Total Eclipse of the Heart – Disrupting the InterPlanetary File System

Bernd Prünster
Insitute of Applied Information Processing and Communications (IAIK), Graz University of Technology

Alexander Marsalek
A-SIT Secure Information Technology Center Austria

Thomas Zefferer
A-SIT Plus GmbH

Abstract

Peer-to-peer networks are an attractive alternative to classical client-server architectures in several fields of application such as voice-over-IP telephony and file sharing. Recently, a new peer-to-peer solution called the *InterPlanetary File System* (IPFS) has attracted attention, with its promise of re-decentralising the Web. Being increasingly used as a standalone application, IPFS has also emerged as the technical backbone of various other decentralised solutions and was even used to evade censorship. Decentralised applications serving millions of users rely on IPFS as one of their crucial building blocks. This popularity also makes IPFS attractive for large-scale attacks. We have identified a conceptual issue in one of IPFS's core libraries and demonstrate its exploitation by means of a successful end-to-end attack. We evaluated this attack against the IPFS reference implementation on the public IPFS network, which is used by the average user to share and consume IPFS content. The results obtained from mounting this attack on live IPFS nodes show that arbitrary IPFS nodes can be eclipsed, i.e. isolated from the network, with moderate effort and limited resources. Compared to similar works, we show that our attack scales well even beyond current network sizes and can disrupt the entire public IPFS network with alarmingly low effort. The vulnerability set described in this paper has been assigned CVE-2020-10937[1]. Responsible disclosure procedures have led to mitigations being deployed. The issues presented in this paper were publicly disclosed together with *Protocol Labs*, the company coordinating the IPFS development in October 2020.

1 Introduction

Modern computer networks typically rely on one of two fundamental architectural models. The client-server model, which is the predominating model in the *World Wide Web* (WWW), clearly distinguishes network nodes into content providers (i.e. servers) and content consumers (i.e. clients). In fields of application, where a strict separation of roles is undesirable, computer networks based on the *peer-to-peer* (P2P) model have gained ground. Entities participating in P2P networks are equal to a large extent, enabling decentralised applications. This, in turn, makes it possible to escape centralised control and governance as illustrated by cryptocurrencies such as *Bitcoin* [19], and systems like *Ethereum*[2], for example.

Recently, a new P2P-based solution called the *InterPlanetary File System* (IPFS) has attracted attention. IPFS defines itself as a "peer-to-peer hypermedia protocol designed to make the web faster, safer, and more open"[3]. Developed by *Protocol Labs*[4], the ambitious goal of IPFS is to re-decentralise the WWW in order to relieve it from the drawbacks of classical client-server-based architectures. To achieve this goal, IPFS replicates and distributes content among participants.

During the past few years, IPFS has increasingly gained traction. *Protocol Labs* reported a 30x growth in network size in 2019 and millions of users every week consuming IPFS content through their HTTP to IPFS gateway [13]. The report also mentions hundreds of thousands of users actively participating in the IPFS core network and hundreds of individual developers contributing every month to the IPFS code base on GitHub.

At the same time, IPFS has also established itself as the technical foundation for various other decentralised applications. For instance, IPFS acts as one of the enabling technologies for *Filecoin* [11], a cryptocurrency developed by Protocol Labs, pitched as a *robust foundation for humanity's information*[5]. Filecoin has had one of the largest ever *initial coin offerings* (ICOs) to date, raising over $205Mio [1]. Amongst others, IPFS also serves as the technical foundation of *DTube*[6], a decentralised video platform with millions of active daily users. Moreover, the cryptocurrency *Ethereum* will be using *libp2p*, a key component of IPFS as the networking layer for the *Ethereum 2.0* network [23]. The growing relevance of IPFS is also underpinned by the fact that the Opera web browser has added native IPFS support on Android [2]

[1] https://cve.mitre.org/cgi-bin/cvename.cgi?name=2020-10937

[2] https://ethereum.org/en/

[3] https://ipfs.io

[4] https://protocol.ai

[5] https://filecoin.io

[6] https://d.tube

recently. Finally, IPFS has also been used to evade censorship. For instance, after being declared illegal in 2017, the Catalan independence movement used IPFS [10]. This increased popularity leads to the general question of the resilience of ungoverned, open P2P systems against attacks inherent to this model.

In contrast to client-server settings, which enable service operators to exert full control and apply a layered approach to system and security architectures, decentralised P2P networks require a different mental model: Simply considering a P2P network as a connectivity layer and taking the security services they provide for granted is an ill-fated strategy. The works of Heilman et al. [7], Marcus et al. [14] and Henningsen et al. [8] aiming at Bitcoin and Ethereum have demonstrated this in the past. Since every participant contributes to the infrastructure of a P2P system, their behaviour on the application layer (and incentives to alter it) has a direct impact on the lower layers of the stack. Systems like IPFS, which foster the deployment of arbitrary applications on top of an abstract network fabric, have even more potential to fall victim to this issue. The IPFS stack therefore provides a feedback loop between application-layer actions and P2P network layer to account for this and enable tighter vertical integration, although we discovered that it still fails to do so in practice.

Contribution and Scope: This paper presents an end-to-end eclipse attack on IPFS, exploiting a conceptual issue in a core component of IPFS that compromises the system's overall security. The attack is based on an abstract model, which allows for attack cost estimations beyond IPFS's current deployment parameters and enables qualitative comparisons with other, related attacks. We evaluated our attack against the IPFS reference implementation, *go-ipfs*, version 0.4.23 (released in January 2020) and the decentralised P2P network spanned by those nodes. In particular, our contribution is twofold:

Attack model We introduce an abstract P2P system model, which allows quantifying the cost of P2P-specific attacks.

Attack We demonstrate how this model applies in practice by introducing the first known end-to-end eclipse attack on IPFS. This attack enables an attacker to single out network nodes of their choice, partition, and disrupt the IPFS network.

Implementation We describe successful mounting of the proposed attack on live IPFS nodes.

Evaluation We elaborate on the threat potential of the attack, concluding that even modestly powerful attackers can carry out the attack to disrupt the whole public IPFS network.

Countermeasures We have reported our findings to Protocol Labs, discussed mitigations with them, which resulted in fixes being rolled out. While the specific attack presented here has since been mitigated, the hardening process is still ongoing, highlighting the sustainable impact of this work on systems used in production. IPFS 0.5 released in May 2020 already includes a major rewrite of a previously vulnerable core component. The 0.6.1 version of IPFS introduced a substantial number of changes that further contribute to attack resiliency and inflate the cost of our attack by several orders of magnitude. The 0.7 version released in September 2020 finally breaks compatibility with older, vulnerable releases.

The issue we uncovered is a conceptual one and thus has an impact beyond IPFS itself. Actually weaponising this weakness requires specific attack vectors, three of which were discovered to affect the public DHT-based IPFS network, which will be referred to as *IPFS DHT* in this paper. This is the network a user will interact with when using the official desktop or command-line IPFS distributions downloadable from `ipfs.io`. A detailed explanation on this terminology is provided as part of Section 4.

At the time of discovering the vulnerability enabling our attack (April 2020), go-ipfs 0.4.23 was the most current release. Due to the modularity of IPFS and the wide use of (parts of) it in other projects, our attack's impact beyond the public DHT-based IPFS network needs to be evaluated on a per-project basis, which is beyond the scope of this work[7]. However, as mitigations have been rolled out, other projects already benefited from the fixes resulting from this work.

Paper Outline This paper is structured as follows. Ethical aspects are laid out in Section 2. Relevant background information is provided in Section 3 to support an in-depth understanding of our attack and its consequences. Details on the attack itself are introduced in Section 4. Subsequently, figures obtained from applying our attack on live IPFS nodes to evaluate the attack's feasibility are presented in Section 5. Finally, we discuss potential countermeasures in Section 6, introduce related scientific work in Section 7, and conclude the paper in Section 8. Appendixes discuss details on implementations flaws, which made our attack possible, and also provide in-depth technical information on the attack procedure.

2 Ethical Considerations and Responsible Disclosure

Mounting attacks on existing solutions used in practice and publishing details on these attacks raises ethical issues. This especially applies to the work presented in this paper, as we have mounted our attack also on the live network for evaluation purposes.

As potential security impacts of the found vulnerabilities were apparent right after its discovery in April 2020, **we im-**

[7] For example, the main attack vectors exploited for attacking the IPFS DHT is not present in Filecoin, according to Protocol Labs.

mediately contacted Protocol Labs, initiating a responsible disclosure process. In the scope of this process, the vulnerability that serves as basis for our attack has been assigned CVE-2020-10937[8]. Furthermore, we have closely aligned follow-up activities with Protocol Labs to prevent negative impacts on the IPFS live network while conducting further research.
We have only attacked nodes operated by ourselves or (with explicit permission) nodes controlled by Protocol Labs, such that impact on regular operations and honest users was prevented. This way, nobody was harmed or put in danger. More precisely, negative impacts have been prevented by the following measures:

- In general, attacks have been run only on self-operated IPFS nodes to avoid negative effects on third-party nodes.
- By attacking a single self-operated node at a time, connectivity to this node was impaired, rendering only this specific node invisible to the rest of the network. This has no practical side effects, due to the inherent redundancy of the network.
- When evaluating our attack on bootstrap nodes run by Protocol Labs (see Section 5), only one was being attacked. However, four out of eight nodes in total are used for bootstrapping as of IPFS 0.5. Hence, running our attack on one node provided tangible results without causing adverse effects.

Protocol Labs actively supported our research, e.g. by monitoring bootstrap nodes under attack and providing direct monitoring of these nodes under attack. *As this process was monitored live by Protocol Labs, it could instantly be stopped should anything go wrong causing (at worst) minutes of degraded service quality.* Overall, the team at Protocol Labs acted professionally, actively supported us in evaluating our attack against core IPFS infrastructure, and invited further research based on our discoveries. Public disclosure of the found vulnerability has been coordinated with Protocol Labs for October 2020 which included publishing an eprint version of this paper's original submission on arXiv.org and a blog post on the official IPFS blog. At this point in time, a hardened version of IPFS had been available for several months, which includes mitigations and substantially raises the bar for exploiting the vulnerability described in this work.

3 Preliminaries

This section provides the necessary background information to ensure a comprehensive understanding of the attack described in this paper. Section 3.1 thus provides a technical overview on IPFS. In-depth technical details are introduced in Section 3.2, which focuses on the library *libp2p* implementing key functionality of IPFS and representing the core target of our attack. Finally, Section 3.3 provides background information on known attack vectors for P2P networks, which have inspired our attack.

3.1 IPFS

From a technical perspective, IPFS is a distributed, content-addressed file system, in which data is not identified by name or path, but by its hash. IPFS stores all data in a decentralised way overlaying the whole network with a Merkle *directed acyclic graph* (DAG) [17] to create a navigable structure. All content stored in the network and every node participating in the network are assigned a unique identifier from the same flat *identifier* (ID) space. Content IDs are derived directly from the respective data by computing the data's cryptographic hash value. Peers, i.e. nodes participating in the network, generate an asymmetric cryptographic key pair. The public key serves as unique ID for the peer. The private key is used by the peer to sign outgoing data in order to provide receiving nodes evidence on its identity. In summary, IPFS builds on the concepts of the Self-certifying File System introduced by Mazières [16] and uses public-key cryptography for the self-certification of objects.

In the absence any central authorities, secure and reliable content and peer discovery is a key challenge. IPFS implements this functionality based on a *Kademlia distributed hash table* (DHT) [3]. Accordingly, each node maintains its own routing table containing information about neighbouring nodes. This information is structured as binary tree containing mappings of node ID to network (IP) addresses. To find a certain node or specific data, the node traverses its own tree for the required ID. If it is able to discover the required ID, the node can access the associated information using the assigned network address. Otherwise, the node asks peers that are closest to the sought-after information. As IPFS uses *Kademlia*, the closest nodes can be found by means of its *exclusive OR* (XOR) distance. This last step, i.e. finding the closest nodes, can be repeated until the required node or information is found. This approach is proven to be efficient, taking only $O(\log_2(\textit{network-size}))$ many requests to locate any content or node. This efficiency comes at the price of limiting the amount of routing information that can be locally stored. To compensate for this, IPFS features a data structure called the *swarm*, which essentially keeps connections beyond the DHT. This is used for the main content-distribution functionality of IPFS: First, the swarm is asked for data. If someone in the swarm is able to provide it, discovering data is a constant-time operation, if not, the DHT is used[9].

IPFS has been designed to be fully open and decentralised,

[8] https://cve.mitre.org/cgi-bin/cvename.cgi?name=2020-10937

[9] https://github.com/ipfs/go-bitswap/blob/master/docs/how-bitswap-works.md

and is thus not guarded by a central authority. As a result, anyone can join the network and identify themselves using a generated asymmetric cryptographic key pair. This obvious strength of IPFS is also one of its Achilles heels, facilitating the attack proposed in this paper, as presented in Section 3.3.

3.2 libp2p

The library *libp2p*[10] is a stand-alone project that was originally an integrated part of IPFS, but has since been externalised. It encompasses a DHT, transport abstractions and other components required to build decentralised applications. In essence, libp2p serves as basis for various solutions that require a P2P network. Moreover, it supports connecting through *network address translators* (NATs) and to some extent also supports browser-based environments using WebSockets [6]. IPFS is one of many solutions that heavily rely on *libp2p* and its P2P functionality.

The attack described in this paper employs a set of vulnerabilities in *libp2p*. Accordingly, this is attack on *libp2p* rather than on IPFS. However, we also exploit the way IPFS interacts with *libp2p* to increase attack efficiency. Moreover, IPFS is the largest public libp2p-based network, which also serves as infrastructure layer for other decentralised services. Since we mounted our attack on the IPFS reference implementation, we refer to IPFS throughout this paper, although the attack actually targets *libp2p*.

DHT As mentioned above, libp2p's DHT is based on *Kademlia* [15]. As of April 2020, libp2p mainly used connection-oriented transport protocols like TCP. Consequently, nodes are only kept in the local routing table as long as an active network link to this node exists. The DHT's binary tree structure allows for a configurable number of nodes to occupy each leaf of this tree. Leaves are referred to as *k-buckets* or simply *buckets*. The bucket-size parameter is called k and is set to 20 in IPFS.

Tree branches can be merged and split on-demand in case buckets are not fully occupied or become overfull. However, the total number of nodes that can be kept in a local routing table is limited according to Eq. (1). Currently, libp2p uses SHA2-256 as cryptographic hash function to derive node and content identifiers for DHT routing, which yields a 256-bit ID space. Note that Kademlia proposes a least-recently seen eviction strategy, should a bucket become overfull. When using connection-oriented transport protocols, this is implemented implicitly using transport-layer keep-alive messages. Thus, newly connecting peers will only replace others, in case those others become unresponsive.

$$RT\ size = \text{bits}(ID\ space) \times k \tag{1}$$

Another key feature provided by the DHT's peer discovery functionality, is *bootstrapping*: In order to initially join the network, the IP address of at least one node already participating in the network needs to be known. Once connected to one such pre-known node, a newly joining peer queries this *bootstrap* node's routing table for their own ID. This prompts a response containing the IDs and IP addresses of other nodes known to the bootstrap node, which are closest to the newly joining peer. As of version 0.4.23, IPFS comes pre-configured with eight bootstrap nodes run by Protocol Labs.

Swarm IPFS raises the requirement to store connection information that exceeds the DHT's limited capacity. For this, IPFS nodes make use of the so-called *swarm*. First and foremost, the *swarm* is the set of all currently active connections and thus a superset of the connections stored in the DHT. IPFS also uses the swarm to speed-up content discovery by initially querying the whole swarm for content, prior to querying the DHT[11]. On its own, the swarm is unbounded, which could lead to resource exhaustion. To prevent this, a component called the *connection manager* or *ConnMgr* is in place, as described below.

ConnMgr The connection manager is provided by libp2p. Its main job is to keep only a sensible number of open connections. This ensures that (a) resource exhaustion is prevented and (b) content discovery and the overall P2P network flow can operate efficiently.

Currently, libp2p features a single implementation of the connection manager. This implementation traverses the set of active connections (i.e. the swarm) once a minute. In the event that more than a configurable threshold of connections are open (referred to as the `highWater` mark), one connection at a time is trimmed, until the second configured threshold (the `lowWater` mark) of open connections is reached[12]. Recently-established connections (within a configurable *grace period*) are exempt from pruning. Starting with the libp2p version that ships with IPFS 0.4.23, these connections do not count towards the total number of active connections. This is crucial, since it prevents a cheap attack where an attacker could connect `highWater` many connections within the grace period to have the ConnMgr unconditionally trim all older connections.

The main challenge with this approach is determining which connections to trim. An abstract scoring system is in place for this purpose, which is available to all components interacting with *libp2p*. Any interacting component is allowed to add a freely-definable tag to any active connection and award points under this tag. The general idea behind this approach is to keep highly useful connections open. For instance, connections in the DHT are awarded points according to Eq. 2. This effectively means that closer (according to their XOR distance) nodes are awarded higher scores and are less likely to be disconnected.

$$score = 5 + \text{commonXORPrefixLen}(remote\ ID, own\ ID) \tag{2}$$

[10] https://libp2p.io

[11] https://github.com/ipfs/go-bitswap/blob/master/docs/how-bitswap-works.md#discovery

[12] Whether these connections are incoming or outgoing is irrelevant

Other sources of points include a relaying subsystem part of libp2p, which is used to help nodes behind firewalls connect to the network: In short, any node can advertise themselves as relay and offer multiplexing other nodes' connections over an already established link between target node and relay. Points can also be awarded by the so-called *Bitswap* subsystem, which is a core component implementing the content-distribution strategy employed by IPFS [4].

While the applied scoring system is essential for the connection manager's functionality, it causes a potential vulnerability: If ways can be found to artificially inflate the score of connections to a node, this node will less likely be disconnected, even if it behaves maliciously. The attack proposed in this paper employs this vulnerability. As we have discovered, *Bitswap* awards points even when receiving unsolicited data blocks. This can easily result in more points than awarded from some of the lower DHT buckets (those containing the nodes that are farthest-away). Given that the swarm is a superset of the DHT, this can lead to DHT connections being trimmed in favour of non-DHT actively advertising data. The relaying subsystem is another source of cheap points, as it awards one point for each relayed connection, regardless of its use to the node. In combination with IPFS's inherent susceptibility to Sybil attacks (see Section 3.3), a node can be manipulated into eclipsing itself from the honest network. As we will show, it is possible to execute such a strategy at an extremely low cost. It is important to note that this is a conceptual flaw with the only variable being the actual resources required to mount an attack.

3.3 Known Attack Strategies

IPFS is a fully open and decentralised solution with no central coordinating or regulating authorities. These properties make IPFS vulnerable to two different attack strategies, which are not specific for IPFS, but apply to any P2P network with comparable properties. The two attack strategies that make use of these vulnerabilities have become known as *Sybil* and *eclipse* attack.

In the *Sybil* attack [5], a single attacker presents itself to the network as many seemingly independent nodes by generating multiple identities. This can subvert any network operation that works under the assumption of interacting with distinct, non-colluding entities, such as the distributed routing protocol itself. Due to its decentralised nature and openness, IPFS is conceptually vulnerable to Sybil attacks.

The second prominent strategy to compromise P2P systems is the *eclipse* attack [22]. In essence, the attacker manipulates the local routing information of a node, such that any request from or to the victim passes through nodes controlled by the attacker. For a structured P2P network, this requires generating identifiers of a specific distance to the chosen victim's identifier. Since node identifiers in IPFS are hashed prior to computing this distance, large numbers of IDs need to be generated and tested to obtain IDs with suitable distances. While this may seem infeasible, we show that a brute-force approach to this problem actually scales well, enabling global attacks even for large network sizes[13] (see below). Once this is done, as many nodes as required to eclipse a victim using these IDs can be operated by applying a Sybil attack.

Any open and fully decentralised P2P network is vulnerable to Sybil and eclipse attacks on a conceptual level. Fortunately, a variety of countermeasures to these generic attack strategies exist in practice, although as of version 0.4.23 (April 2020), IPFS did not include any.

3.4 Abstract P2P Attack Model

The concepts discussed so far can be used to derive a generic model for eclipse and Sybil attacks. Prünster et al. [20] laid out theoretical background on this matter. The experience gained from the end-to-end attack on IPFS and surveying related work allowed us to put their theories to the test, and to expand their work. This has resulted in a generic model, which can be used to gauge the cost of attacking P2P systems. For the sake of clarity, this paper focuses on ways to exploit this model in the context of Kademlia-like designs. The model consists of an attacker running Sybil nodes, which interface with a target in order to eclipse it. Aside from communicating with a target over the Internet, no other interactions are allowed, although this may include attacker-controlled entities other than Sybil nodes. This specifically rules out relying on bugs present in the operating system or any other components not related to the P2P network node implementation under attack. Any flaws in the P2P system's protocols, however, may be exploited as these are considered part of the attack target. In short, the node under attack is assumed to be configured and operated as intended. We have identified the following model parameters:

n network size (= number of nodes)

s_{100} number of Sybil identities, which need to be generated to obtain enough identifiers, to fill a freely-selectable network node's routing table in order to eclipse it. Brute-force generation of identifiers (which follows a uniform distribution) is the only way to generate these identifiers. s_{100} hence refers to the number of identifiers, which need to be generated to reach this goal with 100% probability, including a considerable safety margin.

e number of Sybil nodes, which need to be run to actually fill these routing table spots

$c_{s_{100}}$ cost of Sybil identifier generation

c_e cost of operating Sybil nodes required during an actual attack on a single target

[13] This scales, since only the network size is relevant, not the ID space.

c_a additional attack costs

c_i total cost of attacking an individual node

c_t total cost of attacking the whole system

The first parameters, s_{100} and e mainly depend on system architecture and routing table organisation. Neither take running or up-front costs into account, but serve as foundation required to estimate the actual attack costs. In the case of Kademlia-like designs, the results obtained by Prünster et al. [20] allow for estimating these parameters based on network size (n) and bucket size (k), as shown in Eq. 3, and Eq. 4.

$$s_{100} \approx 2 \times k \times n \tag{3}$$

$$e \approx \lceil k \times \log_2(n) \rceil \tag{4}$$

The cost parameters, $c_{s_{100}}$ and c_e, on the other hand, depend to a great extent on implementation details and attack strategy. Small networks, for example, may allow for all required identifiers to be generated on-the-fly, which will result in $c_{s_{100}}$ incurring for every attack run. In general, however, $c_{s_{100}}$ can be considered up-front costs, since a pool of pre-generated identifiers can be reused for an arbitrary number of attack runs. On the other hand, c_e always corresponds to running costs, since attack nodes need to be operated for as long as it takes to successfully eclipse a target. Without any further actions to speed up this process, c_e is dependent on a network's *churn rate*[14]. As such, it can take days or even weeks to eclipse a single node, which is why attacks on Bitcoin and Ethereum utilised *denial-of-service* (DoS) attacks or relied on victims restarting [7, 14]. These additional actions also incur costs, which is reflected by parameter c_a.

Section 4, and 5 put this model into context by illustrating how we mounted and evaluated an end-to-end eclipse attack on IPFS. As we will show, this model is flexible enough to accommodate for different attack strategies and provides ways to map network parameters to attack costs.

4 Eclipsing IPFS Nodes

If one node after another can be eclipsed from the rest of the network and the amount of resources required to keep nodes from reconnecting are low, even an average-powered attacker can disrupt P2P networks that are as a whole many orders of magnitude more powerful than the attacker. This work demonstrates precisely this kind of attack against the live IPFS DHT. We show how we can advance from attacking single nodes to partitioning the network with negligible running costs. Our implemented end-to-end attack is able to automatically poison any node's routing table on the main IPFS network within minutes, regardless of the network's churn rate and to fully eclipse an average node in less than an hour with $\approx 75\%$ probability (see Section 5). The only input required to start the attack is a target node's identifier, all other parameters can be queried remotely. Moreover, IPFS uses a configurable, but otherwise static set of nodes for bootstrapping. Poisoning the bootstrap node's routing tables (with potentially bogus information) is therefore enough to keep any node that carries out the bootstrap routine from ever interfacing with other legitimate nodes[15]. At this point, partitioning the IPFS DHT becomes possible.

4.1 Naïve Attack Strategy

The IPFS reference implementation, by default, combines Bitswap (which does not reach beyond immediate swarm connections) and DHT-based content routing to distribute data and locate other nodes. The DHT component itself knows two modes of operation: *client* mode, and *full* mode (also referred to as server mode). Only those nodes presenting themselves as operating the full DHT, and advertising themselves as directly reachable on the IP network layer will be added to other nodes' local routing tables. The set of all nodes meeting these requirements are critical for the public IPFS DHT. Disrupting these nodes will also affect client-mode nodes, given that 'In IPFS, the DHT is used as the fundamental component of the content routing system' [21]. A collapsing DHT will have client-mode nodes rely purely on their immediate swarm connections for content discovery (see Section 3.1). As an immediate effect, the long tail of available data will no longer be available. Maintaining a disrupted state, however, will have severe effects: Eclipsing all DHT server-mode peers means that all IP-layer information about honest nodes (which is ultimately used to connect to nodes) will vanish from the DHT. Therefore, it is sufficient to take out server-mode nodes to also affect all clients. Henningsen et al. [9] found on average 44474 concurrently online server nodes for the live IPFS DHT in early 2020. Of these 44474 nodes on average only 6.55% (about 3000 nodes) responded to connection attempts, meaning the remainder of nodes are likely operated by private individuals behind a NAT.

In accordance with the attack model introduced in Section 3.4, a naïve attack strategy would require pre-generating $s_{100} = 2 \times 20 \times 3000$ identifiers and subsequently operating $e = 20 \times \lceil \log_2(3000) \rceil$ many Sybil nodes, which would continuously ping an attack target until they became resident in the lowest $\lceil \log_2(3000) \rceil$ buckets of victim's routing table to replace all honest nodes in order to eclipse it. Our experiments showed that it is possible to operate this many (reduced-functionality) nodes for less than $0.01€/h$ including VAT based on a cloud offer[16]. The time it would take to reach this goal without further actions, however, is impossible to quantify, as it is effectively churn-dependent. Such a wait-and-see approach to attacking is therefore unattractive in

[14] The churn rate is defined as the participant turnover in the network, i.e. how fast participants join and leave the network.

[15] unless measures beyond the default behaviour have been explicitly set up

[16] https://www.hetzner.com/cloud

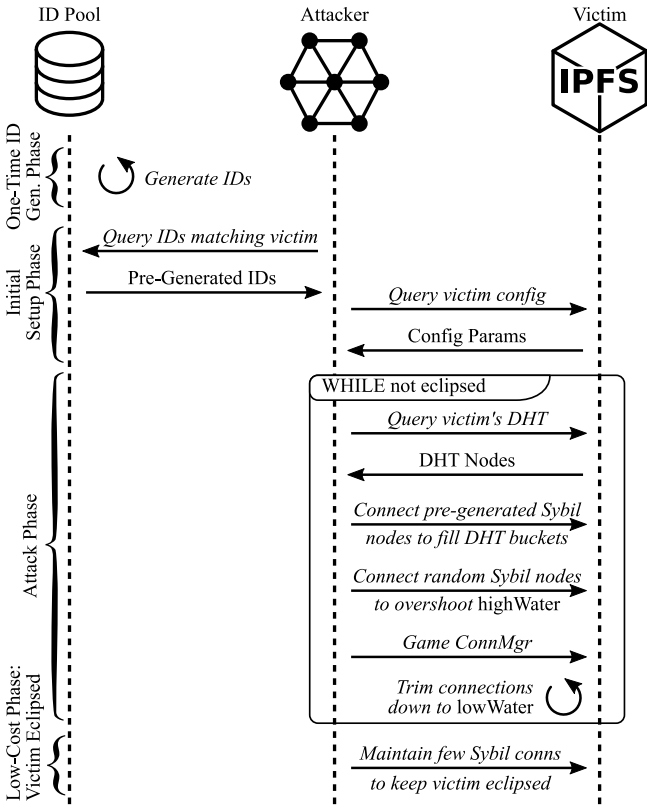


Figure 1: Abstract end-to-end flow of eclipsing an IPFS node

general. ID pre-generation for this network size, on the other hand, incurs such negligible cost, that it's impossible to even quantify.

Instead of following this wait-and-see approach and only taking the theoretical characteristics of Kademlia into account, we have chosen a different attack strategy, which is based on exploiting a variety of weaknesses, resulting in the following attack workflow, which also results in quantifiable attack costs.

4.2 Fully Automated, End-to-End Eclipse Attacks on IPFS

In contrast to the naïve attack strategy, which depends purely on network size and churn, we actively exploit the ConnMgr's behaviour described in Section 3.2. Nodes in the DHT are awarded ConnMgr points in accordance with Eq. 2 (such that closer nodes are scored higher) we do not limit identifier generation to s_{100}, but try to pre-generate as many identifiers as possible to fill far more buckets in an attack target's DHT for additional points. We found a sweet spot at pre-generating $\approx$ 146bn identifiers to fill the lowest 33+ buckets of any node, requiring 29TB of disk space[17]. In addition, we also exploit the fact that unsolicited advertisement of content results in points awarded by the Bitswap subsystem. Doing so does not incur any measurable cost. Moreover, the relaying subsystem is a source of virtually unlimited points, since each relayed connections is awarded one point. Therefore, fake nodes, which provide no functionality can be operated solely to relay their connection in order to inflate the score of other nodes. As we have observed, a single link can support as many as 1000 relayed connections.

We have mounted a fully automated, end-to-end attack on IPFS based on these findings. In accordance with Fig. 1, the attack consists of four phases: ID generation, setup phase, attack phase, and a low-cost phase, which can be maintained, once a victim has been eclipsed.

The actual attack is carried out by a stripped-down libp2p-based node (to further reduce costs) that performs a Sybil attack, which is then used to eclipse the target. This node requires a set of pre-generated IDs (from the ID generation step), that fit the victim's DHT. The main attack loop then consists of the following steps:

1. Establish as many connections to the target as possible, each with one of the pre-generated identifiers, covering the lowest 33+ buckets of the victim.

2. Establish additional connections with randomly generated identifiers to reach a total of `highWater` many active connections. This ensures periodic trimming of connections.

3. Messages are sent over each connection to inflate its score, thus tricking the target's ConnMgr into considering these connections to be more important than those to honest nodes.

4. When the target's ConnMgr trims connections to reach `lowWater` many active connections, only legitimate connections established within the grace period will survive. All others will be pruned, leaving mostly those connections established by our malicious libp2p node, since we previously tricked the target's ConnMgr into considering our connections to be more important than those to honest nodes.

Since we are able to exploit the ConnMgr to our advantage, our connections will gradually fill up the target's routing table (this takes 2 minutes at most) as well as the swarm since honest ones are pruned. When queried for content or other nodes, our attacker nodes will filter out any information on other (legitimate) nodes from responses to hinder the victim from

[17] This covers network sizes far beyond any realistic bounds, awards substantial amounts of points without putting any strain on victim nodes during an attack and scales well during identifier generation. Generating more identifiers, on the other hand, scales exponentially with the number of additional buckets to cover and is therefore not economic.

learning about other nodes. We rely on continuously probing the target's routing table to receive feedback about the attack's progress (which is possible due to the deterministic behaviour of Kademlia). Once successful, only four connections suffice to keep a regular IPFS node eclipsed. This is a hardcoded value—a node with less connections will contact the bootstrap nodes.

In summary, our attack exploits a variety of flaws in libp2p/IPFS, specifically in the uncoordinated coexistence of the ConnMgr, Bitswap and relaying DHT subsystems. This results in nodes actively trimming connections to honest peers after mere minutes, due to the ease of tricking the ConnMgr. As part of engaging in a responsible disclosure process with Protocol Labs, this has been acknowledged as a conceptual problem without an easy solution. More exhaustive information on issues and implementation flaws can be found in Appendix A, while technical details on remotely probing nodes, identifier generation, ConnMgr exploitation are provided in Appendix B.

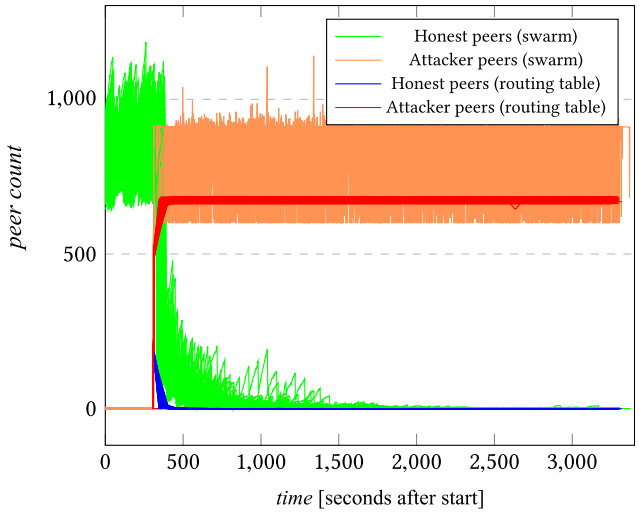
5 Evaluation

In order to gauge the impact of our attack, two key scenarios were evaluated. Attack runs were carried out against unmodified IPFS nodes operated within the live IPFS network. While our main attack target was go-ipfs 0.4.23, version 0.5.0 was also evaluated to asses the impact of countermeasures included in this release (see Section 6). In both cases, the state of an attack target's swarm was queried locally at the target node, while routing table information was obtained remotely (see Appendix B.2) as part of the main attack loop. First, the impact on an average node with default parameters was measured. Subsequently, nodes with the same configuration as the live IPFS bootstrap nodes[18] were attacked. The evaluation results for both scenarios are provided in Fig. 2. In both cases, the DHT is fully poisoned within minutes (Sections 5.1 and 5.2 provide in-depth discussions of attack performance for each scenario). Apart from attack performance evaluation, we also provide an estimate regarding the cost of completely disrupting the whole live IPFS DHT for version 0.4.23. Every attack run was carried out 100 times against newly-spawned IPFS nodes with random IDs we operated ourselves, with permission and support from Protocol Labs when attacking nodes operated by them.

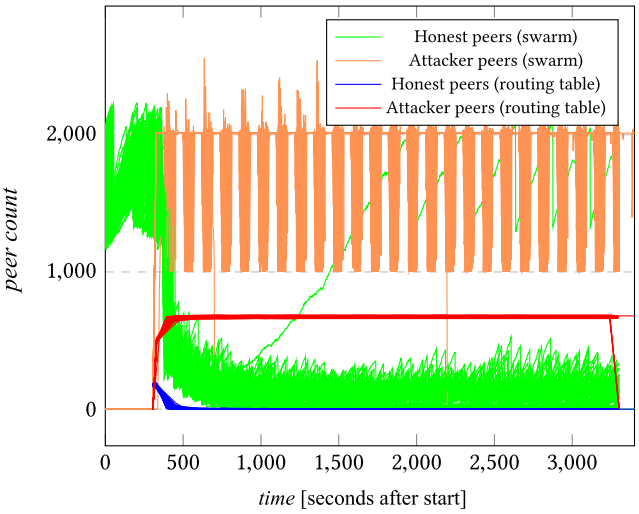
5.1 Default Settings

The goal of this evaluation against an unmodified go-ipfs v0.4.23 node with default ConnMgr parameters (`lowWater` = 600, `highWater` = 900, 20s grace period) is to see whether we can eclipse average nodes in the IPFS network. This setting

[18] These parameters were kindly provided to us by Protocol Labs to help evaluating our attack's impact.



(a) Default settings (`lowWater`=600, `highWater`=900, grace period=20s)



(b) Bootstrap settings (`lowWater`=1000, `highWater`=2000, grace period=60s). Two runs failed due to attack nodes crashing, causing outliers. Note that only the routing table is relevant for an attack bootstrap nodes.

Figure 2: Visualisation of the number nodes in an attack target's swarm and routing table for 100 runs (overlaid). Regular operation is followed by a surge of malicious nodes after starting the attack.

used an attack duration of 50 minutes. Given that the pregenerated IDs amount to more than the default 600 `lowWater` many peers, poisoning a targets routing table and eclipsing the swarm is virtually equivalent. As a result, few relayed connections are sufficient to eclipse a target.

Fig. 2a visualises the number of attackers and other nodes in the swarm and the target's routing table for all 100 runs. First, the plot clearly shows that all 100 victims were well connected and that during the first 5 minutes the `lowWater` mark was never undershoot. Starting the attack after 5 min-

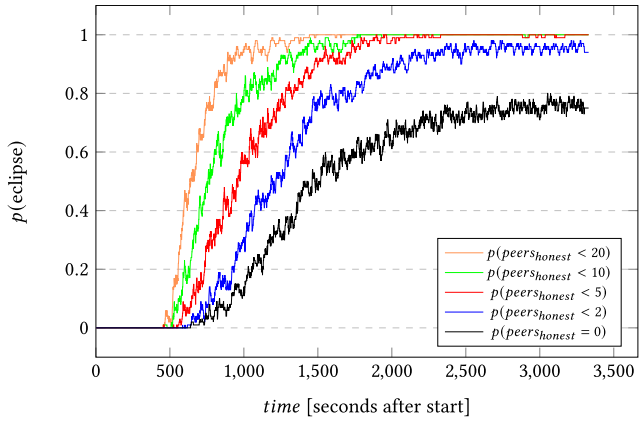


Figure 3: Probability of eclipsing a node with default settings (`lowWater`=600, `highWater`=900, grace period=20s)

utes, the number of attackers in the swarm and the routing table increase almost instantly, while the number of other nodes drops rapidly. After less than ten minutes, the routing tables of all attacked nodes are fully occupied by malicious nodes, while after less than 17 minutes, the probability of fully eclipsing a node is already $> 50\%$ as shown in Fig. 3. Even less time is required in cases where an attacker's goal is not to fully eclipse a node but to prevent a node from discovering any content with high probability. As also shown in Fig. 3, diminishing a target's swarm to less than ten connections is virtually guaranteed in less than half an hour.

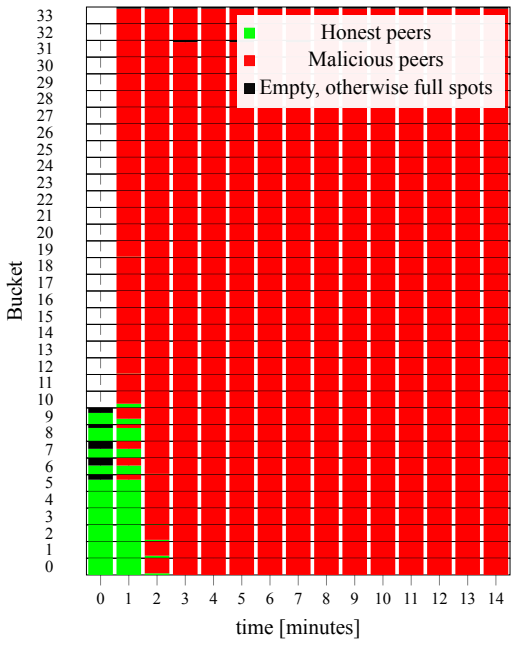


Figure 4: Visualisation of a victim's routing table over the first 15 minutes for IPFS 0.4.23, illustrating a sustainable effect

To better illustrate the impact of our attack on a target's routing table, Fig. 4 visualises table buckets for a single target. This reveals that initially only about 9 buckets are full or partially filled, before the attack is started. The second time slot on the x-axis shows the state directly after the start of the attack. All empty spaces in all buckets until bucket 33 are filled up by the attackers. The third time slot shows that after the connection manager tries to reduce the number of connections, the number of other peers decreases drastically, meaning the node has been tricked to harm itself. After the next connection cleanup phase, the routing table is fully poisoned.

5.2 Bootstrap Settings

This setting uses an unmodified go-ipfs v0.4.23 node configured to the same settings as the official bootstrap nodes (`lowWater` = 1000, `highWater` = 2000, 60s grace period). The goal of this evaluation is to see whether we can eclipse the official bootstrap nodes. The attack duration was set to 50 minutes.

For bootstrap-node-like configurations, the number of peers using a pre-generated ID for routing table poisoning is not sufficient to have the ConnMgr prune all honest peers from an attack target's routing table. As a result, the number of relayed connections can easily explode when seeking to fully eclipse such a node, which can result in inadvertently overburdening an attack target. In order to keep this period of high strain as short as possible, we disable spawning relayed connections as soon as a target's DHT is fully poisoned. As shown in Fig. 2b this does not result in a successful eclipse attack against an IPFS node run with bootstrap node configuration. However, the swarm state is irrelevant for actual bootstrap nodes, as swarm connections are not used for bootstrapping. In addition, new swarm connections to the bootstrap nodes are established as nodes are constantly joining the network Therefore, attacking the swarm of bootstrap nodes is futile. Nevertheless, even poisoning "only" the bootstrap nodes' routing tables results in all newly (re)connecting peers to learn only about malicious nodes, completely disrupting the live IPFS network over time. Since this strategy is highly churn-dependent, estimating the time required to reach the whole network is beyond the scope of this work.

5.3 Attack Costs

Since our attack is rooted in pre-generating large quantities of identifiers, which require terabytes of storage space and a fair amount of computing power, this task drives costs. In order to quantify the actual costs of running large-scale attacks against IPFS, attention must also be paid to key network metrics provided by Henningsen et al. [9]. Most prominently, only about 3000 nodes were recorded to responded to connection attempts, which means that the vast remainder of IPFS nodes

is likely operated by private individuals behind a NAT. As a consequence, it is sufficient to attack these 3000 nodes to prevent the network from responding to any queries for honest nodes connecting to the network. In addition, permanently occupying the bootstrap nodes' routing tables will prevent any (re)connecting node from ever connecting to the live IPFS network. Combining these two strategies will thus have a devastating effect.

In accordance with this setting, deployment- and routing-table-organisation-based attack model parameters matching our attack strategy can be set as follows:

$n = 3000$ (= number of all reachable DHT nodes)

$s_{100} = 2 \times 20 \times 3000$ (in accordance with Eq. 3)

$e = 20 \times \lceil \log_2(3000) \rceil$ (in accordance with Eq. 4)

Up-Front Costs The only up-front cost for our attack is related to pre-generating 146bn identifiers, which occupy 29TB of disk space (see Section 4.2). A quick search on a European price comparison service reveals a per-terabyte price of less than €20 including VAT[19] for external hard drives. This results in an up-front storage cost of less than €600, to store 29TB of identifiers[20]. In order to actually generate this many identifiers, any commodity personal computer can be used, adding an estimated €1000 for a machine dedicated to generating identifiers, featuring an *AMD Ryzen 3700X* octa-core CPU[21]. This results in an estimated up-front cost of less than €2000 including a large margin for electricity costs that easily covers the time it takes to generate the required amount of identifiers as outlined in Section B.1. Mapping this number to the attack model reveals that this ID pre-generation is not dependent on any other parameters. As such, these costs are considered part of c_a (additional attack costs), while $c_{s_{100}}$ can be neglected.

Running Costs The evaluation setup used to eclipse a single node relied on virtual machines with 4 cores and 16GB RAM, which cost €0.031 per hour and instance[22] including VAT. These costs already include c_e, which corresponds to the cost of running nodes to fill all routing table spots for a particular network size. However, as we also inflate the scores of these nodes and run additional nodes to occupy more DHT spots for additional ConnMgr points, these running costs cover c_e, while also contributing to c_a. Mapping this to the 3000 reachable IPFS nodes discovered by Henningsen et al. [9] results in running costs of $3000 \times 0.031€ = 93^{€}/_{h}$ to attack all reachable nodes simultaneously. This does not allow conclusions to be drawn regarding the overall number of active IPFS users, since it does not respect the network's churn rate. However, this has no bearing on the cost of eclipsing all available nodes (as this number would not change, only individual attack targets would come and go). Due to libp2p's routing table not keeping disconnected peers, a fully eclipsed node is known to be undiscoverable by the rest of the network. Once this is achieved, the cost of keeping a node eclipsed drops significantly, since it is sufficient to maintain as little as four connections to keep the node from re-connecting to bootstrap nodes. This scenario is no longer restricted by CPU or RAM utilisation and has therefore negligible bearing on attack costs. However, any churn requires continuous runs of the full attack against newly joining nodes, which inflates costs. As of IPFS 0.4.23, global attacks can therefore be made more economic by targeting only bootstrap nodes. *In general, however, this attack scenario scales linearly with network size/churn rate, enabling highly efficient, global attacks.* Recalling that it takes less than an hour to eclipse an average node with 75% probability (see Fig. 3) makes it possible to estimate the absolute cost for reaching this 75% target both for individual nodes and globally, in accordance with Eq.5, and Eq. 6, respectively.

$$c_i = \quad 2000€ + 0.031^{€}/_{h} \times 1h \quad \approx 2000€ \tag{5}$$

$$c_t = \quad 2000€ + 3000 \times 0.031^{€}/_{h} \times 1h \quad \approx 2100€ \tag{6}$$

While this will impact operations and fully eclipse a significant portion of all DHT nodes, it will not fully disrupt and halt the network, since the chance of success is 75%. Considering the low up-front cost, however, the actual cost of isolating individual nodes is negligible, even if it takes many more hours to fully eclipse them. This observation enables a different kind of attack strategy to actually grind the complete DHT-based IPFS network to a halt.

Fully Disrupting the Network Considering the virtually perfect success rate for poisoning a node's DHT after running the attack merely for a few minutes (as shown in Fig. 2b) and also the low number of bootstrap nodes, eclipsing bootstrap nodes presents an attractive global attack strategy. After all, if all bootstrap nodes' routing tables are fully poisoned, any regular node carrying out the bootstrapping routine (after being eclipsed themselves, or after a restart) will only receive information about attacker-controlled nodes. In accordance with Section 4.1, collapsing the DHT in this way will lead to eventually disconnecting all nodes from each other, fully disrupting the network. Keeping bootstrap nodes eclipsed does require running the full attack, since newly joining nodes will continuously establish connections. The cost for global attacks can therefore be estimated as follows: At a cost of $0.031^{€}/_{h}$ for a cloud instance targeting a single node, attacking the eight bootstrap nodes used as of IPFS 0.4.23 results in running costs of $8 \times 0.031^{€}/_{h} = 0.248^{€}/_{h}$. Mapping these numbers to the attack model allows for calculating the total attack cost according to Eq. 7.

$$c_t = \underbrace{2000€}_{\text{ID generation}} + \underbrace{(8 \times 0.031^{€}/_{h} = 0.248^{€}/_{h})}_{\text{keeping bootstrap nodes eclipsed}} \tag{7}$$

[19] https://geizhals.eu/?cat=hdx&xf=5588_HDD [20] Buying physical disks is significantly cheaper than current rates for cloud storage for the scope of our attack. [21] https://geizhals.eu/?cat=sysdiv&xf=10863_8%7E6764_AMD%7E6770_Ryzen+3000 [22] https://www.hetzner.com/cloud

As can be seen, running costs are negligible, and up-front costs are also low, considering the impact of such an attack. Apparently, this formula does not rely on any network parameters, which would mean that the cost of disrupting the whole IPFS network would remain constant regardless of network size. If this attack was to be mounted on a network featuring significantly more DHT server nodes, however, the score of attacker nodes would need to be inflated more aggressively, as more honest nodes would occupy more DHT buckets to begin with. As the number of buckets occupied by honest nodes scales logarithmically with network size, so does the required ConnMgr score, which needs to be exceeded by attacker-controlled nodes. Recalling Section 4.2, our attack exploits the relaying subsystem to do so by spawning large numbers of relayed connection, each of which awards one point to an attacker-controlled node. The resource consumption of this attack part thus also scales logarithmically with network size and as a result will become the dominating factor as the number of honest nodes grows. *In effect, the cost of fully disrupting the DHT-based IPFS network through attacking bootstrap nodes scales logarithmically with network size*, as more powerful instances will be required.

Admittedly, the model introduced in Section 3.4 may appear to be at odds with the chosen attack strategy. This, however, is attributed to the fact that easily exploitable flaws severely reduce the impact of network-related parameters. As discussed in the following section, various countermeasures, which have since been rolled-out, significantly impact attack costs, which puts any potential strategies taking these measures into account more in line with the attack model. Moreover, this abstract model will be shown to enable qualitative comparisons with related attacks on other systems in Section 7.

6 Countermeasures

Our attack only works because it is easily possible to mount Sybil attacks, which were first introduced by Douceur in 2002 [5]. The easiest way to prevent our attack would thus be to prevent Sybil attacks. This, however, is hardly feasible in practice given the open, decentralised-by-design nature of IPFS, and its key promise to let anyone participate in the network without central governance. As Douceur put it: "With no logically central, trusted authority [...] it is always possible [...] to present more than one identity [...]" [5]. This is further supported by a 2006 survey by Levine et al. [12] and by a more recent study by Mohaisen and Kim [18]. Nevertheless, attackers can still be hindered from weaponising the *operation* of large numbers of malicious network nodes to a great extent. The *generation* of a large identifier set cannot practically be mitigated in decentralised systems when considering large-scale attacks. After all, it took us four days to generate enough identifiers to target networks consisting of billions of nodes (see Appendix B). While *Proof-of-Work* (PoW)-based ID generation as proposed by Baumgart and Mies [3] would inflate up-front costs, Prünster et al. [20] have reached the conclusion that this is ultimately futile even for modern systems employing self-certifying identifiers and authenticated end-to-end encryption. Even a moderately funded attacker could simply invest in enough computing power to counter any realistic PoW factor.

Presenting our attack to Protocol Labs as part of the responsible disclosure process has intensified an already ongoing effort of hardening IPFS/libp2p, resolving the issue of unconditional removal of DHT nodes (see Section A) in the libp2p version that ships with go-ipfs v0.5, as released on April 28, 2020. Apart from that, many other fixes were released, with go-ipfs 0.6 (published in June) effectively preventing casual attackers from carrying out the attack presented in this paper.

Protocol Labs allowed us to attack bootstrap nodes running go-ipfs 0.5 right after its release, as well as performing attack runs on 0.6. In accordance with the timeline of releases, major changes affecting our attack are highlighted for go-ipfs 0.5, followed by a discussion on 0.6 improvements as well as other fixes related the discovered vulnerabilities.

IPFS 0.5 Mitigations Amongst many other changes, the libp2p version shipping with go-ipfs 0.5 in April 2020 introduced a DHT eviction strategy including periodic routing table refreshing and testing peer liveliness based on three parameters: (1) *Time of last successful outbound query*, (2) *last time a peer was considered useful* (see below), and (3) *eviction grace period*, which depends on bucket size and refresh period; typically in the order of 45 minutes. The first parameter is used during periodic routing table refreshes. A ping is issued when a peer has not been successfully queried within the grace period. Failure to respond results in eviction. Consequently, stale peers are periodically removed from the DHT even if the buckets they reside in still feature vacant spots. A peer is considered useful, if it either responded first to a query or if responding took less than twice the time of the fastest responding peer. Whenever this condition is met, the last useful time is recorded. In case a bucket is full, while another peer shall be added to this bucket, peers whose last useful time lies beyond the grace period are evicted. In such cases, however, responding to a query with an empty result is also considered useful by the DHT implementation (even though it is not *actually useful* in reality). This prevents nodes from penalising honest peers that simply could not provide the data required to respond to a query, which helps new nodes join the network. At the same time, however, becoming useful can be relatively cheap, since no information is required to still become useful. In addition, only routing table peers are evaluated for usefulness, due to the usefulness definition being limited to DHT operations. Thus, even a theoretic swarm node providing all the content ever queried would not join the routing table easier than any other node.

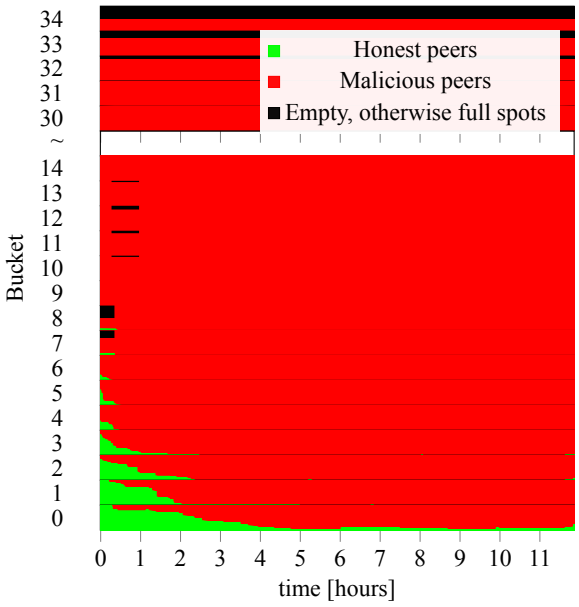
As a consequence of these changes, the connection trimming of the ConnMgr has no immediate effect on the routing

table, since even disconnected peers are remembered and are reconnected to if necessary. Thus, `lowWater`, `highWater` marks, and grace period also have no effect on the routing table. Given that the typical eviction grace period is close to an hour, the cost of poisoning a target's routing table is increased by several orders of magnitude compared to the mere minutes required to fully take over an IPFS 0.4.23 node's routing table. In fact, even after twelve hours, an IPFS 0.5 node in bootstrap configuration still features honest peers in its routing table (see Fig. 5a).

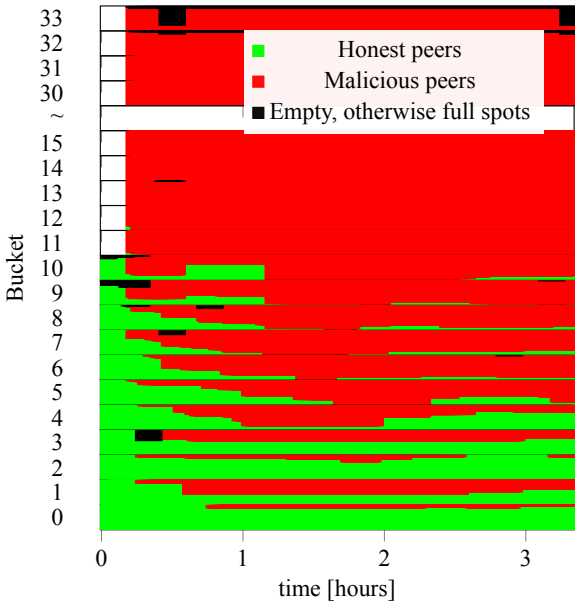
Shortly after releasing go-ipfs 0.5 on April 28, 2020, Protocol Labs let us evaluate our attack against one of the production IPFS bootstrap nodes over three hours in the live IPFS network in order to gauge the impact of the newly deployed countermeasures. Since this concrete attack run targeted a production system, the evaluation period was chosen in such a way that results could be obtained to make an initial judgement regarding the deployed countermeasures' utility without causing any real disruption to the network. The results are presented in Fig. 5b. Compared to attacking nodes specifically for evaluation purposes (not used by others for actual bootstrapping), an increased resilience can be observed. This was to be expected, since the increased query frequency to bootstrap nodes awards a larger set of honest peers a *useful* state. As a consequence, these peers are kept in the routing table. After only one hour, however, the initial number of 200+ honest peers in the bootstrap node's routing table could be more than halved and then remained below 100, clearly demonstrating the impact our attack still has on IPFS 0.5. Protocol Labs's estimate regarding attack difficulty for go-ipfs 0.5, is that fully poisoning a bootstrap node's routing table is still possible within several days.

IPFS 0.6 and 0.7 Measures The most crucial measure to boost attack costs by orders of magnitude was included in go-ipfs 0.6 released in June 2020. In effect, IP-addresses are now considered with respect to adding nodes to the local routing table, making it impossible to mount large-scale Sybil attacks from a single host. Instead, at most three nodes associated with a single host (IPv4, IPv6[23]) can become resident in a node's routing table. Therefore, this measure by itself already inflates the cost of attacking a single node by over two orders of magnitude compared to IPFS 0.5. Mapping this against the estimate of several days to fully eclipse a bootstrap node clearly puts this out of reach for casual attackers. Compared to theoretic proposals of limiting the numbers of identifiers that can be advertised to a P2P network per IP address, multiple nodes can still join the network from the same host. In fact, this aligns with the distinction between client and server-mode DHT that aims at mapping to nodes operated behind NATs. In addition, IPFS 0.7 deprecated the previously used transport security mechanism, which breaks compatibility to pre-0.6 releases.

[23] Even presenting multiple IPv6 addresses from a single large IPV6 subnet has no effect.



(a) Attack performance on a IPFS 0.5 node run with bootstrap node configuration over twelve hours. This node baehaved like a regular node and was not used for actually bootstrapping. Thus, it more closely resembles a regular node, not a bootstrap node with respect to network interactions.



(b) Attack performance on a live IPFS 0.5 bootstrap node over three hours. The attacks was launched 15 minutes after starting to collect metrics.

Figure 5: Visualisation of routing tables for go-ipfs 0.5. Higher attack resiliency can be observed for the bootstrap node due to interacting with more nodes. By comparison, the attack on a regular nodes becomes increasingly successful during the course of the first five hours. Honest peers are visualised in green, malicious peers in red and empty spots in otherwise filled buckets are shown in black.

As a result, older instances are required to update to a release containing fixes, if operators want to continue to participate in the network.

Additional Deployed Changes Aside from these key changes, additional countermeasures were rolled out since IPFS 0.4.23. These include a fixed score for nodes acting as relays to prevent relay-based inflation of ConnMgr scores. While many third-party projects rely on different parts of libp2p's functionality, the relaying subsystem is expected to be used in virtually all P2P-related projects that require connecting users behind NATs. Therefore, this change is expected to impact many libp2p-based applications. Moreover, nodes in the lowest two DHT buckets are exempt from pruning, while higher buckets are now assigned a fixed score of five points. While this may seem counter-intuitive, it prevents attackers capable of generating a huge body of valid identifiers from gaining an advantage for small network sizes by ConnMgr points awarded from the DHT subsystem. In effect, nodes are scored more equally due to these changes. Moreover, libp2p now verifies reachability of advertised IP addresses, and only adds those nodes who actually respond to connection requests. This already goes a long way towards solving the issue of unconditional trust and makes it harder to become resident in routing tables. In addition, configuration of direct peering agreements was also elevated to a core feature of the IPFS reference implementation.

Revisiting Attack Cost Putting the impact of these countermeasures into context is easily possible, when inserting updated cost factors into the abstract attack model introduced in Section 3.4 and comparing them to our attack's cost estimates obtained for IPFS 0.4.23 (see Section 5). While up-front costs remain unchanged, running costs have increased dramatically due to a discrete cloud instance being required for every three attacker nodes aimed at poisoning a node's routing table. As these instances will only run a small number of nodes, cheaper, low-power offers can be used, priced at $0.005€/h$, for example[24]. In addition, the relaying subsystem cannot be exploited any more to gain more ConnMgr points than an honest connection by relaying large numbers of nodes, which is why higher-performing cloud instances would not provide any advantage. In effect, our attack cannot be sped up and is dependent on churn only. Moreover, attacker-controlled nodes occupying higher buckets than regular nodes (which is based on network size) makes little sense. In fact, the naïve attack strategy from Section 4.1 becomes attractive again, which results in estimated costs for poisoning an individual node's routing table according to Eq. 8:

$$c_e = \underbrace{20}_{k} \times \lceil \log_2(\underbrace{3000}_{\text{total DHT nodes}}) \rceil / \underbrace{3}_{\text{nodes per cloud instance}} \times 0.005€/h \approx 0.4€/h \quad (8)$$

In order to estimate the absolute cost of such an attack, it is important to take into account how the deployed countermeasures affect the required duration of such an attack to be successful. Monitoring regular IPFS nodes revealed that the lowest two buckets of their routing tables still contained a combined number of twenty nodes even after a week. The majority of these honest nodes were evicted from the DHT within the first 24 hours, with little changes being observed afterwards. Since all routing information in these lowest two buckets is protected from being pruned by the ConnMgr, this observation provides insights with respect to how the deployed countermeasure improved attack resiliency: An attack needs to be run for weeks (at a cost of $c_i = 0.4 \times 24 \times 7 = 67.20€/\text{week}$) rather than minutes in order to fully poison a single node's routing table. As such, individual and global attacks become infeasible for casual attackers.

7 Related Work

As our work presents an attack on a peer-to-peer system in production use, this section focuses on work related to analysing and attacking similar live systems. For theoretical models and surveys on this matter, we refer to the works of Levine et al. [12] and Mohaisen and Kim [18] (see Section 6).

Although IPFS was launched in 2013 and has continuously gained traction, scientific literature on IPFS is still scarce. Apart from the original whitepaper [4] and protocol specifications of varying maturity[25], little in-depth documentation on IPFS is currently maintained. Metrics on the overall IPFS network are also not available. However, a 2020 paper [9] crawled the live IPFS network, the results of which were used to estimate the cost of our attack in Section 5. This work also stressed that no countermeasures against Sybil attacks were in place as of version 0.4.23, contrary to the original IPFS whitepaper. The authors also observed that the way IPFS implements content discovery (first querying all connected peers and only then falling back to querying the DHT in case none of the contacted peers would serve the requested content) provides some resilience against eclipse attacks aimed at the DHT. Our work demonstrates, however, that this defence was extremely limited in scope, leaving the possibility to fully eclipse a node in less than one hour with $\approx 75\%$ probability.

Similar to our work on IPFS, Heilman et al. [7] demonstrated a successful eclipse attack against *Bitcoin* [19]. On the surface, this attack was based on flaws similar to the issues we discovered in IPFS. Most prominently, peers that were still running and maintaining connections to an attack target could be replaced by fresh malicious peers. Among the countermeasures deployed to the Bitcoin reference implementation, an adapted eviction strategy—keeping live peers instead of having them easily replaced—showed the most impact in the context of defending eclipse attacks. Actually carrying out an eclipse attack against even a single Bitcoin node without such countermeasures is considerably more expensive than our

[24] We rely on the same service provider for this estimate, whose pricing of discrete instances is a better fit that buying IP addresses.

[25] https://github.com/ipfs/specs

Table 1: Cost comparison of different attacks on IPFS (with/without countermeasures) and Ethereum (Henningsen et al. [8])

	c_i	c_t	Runtime[26]	Real-world cost c_i	Real-world cost c_t	Params
Ours (IPFS 0.4.23)	$2000€ + 0.031€/h$	$2000€ + n \times 0.031€/h$	$<1h$	$2000€$	$2100€$	$n = 3000$
Ours (IPFS 0.4.23; Bootstrap attack)	n/a	$2000€ + b \times 0.031€/h$	∞	n/a	$2000€ + 0.248€/h$	$b = 8$
Naïve (IPFS 0.7)	$0.4€/h$	$n \times c_i$	weeks	$67.20€/\text{week}$	€€€	$n = 3000$
Ethereum	$2 \times 0.05€/h$	$p \times 25000 \times 2 \times 0.05€/h$	$\approx 5d \;/\; \infty$	$\approx 12€$	???	$n = 25000,\ p = ?$

attack against IPFS prior to version 0.6, due to the fact that widely-distributed IP addresses are required. This was mainly due to the different organisation of Bitcoin's routing tables compared to the libp2p DHT and not primarily for reasons of hardening against attacks. Since the paper by Heilman et al. does not provide tangible cost estimates to use for an abstract attack model, a qualitative economic comparison with our work is considered out of scope.

An attack that uses similar principles as ours has been shown to be successful against the *Ethereum* cryptocurrency [14][27]. However, there are several important differences to our work. Most prominently, our attack is more powerful, as it works in near-realtime by tricking a node into actively disconnecting itself from the rest of the network. In contrast to the attack on Ethereum, our attack thus requires no other forms of DoS or high churn rates attacks to succeed. This is crucial, as system downtime could be easily detected, while our attack simply requires additional connections to be established (many of which can be multiplexed, thus not showing up on network monitors). The generic issue of how IPFS implements connection management is distinctly different from Ethereum and also independent of the DHT-related attack vector that shows similarities to the attack on Ethereum. As such, our attack is not limited to the DHT subsystem that shares much of the conceptual functionality of Ethereum's P2P network (although it exploits the fact how the IPFS DHT subsystem awards ConnMgr points). Moreover, our way of gaming IPFS's ConnMgr makes it possible to deliberately have nodes outside an attackers control induce churn with low effort. This in itself is an attack not discussed in related works targeting systems used in practice[28]. Our attack against IPFS v0.4.23 also takes only minutes to fully poison any node's routing table and less than an hour to fully eclipse a node with high probability, without requiring additional DoS attacks. In addition, countermeasures such as non-public mapping from node ID to bucket are simply not applicable to IPFS since these would render one of its core features defunct. In short, the attacks against cryptocurrencies require the target to reboot, take several days to execute, are orders of magnitude more expensive, and suggested countermeasures are not applicable to IPFS. One reason for these differences is the fact that attacks on cryptocurrencies typically aim at single nodes while our motivation was to mount cheap global-scale attacks.

The same applies to the follow-up attack on Ethereum by Henningsen et al. [8]. This attack's scenario and execution reflects the naïve attack strategy introduced in Section 4.1, making it churn-dependent. As Ethereum is also Kademlia-based, the same attack formulas apply. In accordance with Henningsen et al., the number of relevant Ethereum nodes is small enough to also result in negligible cost for pre-generating identifiers (based on an assumed total of 25000 relevant nodes). In addition, flaws of Ethereum's peer-discovery process are exploited to drastically cut running costs. In effect, eclipsing a single node amounts to operating two hosts in distinct /26 IPv4 subnets. Based on our cost model (and mapping the more powerful machines used by Henningsen et al. to cloud offers), this would result in running costs of $c_i = 2 \times 0.05 = 0.1€/h$. The authors came to the conclusion that this attack needs to be run for about five days when not forcing the victim to reboot, which results in overall costs of $\approx 12€$. Even when considering the fact that multiple targets could be attacked by the same cloud instance (a parallelisation factor that decreases global costs), expanding this attack to the estimated 25000 relevant nodes scales linearly, in accordance with Eq. 9.

$$c_t = \underbrace{p}_{\text{parallelisation factor } <1} \times \underbrace{25000}_{\text{number of nodes}} \times \underbrace{2 \times 0.05€/h}_{\text{cost of running two instances}} \tag{9}$$

Not knowing about the exact amount of resources required to attack a single node in terms of main memory and CPU power, precisely estimating the parallelisation factor is difficult. However, linearly scaling with network size and taking days to show results, this attack on Ethereum is clearly less economic on a global scale than our approach aimed at bootstrap nodes—especially when considering that such attacks need to be run continuously. Still, actually evaluating such a strategy against the whole Ethereum network is beyond the scope of this work. Moreover, none of this is of any signifi-

[26] Global attacks need to be run continuously, while the runtime for attacking single nodes indicates how long it takes to reach an eclipsed state.

[27] This attack was performed prior to Ethereum's planned switch to libp2p.

[28] Evaluating the impact of such churn attacks is out of this work's scope.

cant practical relevance due to Ethereum's switch to libp2p. As such, our findings and the deployed countermeasures also benefit Ethereum and can serve as basis for further, in-depths analysis and hardening efforts. As a starting point, Table 1 provides a cost comparison of all discussed attack strategies and their targets (note that the attack on IPFS 0.7 and variants of the Ethereum attack are highly churn and network-size dependent).

8 Conclusions

In this paper, we described, and demonstrated a successful end-to-end eclipse attack on IPFS. Exploiting vulnerabilities of the libp2p library, one aspect of our attack is to successfully poison routing information locally stored by nodes of IPFS's underlying P2P network. In addition, we have shown that our attack enables us to eclipse arbitrary IPFS nodes and, consequently, to disrupt the entire public DHT-based IPFS network. Applying our attack on live IPFS nodes demonstrated the effectiveness and feasibility of the attack. The conducted evaluations have demonstrated the alarming fact that attacks on a global scale can already be mounted with low effort, meaning that this attack is feasible even for attackers with limited resources.

The impact of our proposed attack is substantial mainly for two reasons. First, our attack exploits a conceptual flaw in the connection management of IPFS for which there is no easy solution. Secondly, our work has lead to a successful, ongoing hardening process. The entire ecosystem beyond decentralised exchange of data (but also other IPFS-based services) benefits from upstream releases incorporating fixes. Two major version of the IPFS reference implementation were released since reporting our findings to Protocol Labs, both of which contain fixes, resulting in a huge increase of attack costs. As a consequence, casual attackers will not be able to replicate our attack against updated nodes.

The main reason why our attach was so successful with such low costs can be attributed to a lack of integration: On the one hand, IPFS in general (and libp2p in particular) recognises the fact that a P2P system's infrastructure is upheld by its users and therefore provides a feedback loop to have application-layer behaviour impact the P2P network layer. In the case of IPFS, this feedback loop is realised as the ConnMgr's scoring system. On the other hand, different subsystems still award points oblivious to each other. This made it possible to game the ConnMgr without much effort. The fully open and extensible nature of IPFS will always introduce issues of this kind, as third-party subsystems have the potential to interfere with each other's scoring methods. At the same time, our attack harnessed points from the P2P network layer as well as from the application layer, also highlighting a lack of vertical integration. This lesson should, at the very least, be carefully observed by developers of P2P systems, as it highlights how a layered mental model towards security does not apply well to this environment. Moreover, our attack model can be applied to other systems for gauging attack costs and comparing attack strategies. Following this course makes it possible to identify cost-driving factors and to distinguish between the impact of the operational environment (number of nodes, churn, ...), system properties (such as routing table design, and the choice of using fixed bootstrap nodes), and implementation flaws. As demonstrated by the fixes introduced with IPFS 0.7, minor design changes deployed to the same environment can have a huge impact.

References

[1] JD Alois. *Protocol Labs Files Two Form D's with SEC Regarding Enormous Filecoin ICO*. Aug. 26, 2017. URL: https://www.crowdfundinsider.com/2017/08/121075-protocol-labs-files-two-form-ds-sec-regarding-enormous-filecoin-ico/ (visited on 10/08/2020).

[2] Dietrich Ayala. *IPFS in Opera for Android*. URL: https://blog.ipfs.io/2020-03-30-ipfs-in-opera-for-android/ (visited on 10/08/2020).

[3] I. Baumgart and S. Mies. "S/Kademlia: A Practicable Approach towards Secure Key-Based Routing". In: *2007 International Conference on Parallel and Distributed Systems*. Dec. 2007, pp. 1–8. DOI: 10/dc6c8p.

[4] Juan Benet. *IPFS - Content Addressed, Versioned, P2P File System (DRAFT 3)*. July 2014. URL: https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX/ipfs.draft3.pdf.

[5] John R. Douceur. "The Sybil Attack". In: *Peer-to-Peer Systems*. Ed. by Peter Druschel, Frans Kaashoek, and Antony Rowstron. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, pp. 251–260. DOI: 10/b8hmpj.

[6] I. Fette and A. Melnikov. *The WebSocket Protocol*. RFC 6455. Dec. 2011. URL: http://www.rfc-editor.org/rfc/rfc6455.txt (visited on 08/12/2020).

[7] Ethan Heilman et al. "Eclipse Attacks on Bitcoin's Peer-to-Peer Network". In: *Proceedings of the 24th USENIX Conference on Security Symposium*. SEC'15. Berkeley, CA, USA: USENIX Association, Aug. 2015, pp. 129–144.

[8] Sebastian Henningsen et al. "Eclipsing Ethereum Peers with False Friends". In: *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*. 2019 IEEE European Symposium on Security and Privacy Workshops (EuroS PW). June 2019, pp. 300–309. DOI: 10/ghfq6p.

[9] Sebastian Henningsen et al. "Mapping the Interplanetary Filesystem". In: *arXiv:2002.07747 [cs]* (Feb. 2020). arXiv: 2002.07747 [cs].

[10] Paul Hill. *Catalan Referendum App Removed from Google Play Store*. en. https://www.neowin.net/news/catalan-referendum-app-removed-from-google-play-store.

[11] Protocol Labs. *Filecoin: A Decentralized Storage Network*. https://filecoin.io/filecoin.pdf. July 2017.

[12] Brian Neil Levine, Clay Shields, and N. Boris Margolin. *A Survey of Solutions to the Sybil Attack*. Tech. rep. 2006-052. Amherst, MA: University of Massachusetts Amherst, Oct. 2006.

[13] Molly Mackinlay. *IPFS Project Focus for 2020*. URL: https://blog.ipfs.io/2020-02-10-our-focus-for-2020/ (visited on 10/08/2020).

[14] Yuval Marcus, Ethan Heilman, and Sharon Goldberg. *Low-Resource Eclipse Attacks on Ethereum's Peer-to-Peer Network*. Tech. rep. 236. 2018.

[15] Petar Maymounkov and David Mazières. "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric". In: *International Workshop on Peer-to-Peer Systems*. Vol. 2429. Berlin, Heidelberg: Springer, 2002, pp. 53–65. DOI: 10/b9hjzk.

[16] David Mazières. "Self-Certifying File System". PhD Thesis. Cambridge, MA, USA: Massachusetts Institute of Technology, 2000.

[17] Ralph C. Merkle. "A Digital Signature Based on a Conventional Encryption Function". In: *Advances in Cryptology — CRYPTO '87*. Lecture Notes in Computer Science. Springer, Aug. 1987, pp. 369–378. DOI: 10/fj4rw8.

[18] Aziz Mohaisen and Joongheon Kim. "The Sybil Attacks and Defenses: A Survey". In: *Smart Computing Review* 3.6 (2013), pp. 480–489. DOI: 10/ggskgw.

[19] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. https://bitcoin.org/bitcoin.pdf. 2008.

[20] Bernd Prünster et al. "A Holistic Approach Towards Peer-to-Peer Security and Why Proof of Work Won't Do". In: *14th EAI International Conference on Security and Privacy in Communication Networks*. Vol. 255. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Singapore: Springer, 2018, pp. 122–138. DOI: 10/ggsdq6.

[21] Adin Schmahmann. *IPFS 0.5 Content Routing Improvements: Deep Dive*. URL: https://blog.ipfs.io/2020-07-20-dht-deep-dive/ (visited on 10/08/2020).

[22] Atul Singh et al. "Defending Against Eclipse Attacks on Overlay Networks". In: *Proceedings of the 11th Workshop on ACM SIGOPS European Workshop*. EW 11. New York, NY, USA: ACM, 2004. DOI: 10/cr3pdt.

[23] The libp2p Team. *Libp2p in 2020*. URL: https://blog.ipfs.io/2020-06-09-libp2p-in-2020/ (visited on 10/08/2020).

A Implementation Flaws

Apart from the conceptual issue of lacking integration between the ConnMgr and other subsystem,we have identified the concrete flaws, which contribute significantly to the performance of our attack. Exploiting these issues is therefore key for low-budget, high-impact attacks. This appendix provides more in-depth details on these matters.

Allowing Only Inbound Connections: While libp2p differentiates between inbound and outbound links, this has no bearing on the ConnMgr's trimming routine, making it possible to trim all outgoing connections.

Unconditional Removal of DHT Nodes: Since the ConnMgr does not interpret scores and tags, and DHT connections do not receive special treatment, it is possible to push all legitimate connections out of the DHT and replace them with malicious ones. This goes against the original Kademlia design, which favours older connections. Note that from the DHT's point of view, this characteristic is still upheld, but the ConnMgr manually disconnects already known connections that would prevent new ones from entering the DHT.

Stateless Connectivity Monitoring: The DHT is instructed to re-execute a bootstrap manoeuvre to connect to preconfigured bootstrap nodes, whenever less than four open connections remain. However, no further action is taken even if this situation becomes stationary. This effectively enables an attacker to keep a node eclipsed with as little as four open connections, once an eclipsed state is initially reached. This monitoring loop to check connectivity is executed once a minute.

Static Bootstrap Nodes: IPFS relies on a static set of bootstrap nodes. Although this set can be configured, a node does not keep track of peers to which it was once connected to. As a result of this, a restarting node will always bootstrap against the same set of nodes, regardless of connectivity prior to restarting. As a consequence, compromising the default set of bootstrap nodes will affect all nodes as soon as they restart (and not only newly joining nodes).

Unconditional Trust: Although rigorous data integrity checks are a core feature of the main IPFS functionality, the same can not be said with respect to information concerning the network's node. This meta issue affects many subsystems. In short, almost all claims made by a peer about its characteristics and capabilities are taken at face value, even when simple checks could expose cheating. As for the DHT, this makes it easily possible to fill a node's DHT with bogus IPs.

B Technical Attack Details

This appendix provides in-depth details on how we implemented the key steps of our attack. More specifically, this covers the one-time ID generation phase, the setup, and attack phase depicted in Fig. 1 and includes details on remotely probing an attack target's state.

B.1 ID Generation

Our attack requires efficient generation of vast amounts of valid identifiers to position nodes at specific distances to *any* node. Thus, a one-time ID pre-generation routine is run. libp2p supports RSA and EC keys. By generating only EC-based identifiers, both generation times and the amount of storage required to manage a large set of IDs is reduced compared to RSA. To maximise throughput, we increment an integer and interpret it as a private key[29]. This approach generates 8-10k keys per second per CPU core based on an *Intel*® *Xeon*® E5-2699 v4 CPU @ 2.20GHz. Overall throughput is mainly limited by IO speed.

Our storage format is simple and efficiently searchable: Each generated key is stored to a file named after the first 14 bits of the DHT identifier corresponding to the key, along with this identifier. This set of pre-generated identifiers amounts to 29TB of data and encompasses $\approx$ 146Bn individual IDs. Still, it is possible to efficiently query this database, as the number of entries per file remains manageable. A dedicated service in charge of producing keys and identifiers corresponding to any node's lowest 33+ DHT buckets, based on the target node's ID as input. Answering a query takes less than five minutes[30].

B.2 Probing

In order to carry out our attack, ways to continuously probe an attack target's state are required. Additionally, some static parameters are needed during the attack's setup phase.

Setup Phase Naturally, an attack target's IP address is required to connect to it. This information is obtained by operating a regular IPFS node that is connected to the IPFS network, This node is used to query the target's IP address, which is then fed into the actual attack carried out on a distinct machine.

Moreover, the bucket size of the target's routing table is required in order to advertise a precisely distributed set of identifiers for routing table poisoning. Querying the target for any identifier will trigger a response containing *bucketSize* many peers, thus providing this information.

While `lowWater` and `highWater` marks are theoretically required to perform our attacks, choosing too high values has no impact on attack success rates, but only consumes more resources than necessary. Given that even critical infrastructure like bootstrap nodes use values of 1000 and 2000, respectively, this does not cause any real issues with respect to attack performance. However, simply starting with those values and and observing the impact of connection trimming allows for detecting values set too high, which enables reducing each value accordingly.

The interval for the ConnMgr's connection trimming routine is hardcoded to 1 minute. However, detecting disconnect waves when maintaining hundreds of open connections is trivial, as our attack operates a little over `highWater` many connections to ensure that disconnect waves are triggered. The grace period used to protect newly established connections is irrelevant for our attack and is thus not probed.

Our attack targets the latest IPFS version (0.4.23) released as of April 27, 2020. Earlier versions are even easier to eclipse. However, our attack performs a strict superset of the actions required to eclipse earlier versions and thus requires no knowledge of the attack target's IPFS version, except for increased efficiency. Still, the IPFS protocol defines a message to remotely query a node's version.

Continuous Probing Our attack requires knowledge about the target's routing table. In essence, we need to know which buckets are occupied by honest nodes, in order to outperform these nodes from the ConnMgr's point fo view. As mentioned before, the target will respond with *bucketSize* many closest peers to any query for other peers. We can thus simply traverse the set of pre-generated identifiers used for poisoning the target's routing table and query for one identifier in each bucket. This way, it is possible to construct a contiguous view of the target's routing table and know precisely which nodes occupy which buckets. Given that our pre-generated ID set consists of $\approx$146Bn nodes, this easily covers all realistically possible routing table configurations that can ever be encountered. These queries are performed once during each attack loop.

B.3 Gaming the ConnMgr

This section describes the main attack loop and the actions performed to trick a victim's ConnMgr. The overall goal is to raise the score of the connections made by an attacker above the highest score of any legitimate node connected to the victim. Based on observation of live IPFS nodes, the score of connections to honest peers will usually range from 0 to around 20. In order to reach this goal, our attack strategy relies on three sources of points to game the ConnMgr:

DHT: Each node that occupies a DHT spot is awarded points according to Eq 2. The amount of identifiers we pre-generate is several orders of magnitude larger than the number of nodes participating in the live IPFS DHT. Because of this, simply connecting using these IDs is enough to be assigned a spot

[29] Since identifiers are hashed prior to calculating distances, lack of randomness in the raw key material is not an issue. [30] This could be further sped-up thorugh parallelisation.

in the target's DHT, as most buckets for those identifiers will be empty. We can therefore maintain more than 400 connections[31] that will be awarded enough points to become resident in the target's swarm. This, however, is significantly less than the required default `lowWater` value of 600.

Bitswap: As mentioned in Section 3.2, Bitswap awards points for unsolicited content advertisement (which is understandable from a content-distribution perspective). We exploit this by continuously advertising an empty block of data. This is cheap for an attacker, since sending such a message every few seconds suffices, with no need to process responses and results in 10-16 points. Other ways of inflating a connection's score based on Bitswap include re-sending blocks a target previously requested.

Relaying: A virtually unlimited source of ConnMgr points is the relaying subsystem that is used to help nodes located behind NATs or firewalls reach the network. For one, simply advertising relaying capabilities to the target already awards a fixed amount of two points. More importantly, however, actively relaying connections from and to the target awards one point for each relayed connection. Given that libp2p supports multiplexing many virtual connections over a single (TCP) link, the number of available ports is not a limiting factor for this strategy. Initial experiments have shown that > 1000 connections can be multiplexed over a single link.

This last method of obtaining points can be especially devastating, since finding a countermeasure is challenging. While pathological cases like those from our initial experiments could be detected using heuristics, the general strategy of considering a link supporting many relayed connections important is understandable, especially in real-world settings that include firewalls and NATs. The required resources for creating a relayed connection are minimal: The only thing that is really required is a (randomly generated) key pair to obtain a valid self-certifying node identifier. This is then used during the initial handshake when establishing an end-to-end-encrypted connection. While this comprises computationally somewhat expensive asymmetric cryptographic operations, these have to be performed only once during connection establishment. Overhead for the node acting as relay is also moderate. Given that our attack strategy is based on performing a Sybil attack from a single host no actual links between relay and relayed nodes are required, since these nodes are, in fact, virtual.

When combining this way of inflating connection scores with the continuous probing of a target's routing table, a highly efficient attack behaviour can be implemented.

In order to occupy all spots in the target's routing table and, subsequently, eclipse the whole swarm, any nodes previously connected need to be outperformed. However, fully poisoning the routing table is prioritised, since the DHT is used for peer discovery and content routing beyond content distribution. In order to accomplish this, estimate the highest score of any honest node connected to the target as follows:

1. Based on the routing table information, we calculate the highest score over all legitimate peers that reside in the target's routing table according to Eq. 2.
2. We consider a safety margin of 10 points, meaning that we assume that each honest peer has an additional 10 points awarded from other subsystems, such as Bitswap. We apply this margin regardless of whether an honest peer is resident in the target's routing table or simply part of the swarm.
3. These two numbers are then added to arrive at the target score that needs to be beaten by at least `lowWater` many of our malicious nodes in order to have the target's ConnMgr trim all connections to honest nodes (except for those within the grace period).
4. In order to reach this score, we traverse the set of our malicious nodes and start relaying connections to randomly generated virtual nodes as required. We prioritise those of our nodes that are based on pre-generated identifiers, referred to as *DHT nodes* to fill the target's routing table. If this is not sufficient (which is the case for higher-than-usual `lowWater` marks), we also boost the score of the random nodes that are run to reach `highWater` many connections to trigger the ConnMgr's disconnect routine.

Careful observation might suggest that *precise* probing of the target's `lowWater` mark is required, since a too high estimate would cause those of our nodes that should occupy the lower buckets to be disconnected by the ConnMgr. While this is technically correct, our nodes reconnect to the target within milliseconds. This leaves only an extremely short window of opportunity for honest nodes to slip into the target's routing table. This is due to the fact that routing table spots that become vacant during a disconnect wave are not automatically filled by remaining connections. Instead, the DHT component of libp2p only reacts to a well-defined set of messages to insert peers into a node's routing table. One such message is a ping. We exploit this behaviour and re-establish any severed connection as soon as the ConnMgr executes its connection trimming routine and ping the target from all still connected nodes at the same time. As elaborated in Section 5, we have evaluated this strategy to be effective, as fully occupying any node's routing table takes mere minutes. Moreover, completely eclipsing nodes with $\approx 75\%$ probability takes less than an hour. In effect, this results in overall low costs, even when seeking to disrupt the complete public DHT-based IPFS network.

[31] The remainder of these connections are initially outperformed by honest ones.