor, relays [20], [2]). Although this approach can potentially detect EREBUS attacks without modifying the current Bitcoin protocol, there are a few *caveats* of designing and using such third-party proxy systems:

- *Limited scalability.* It would be challenging if most of Bitcoin nodes want to have their separate proxy-based reachability tests. There exist around 10K potentially vulnerable nodes in the Bitcoin peer-to-peer system. Providing multiple proxies (ideally in different locations) for all the nodes would be difficult in practice; e.g., no single VPN service has that many different VPN nodes (even Tor has less than 1K exit nodes in total). Moreover, the reliance on the external proxies would increase the attack surface; viz., several vulnerabilities of Bitcoin over Tor [10].
- *Centralization.* Due to the limited scalability, any proxy-based approaches could easily end up with a small number of centralized proxies; e.g., a handful of Bitcoin relays, a few popular VPN servers. When a small number of proxies become the centralized components in the peer-to-peer system, they naturally have the power to control the peer-to-peer network topology. Unless we have a highly trustworthy proxy service for the Bitcoin peer-to-peer network, we should critically evaluate the potential risk of centralization in any proxy-based countermeasures.

C. *Countermeasures* with *Bitcoin Protocol Changes*

With some Bitcoin protocol changes, we can make the EREBUS attack much harder. We discuss and evaluate several potential countermeasures.

[C1] Table size reduction. The size of the `new` and `tried` tables is an important system parameter that affects the probability of the adversary-injected IP addresses selected for outgoing connections. The Bitcoin community has *increased* the table sizes four times as the countermeasure against the Eclipse attack [29] because it increases the botnet cost of the Eclipse attack. However, increasing the table sizes, in fact, makes the EREBUS attack much easier; it is the table-size *reduction* that makes the EREBUS attack harder. Figure 10 shows the evaluated probability of all eight outgoing connections to be made to shadow IPs for different countermeasures. As we can see, if we increase the table size, the adversary achieves much higher success probability. The reduction of the table sizes makes the EREBUS attack less effective. Contrary to the Eclipse attackers, an EREBUS adversary has much larger bandwidth capability and significantly more IP addresses she can utilize for attacks.

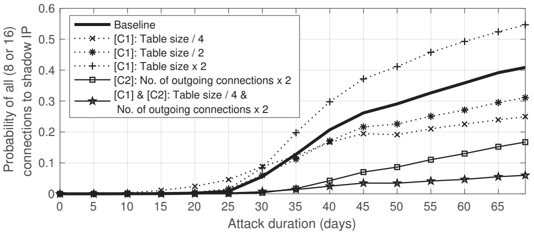


Figure 10: Probability of all outgoing connections made to shadow IPs for varying attack duration (days). In all experiments, attacks are non-adaptive for fair comparisons. We test various countermeasures in Section VII-C. The EREBUS attack becomes harder as the probability decreases.

The actual deployment of this change, however, need much more discussion in the community because the reduction of the table sizes may limit the storage of benign and stable IPs in the Bitcoin peer-to-peer network, which may affect the peering dynamics of the system.

[C2] More outgoing connections. Another effective solution is to increase the number of outgoing connections that can be made in each Bitcoin node. The current maximum is eight and by increasing it we make it harder to occupy all the outgoing connections. As we can see in Figure 10, if we double it, the EREBUS adversary achieves much lower success probability.

Note that the Eclipse attack paper also proposes a similar countermeasure but this has not been adopted in Bitcoin core. The increase of outgoing connections must be carefully designed because this increase may immediately increase the number of peer connections and the total traffic volume of the system. This abrupt volume increase can easily increase the *delay* of messages and blocks in the Bitcoin network.

[C1] and [C2] combined. We combine the two effective Bitcoin protocol modifications and evaluate its final detection effectiveness. The last line in Figure 10 shows the attack success probability when a Bitcoin node reduces the table sizes four times *and* doubles the number of outgoing connections. This clearly shows that the combined approach does make the EREBUS attack extremely difficult; e.g., only 5% success probability even after two months of attack execution.

[C3] Incorporating AS topology in the peer selection. Another highly promising countermeasure is to make the peer selection algorithm *aware* of the AS-level topology so that the peering decision itself makes the EREBUS attack much harder. To be specific, we can use the AS numbers to group IP addresses in the two tables instead of their prefix groups. This way, the attack becomes harder or impossible for the adversaries with IPs distributed in a large number of prefix groups but hosted in a few ASes only. For example, as a result of this defense, about 15% of Tier-1 ASes that do not have at least 100 shadow ASes would not be able to launch the EREBUS attack; see Figure 3 for the distribution of available shadow ASes.

[C4] Eviction policy that protects peers providing fresher block data. A *cross-layer* defense approach is to improve the peer eviction policy so that the Bitcoin node keeps the peers that have propagated more recent block data. As a result, censoring a specific block or transaction from the victim's view becomes less effective if there exists a legitimate incoming connection providing fresher blocks.

Status of our countermeasures. We disclosed our findings with the Bitcoin core security team in early June 2019. As of July 1, 2019, the Bitcoin core developers are positively considering the deployment of countermeasure **[C3]** and **[C4]**. We will keep updating the status of these and any new countermeasures on our public project webpage at https://erebus-attack.comp.nus.edu.sg/.

VIII. Related Work

A. Attacks against Blockchain Peer-to-peer Networks

As we discussed in detail in Section II, the Bitcoin hijacking attack [3] and the Eclipse attack [29] are the closest to our attack. The EREBUS attack shares the same attack capabilities (e.g., being a large AS) and the goal (e.g., hijack all peering connections of a targeted node) with the Bitcoin hijacking attack; yet, the attack strategies are vastly different as the EREBUS is a data-plane attack whereas the Bitcoin hijacking is a control-plane attack. While the EREBUS has some similarity to (and is partly inspired by) the Eclipse attack (e.g., inserting non-existing Bitcoin IP addresses into the internal tables of a targeted node for peering connection hijacking), the EREBUS attack does not exploit the vulnerabilities used by the Eclipse attack but only its topological advantage (which has not been demonstrated before) to hijack the connections. The EREBUS's strategies (e.g., controlling a huge set of IPs persistently for weeks) are in fact unique and dissimilar to those of the Eclipse attack with botnets.

Biryukov et al. [10] proposed a network attack specifically targeting Bitcoin nodes connecting through Tor. The attack exploits the Bitcoin's denial-of-service (DoS) prevention mechanism to force all connections through the adversary-controlled Tor exit nodes. Recently, an Eclipse attack has been demonstrated against the Ethereum peer-to-peer network [31]. For more comprehensive information about Bitcoin attacks, we refer readers to a recent survey by Bonneau et al. [15].

B. Proxies for Enhancing Bitcoin Peer-to-peer Network

In Bitcoin, to enhance the block propagation speed, several relays networks have been developed [20], [21]. A recent study introduces SABRE [2], a secure Bitcoin relay network designed to prevent the Bitcoin hijacking attack [3]. SABRE provides an additional, reliable channel for Bitcoin nodes to connect and get the mined blocks and transactions even when their network prefixes are BGP hijacked. SABRE can potentially mitigate the EREBUS attack as well. However, there is one crucial concern that should be addressed before its wide adoption. That is, it requires a *blind trust* on a few SABRE relays. This approach does not solve the root problem of the network adversary based attacks because SABRE relays can be malicious and launch hijacking attacks in a similar way. Worse yet, malicious ASes can simply provide the SABRE relay services for easier hijacking attacks.

Compared to such third-party proxy-based solutions, our countermeasures in Section VII-C are much desirable as they do not require any trust on external third parties and do not create any centralization in the Bitcoin network.

C. Studies on Blockchain Peer-to-peer Networks

Studying the underlying peer-to-peer networks of popular blockchains has been an interesting research area in recent years. Through a large-scale measurement study on Bitcoin and Ethereum, Gencer et al. show that the two networks are not decentralized in terms of mining power across network resources and the network topology [23]. Apostolaki et al. also show that only 13 ASes hosted 30% of Bitcoin nodes in 2017 [3]. A recent study found that the majority of Bitcoin nodes usually do not have the latest copy of the blockchain, making them naturally vulnerable to partitioning attacks [42].

Several studies also have tried to discover the topology of the Bitcoin's peer-to-peer network. Miller et al. present AddressProbe [34], which uses timestamps to explore peer-to-peer links between Bitcoin nodes. More recently, Delgado et al. introduce TxProbe [18], a technique utilizing orphan transactions to effectively reconstruct the connectivity structure of the Bitcoin network.

IX. Conclusion

Whether it is a multi-billion dollar cryptocurrency or a toy blockchain, as long as they are permissionless, their distributed peer nodes heavily rely on the current Internet, where several large autonomous systems (ASes) can mount dangerous attacks. This paper shows that the topological advantage of these large ASes allows them to control the peer connections of a blockchain if its peer-to-peer protocol is not carefully designed with the EREBUS attack in mind. We hope that our work sparks new discussions on hardening thousands of cryptocurrencies even against powerful, sophisticated (e.g., state-sponsored) network adversaries.

Acknowledgments

We thank the anonymous reviewers of this paper and our shepherd Neha Narula for their helpful feedback. We also thank Adrian Perrig and Aziz Mohaisen for useful comments on an early version of the paper. We thank Matt Corallo and other Bitcoin core developers for the discussion on countermeasures [C3] and [C4]. This research is supported by the CRYSTAL Centre at National University of Singapore.

References

[1] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker, "Accountable Internet Protocol (AIP)," in *ACM SIGCOMM CCR*, 2008.

[2] M. Apostolaki, G. Marti, J. Müller, and L. Vanbever, "SABRE: Protecting Bitcoin against Routing Attacks," in *Proc. NDSS*, 2019.

[3] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking Bitcoin: Routing attacks on cryptocurrencies," in *Proc. IEEE S&P*, 2017.

[4] D. Ardelean, "libBGPdump," https://bitbucket.org/ripencc/bgpdump/wiki/Home, 2019.

[5] "AS Rank: A ranking of the largest Autonomous Systems (AS) in the Internet." http://as-rank.caida.org/, 2019.

[6] "AS Relationships by CAIDA," http://www.caida.org/data/as-relationships/, 2019.

[7] "Attempt To Evict Connection When Incoming Slots are Full," https://github.com/bitcoin/bitcoin/blob/0.17/src/net.cpp#L1128-L1136, 2019.

[8] H. Ballani, P. Francis, and X. Zhang, "A study of prefix hijacking and interception in the Internet," in *ACM SIGCOMM CCR*, 2007.

[9] "BGPmon - Monitoring the Internet," https://bgpmon.net/, 2019.

[10] A. Biryukov and I. Pustogarov, "Bitcoin over Tor isn't a good idea," in *Proc. IEEE S&P*, 2015.

[11] "Bitcoin Core 0.18.0," https://bitcoincore.org/en/releases/0.18.0/, 2019.

[12] "Bitcoin Core integration/staging tree," https://github.com/bitcoin/bitcoin, 2019.

[13] "Bitcoin Emulator," https://github.com/Erebus-Attack/Bitcoin-Emulator, 2019.

[14] "Bitcoin Hijacking Detector," https://github.com/Erebus-Attack/Bitcoin-Hijack-Detector, 2019.

[15] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "Sok: Research perspectives and challenges for bitcoin and cryptocurrencies," in *IEEE S&P*, 2015.

[16] A. Cohen, Y. Gilad, A. Herzberg, and M. Schapira, "Jumpstarting BGP security with path-end validation," in *Proc. ACM SIGCOMM*, 2016.

[17] "Cryptocurrency Market Capitalizations," https://coinmarketcap.com/, 2019.

[18] 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," in *Proc. FC*, 2019.

[19] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," *CACM*, 2018.

[20] "Falcon - A Fast Bitcoin Backbone," https://www.falcon-net.org/, 2019.

[21] "FIBRE - Fast Internet Bitcoin Relay Engine," http://bitcoinfibre.org/, 2019.

[22] L. Gao, "On inferring autonomous system relationships in the Internet," *IEEE/ACM TON*, 2001.

[23] A. E. Gencer, S. Basu, I. Eyal, R. Van Renesse, and E. G. Sirer, "Decentralization in Bitcoin and Ethereum networks," in *Proc. FC*, 2018.

[24] 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 CCS*, 2016.

[25] P. Gill, M. Schapira, and S. Goldberg, "A survey of interdomain routing policies," *ACM SIGCOMM CCR*, 2013.

[26] "Global Bitcoin nodes distribution," https://bitnodes.earn.com/, 2019.

[27] E. Heilman, "net: Add test-before-evict discipline to addrman," https://github.com/bitcoin/bitcoin/pull/9037, 2019.

[28] ——, "net: Feeler connections to increase online addrs in the tried table," https://github.com/bitcoin/bitcoin/pull/8282, 2019.

[29] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse Attacks on Bitcoin's Peer-to-Peer Network," in *Proc. USENIX Security*, 2015.

[30] G. O. Karame, E. Androulaki, and S. Capkun, "Double-spending fast payments in bitcoin," in *Proc. ACM CCS*, 2012.

[31] Y. Marcus, E. Heilman, and S. Goldberg, "Low-Resource Eclipse Attacks on Ethereum's Peer-to-Peer Network," Cryptology ePrint Archive, Report 2018/236, 2018.

[32] B. Marczak, N. Weaver, J. Dalek, R. Ensafi, D. Fifield, S. McKune, A. Rey, J. Scott-Railton, R. Deibert, and V. Paxson, "An Analysis of China's "Great Cannon"," in *Proc. USENIX FOCI*, 2015.

[33] G. Maxwell, "Do not add random inbound peers to addrman," https://github.com/bitcoin/bitcoin/pull/8594, 2019.

[34] A. Miller, J. Litton, A. Pachulski, N. Gupta, D. Levin, N. Spring, and B. Bhattacharjee, "Discovering Bitcoin's Public Topology and Influential Nodes," http://www.cs.umd.edu/projects/coinscope/coinscope.pdf, 2015.

[35] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," https://bitcoin.org/bitcoin.pdf, 2008.

[36] K. Nayak, S. Kumar, A. Miller, and E. Shi, "Stubborn mining: Generalizing selfish mining and combining with an eclipse attack," in *Proc. IEEE EuroS&P*, 2016.

[37] C. Orsini, A. King, D. Giordano, V. Giotsas, and A. Dainotti, "BGPStream: a software framework for live and historical BGP data analysis," in *Proc ACM IMC*, 2016.

[38] J. Poon and T. Dryja, "The Bitcoin Lightning Network: Scalable Off-chain Instant Payments," https://lightning.network/lightning-network-paper.pdf, 2016.

[39] "RIPE Network Coordination Centre," https://www.ripe.net/, 2019.

[40] "RIS Raw Data," https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris/ris-raw-data, 2019.

[41] "Routeviews Prefix to AS mappings Dataset (pfx2as) for IPv4 and IPv6," https://www.caida.org/data/routing/routeviews-prefix2as.xml, 2019.

[42] M. Saad, V. Cook, L. Nguyen, M. T. Thai, and A. Mohaisen, "Partitioning Attacks on Bitcoin: Colliding Space, Time, and Logic," in *Proc. IEEE ICDCS*, 2019.

[43] A. Sapirshtein, Y. Sompolinsky, and A. Zohar, "Optimal Selfish Mining Strategies in Bitcoin," in *Proc. of FC*, 2016.

[44] P. Sermpezis, V. Kotronis, P. Gigis, X. Dimitropoulos, D. Cicalese, A. King, and A. Dainotti, "ARTEMIS: Neutralizing BGP hijacking within a minute," *Proc. IEEE/ACM TON*, 2018.

[45] A. Toonk, "The Canadian Bitcoin Hijack," https://bgpmon.net/the-canadian-bitcoin-hijack/, 2014.

[46] M. Tran, M. S. Kang, H.-C. Hsiao, W.-H. Chiang, S.-P. Tung, and Y.-S. Wang, "On the Feasibility of Rerouting-based DDoS Defenses," in *Proc. IEEE S&P*, 2019.

[47] "University of Oregon Route Views Project," http://www.routeviews.org/routeviews/, 2019.

[48] X. Zhang, H.-C. Hsiao, G. Hasker, H. Chan, A. Perrig, and D. G. Andersen, "SCION: Scalability, control, and isolation on next-generation networks," in *Proc. IEEE S&P*, 2011.

Appendix A
Survey on Cryptocurrencies with Network Implementation Similar to Bitcoin

Bitcoin is the first widely adopted cryptocurrency and its open source implementation [12], perhaps, is one of the most frequently updated systems among more than two thousands of cryptocurrencies existing today. Many other cryptocurrencies follow or even reuse the network design and implementation of Bitcoin.

Table I: List of 19 top-50 cryptocurrencies that have network implementation similar to Bitcoin and are potentially vulnerable to Erebus attacks.

Rank	**Cryptocurrency name**	**No. of `new` buckets**	**No. of `tried` buckets**	**No. of slots per bucket**
1	Bitcoin	1024	256	64
4	Litecoin	1024	256	64
6	Bitcoin Cash	1024	256	64
10	Bitcoin SV	1024	256	64
13	Dash	1024	256	64
18	Zcash	1024	256	64
22	Dogecoin	1024	256	64
23	Bitcoin Gold	1024	256	64
24	Qtum	1024	256	64
27	Digibyte	1024	256	64
29	ABBC Coin	256	64	64
34	Bitcoin Diamond	1024	256	64
35	Komodo	1024	256	64
37	Verge	1024	256	64
41	Ravencoin	1024	256	64
42	Project Pai	1024	256	64
45	Cryptonex	256	64	64
49	HyperCash	1024	64	64
50	Zcoin	1024	256	64

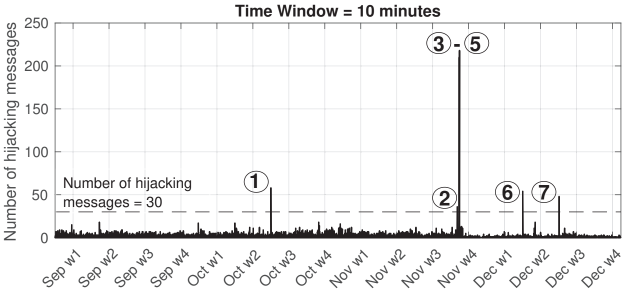


Figure 11: Number of BGP hijacking messages that hijack prefixes hosting Bitcoin nodes in the 4-month period (from September 1, 2018, to December 31, 2018) presented in 10 minutes time window. There are 7 highly suspicious synchronized, large-scale Bitcoin hijacking incidents: a single AS creates BGP hijacking messages to attack 30 prefixes or more in 10 minutes.

We survey the cryptocurrencies that have the network implementation similar to Bitcoin, which thus can potentially be vulnerable to the EREBUS attacks. In particular, we investigate the *source code* of the top 50 cryptocurrencies ranked by their market capitalization (as of March 2019) [17]. Table I shows 19 out of 50 cryptocurrencies have a similar network design and implementation to Bitcoin, making them potential victims of the EREBUS attacks.

APPENDIX B
LONGITUDINAL STUDY ON BITCOIN HIJACKING

We investigate if ASes do launch BGP hijacking attacks for partitioning Bitcoin. We look for synchronized and large-scale Bitcoin partitioning attacks with three conditions in mind: (1) all the BGP hijacking messages are created by a single (potentially malicious) AS; (2) all the messages are observed within a 10-minute time window[10]; and (3) the group has 30 or more unique prefixes, each hosts at least one Bitcoin node IP (see [3] for the conditions for large-scale Bitcoin partitioning attack). We have released our open-source analysis tool [14].

Methodology. Notice that BGP hijacking attempts for partitioning the Bitcoin network are *distinguishable* from generic BGP hijacking attacks because they are targeting network prefixes that include Bitcoin nodes. Our analysis begins with selecting raw BGP update messages for prefixes that host at least one Bitcoin node IP address on the same day. We then detect the BGP hijacking messages and categorize them into two types, following the classification of BGP hijacking attacks [44]: (1) *origin-AS* hijacking, where an AS announce a BGP message claiming the fake ownership of a prefix; and (2) *next-AS* hijacking, where the second last AS in the AS-path announces a seemingly-legitimate BGP message with the actual owner of a prefix as the last AS while in fact having no peering relationship with it. We have observed more complicated hijackings (e.g., when the n^{th} ($n > 2$) peering link from the last AS is fake) as well but their occurrences are negligible compared to the two types, which also has been observed independently in a recent study [16]. Also, our analysis eliminates some false positives, such as the prefixes have multiple origin ASes and the last two ASes are both the actual owners of the prefixes. Finally, we group the BGP hijacking messages into campaigns based on the 10-minute window and further investigate those incidents, e.g., finding their perpetrators, measuring their impacts on the Internet.

Datasets. We analyze in total 48 billion raw BGP update messages collected by 19 RIPE RIS vantage points from September 1, 2018, to December 31, 2018 [40]. We use `bgpdump` [4] to read and store the messages, each includes several useful information for our analysis, such as the timestamp when the message is propagated, the prefix being updated and the AS-path. We also reduce the size of this dataset significantly by removing the redundant messages that update the AS-path for the same prefix in a short period of time (e.g., 120 seconds in our analysis).

For the list of Bitcoin node IPs in each day, we download all Bitcoin network snapshots, which is recorded every 5 minutes for the list of reachable Bitcoin IPs, from Bitnodes via its APIs [26].

The prefix ownership data used in detecting the origin-AS hijacking and the peering relationship used in detecting the next-AS hijacking, however, are usually incomplete due to the lack of ground truth data. We collect the AS-to-prefix data and the AS peering relationship data (including peering through Internet exchanges) from multiple sources and use their superset data. In particular, our AS-to-prefix dataset is merged from Routeviews Prefix-to-AS mapping [41], WHOIS lookup from two domains RADb and NTT, and crawled data from several web portals such as Hurricane Electric Internet Services[11] and ipinfo.io. Similarly, we compose our AS peering dataset from the CAIDA's inferred AS relationships [6], PeeringDB[12], CAIDA's IXPs dataset, and Hurricane Electric's web portal.

Results. We present the number of BGP messages that hijack prefixes hosting Bitcoin nodes in 10 minutes time window in Figure 11. The dashed line indicates the threshold of 30 BGP hijacking messages. Figure 11 shows that BGP hijacking does impact Bitcoin nodes in practice; yet, the number of affected nodes is rather small. Particularly, we observe only seven incidents in the 4-month period in which there are 30 or more Bitcoin nodes are hijacked in 10 minutes, see the numbered spikes.

We conduct an in-depth analysis of the seven highly suspicious Bitcoin hijacking incidents and show our detailed findings in Table II. In all seven cases, the ASes generated the hijacking messages are easily revealed from the hijacking messages. Yet, the most important metric is shown to be the level of global propagation of each incident, specifically, the propagation of all hijacking messages stops after only one or

[10] This time window size is empirically determined to group BGP hijacks with the same purpose.

[11] https://bgp.he.net/

[12] https://www.peeringdb.com/

Table II: Seven highly suspicious cases for potential Bitcoin hijacking attacks occurred from September 1, 2018 to December 31, 2018. Yet, none of them are actual Bitcoin hijacking incidents.

Case	Date of incident	No. of hijacking messages (Origin-AS, Next-AS)	AS number of the perpetrator	No. of hijacked ASes	No. of hops of announcement propagation (avg ± stddev)
1	Oct 17, 2018	57 (0, 57)	14259	3	1.0±0.0
2	Nov 26, 2018	35 (36, 0)	17639	20	1.47±0.74
3	Nov 27, 2018	208 (0, 208)	8928	66	1.0±0.0
4	Nov 27, 2018	214 (0, 214)	8928	65	1.0±0.0
5	Nov 27, 2018	217 (0, 217)	8928	67	1.0±0.0
6	Dec 10, 2018	54 (0, 54)	14259	3	1.0±0.0
7	Dec 18, 2018	47 (0, 47)	14259	2	1.0±0.0

two hops (i.e., they *failed* to be propagated globally). Hence, we conclude that *none* of these seven incidents are in fact carefully crafted Bitcoin hijacking attacks.

Appendix C
Shadow IP Selection Algorithm

Algorithm 1 Select a random shadow IP

```
Require: $\mathcal{P}$: the set of enumerated prefixes.
Ensure: ip: a randomly chosen shadow IP.
 1: procedure SelectRandomShadowIP
 2:     $\mathcal{G} \leftarrow []$                      ▷ Set of unique prefix groups.
 3:     $\mathcal{D}[] \leftarrow []$     ▷ Dictionary of prefixes based on groups.
 4:     for all $p_i \in \mathcal{P}$ do
 5:         $g \leftarrow$ getGroup($p_i$)               ▷ Get prefix's group.
 6:         $\mathcal{G} \leftarrow \mathcal{G} \cup [g]$
 7:         $\mathcal{D}[g] \leftarrow \mathcal{D}[g] \cup [p_i]$
 8:     end for
 9:     group ← getRandomElement($\mathcal{G}$)
10:     prefix ← getRandomElement($\mathcal{D}$[group])
11:     ip ← getRandomElement(prefix)
12:     return ip
13: end procedure
```

We present our shadow IP selection algorithm in Algorithm 1. In general, the harvested shadow prefixes are grouped into a dictionary with the keys are their prefix groups (i.e., /16 for IPv4 addresses and /32 for IPv6 addresses), see Line 4–7. Then, we choose a random prefix group for the selecting shadow IP because we want the groups of multiple randomized shadow IPs to be as much diversity as possible. Based on the selected prefix group, we choose a random prefix with probability proportional with the number of IPs each prefix contains and finally randomly pick a shadow IP from the chosen prefix.

Appendix D
Bitcoin Emulator Operations

Our Bitcoin emulator [13] includes the following Bitcoin nodes' operations:

- *Storing IPs into the internal tables.* We store IPs in a database representing the new and tried tables. Each IP is stored along with its context information and other statistics just as the Bitcoin client does, such as the IP of the peer relayed it, the last timestamp it is advertised, the last timestamp the node attempts to connect to it and so on.
- *IP allocation.* Our emulation implements the deterministic hashing mechanisms of Bitcoin core version 0.18.0 that determine the bucket and slot of an IP address.
- *Adding IPs to the* new *table.* We accurately emulate the procedure of adding an IP into the new table. For example, if one or more copies of the inserted IP already exist in the table, all of their last-heard timestamp in the table are updated. If the inserted IP is allocated to an already occupied slot, our emulation evicts the existing one in favor of the inserted IP if the existing one is terrible. Our emulation checks whether an IP stored in the new table is terrible based on its timestamps and failed attempt counter, e.g., an IP has not been heard in the last 30 days.
- *Outgoing connections establishment.* Our emulation also describes the exact establishment of outgoing connections. Whenever there exists an unoccupied outgoing slot, the node attempts to connect to an IP randomly selected from the new or tried tables. The node can have at most eight outgoing peers and they must be in different prefix groups. If an outgoing peer becomes unreachable according to the Bitnodes database, the outgoing connection to the peer is removed and a new outgoing connection is made.
- *IPs migration from the* new *to the* tried *table.* The emulator migrates the IPs from the new to the tried when they are connected as outgoing peers or via a feeler connection. The emulation also resolves the collision if a migrated IP collides with an existing IP; e.g., it queues the existing one to be tested by a feeler connection.
- *Feeler connections.* Our emulation only establishes feeler connection after all outgoing slots are occupied and at least two minutes have passed since the last feeler connection, as implemented in the Bitcoin client. Our feeler connection implementation tests the colliding IP in the tried table first to resolve IP collisions. If there is no collision, the emulation randomly selects one IP from the new table to test.