Cryptoeconomic Systems •

Time-Dilation Attacks on Lightning Network

Gleb Naumenko, Antoine Riard, Reuben Youngblom

Published on: Aug 25, 2021

ABSTRACT

Lightning Network (LN) is a widely-used peer-to-peer network enabling faster and cheaper Bitcoin transactions. In this paper we outline the ways to steal funds from LN users, per which users cannot detect that they are victims, and thus cannot act. These are the first attacks based on eclipsing a Bitcoin node, which allows stealing funds without access to mining hashrate. The attacks involve dilating the time of the blockchain view of the victims by feeding the blocks at a slower, yet, hardly distinguishable from normal rate. In this paper, we discuss the difference between the security of LN and Bitcoin. We demonstrate three different attack scenarios, which are possible against many LN users today. With a moderate access to resources (in some cases couple hundreds of distinct IPs and a medium-tier VM), the attacks we discuss can allow stealing funds after keeping a node eclipsed for as small as 4 hours. This makes the attacks very practical. We also suggest countermeasures to make time-dilation attacks less feasible and minimize the consequences.

Introduction

Bitcoin is a peer-to-peer electronic cash system, which solves the double-spend problem with a trust-minimized architecture by letting everyone verify all transactions [1]. As of Nov. 2019, the system operates over at least 60,000 nodes[1] simultaneously running Bitcoin protocol software, while also having much more users of custodial services and trusted solutions.

This public auditability of the Bitcoin transaction history has been the foundation of removing third-party arbitrage from the Bitcoin ecosystem. Sadly, from the beginning it has been pointed out that the scalability which can be achieved by this approach is very limited [2]. Overcoming this major pitfall has been addressed by developing so-called *second-layer protocols*. For example, in Lightning Network (LN), solving the double-spend problem is inverted back to a private matter (as opposed to be solved *on-chain*) via payment channel constructions [2].

These protocols, however, introduce new assumptions and change the threat model of Bitcoin. In this work we explore how LN users may be subjected to stealing of funds, once their Bitcoin nodes are isolated from the honest nodes (*eclipsed*).

More specifically, we exploit the requirements of monitoring the Bitcoin blockchain and timely detecting relevant transactions, which is laid upon LN users. Per *time-*

dilation attacks, a malicious actor slows down block delivery to the victim, and then finalizes an outdated state of the Lightning channel on-chain before a victim can even notice.

The most significant properties of the attacks we demonstrate are:

- These are the first eclipse-based monetary attacks which doesn't require any access to mining hashrate.
- The attacks are practical considering current implementations of the Lightning Network and the relevant software.
- In principle, the attacks we demonstrate are very difficult to detect, unless serious countermeasures are taken.

The paper is structured as follows:

1. We provide the background required to understand the Bitcoin system, its advantages and limitations.
2. We describe Lightning Network, one of the most popular scaling solutions to Bitcoin.
3. We provide the background required to understand the time-dilation attacks on LN and explain the attack preparation phase.
4. We thoroughly explain the execution of time-dilation attacks, measure their cost and the benefits they provide to an attacker.
5. Finally, we suggest various countermeasures which would increase the bar required for an attacker to set up time-dilation attacks on LN, and reduce the losses for the victims.

Next, we review the background for our work.

Background

Bitcoin base layer

The primary goal of the Bitcoin base layer includes relaying and validating financial transactions. Bitcoin solves the double-spending problem by organizing transactions into a sequence of blocks.

A transaction in Bitcoin is considered to be unconfirmed unless it is included in a valid block. Then, the number of blocks created on top of that block represent the degree of confirmations the transaction has. This confirmation indication works largely due to the Bitcoin built-in incentivisation system: mining a block is a difficult and expensive task, which may result in a reward.

The more confirmations a transaction has, the more confident the receiver is that the transaction is unlikely to be reverted. We use *unlikely* because absolute transaction finality in Bitcoin does not exist by design. However, the incentives are aligned in a way that reverting a larger number of blocks becomes more and more unprofitable under the fundamental Bitcoin assumption that a fraction of dishonest mining power does not exceed 50% in the long run (usually roughly defined as several hours to several days).

Mining a Bitcoin block mainly consists of two phases: assembling a valid sequence of transactions and finding a random nonce, which would satisfy the Proof-of-Work algorithm requirements.

The Proof-of-Work difficulty adjustment rule changes the expected time of producing a block to be on average 10 minutes, based on the time it took to produce a block in the last 2048 blocks. These 10 minute intervals as well as an upper bound of the block size are used for a number of reasons, related to the security and scalability of the system.

These rules, however, has two negative consequences. First, they make Bitcoin transactions potentially expensive: competition for the block space creates a transaction fee market, which may drive fees up. Second, they make transactions slow: as explained above, in most cases confirming a transaction requires waiting for at least one block. Both of these problems become more apparent when more transactions are happening on the Bitcoin blockchain.

Off-chain scaling

To address these issues, off-chain scaling constructions were proposed. These constructions are usually based on the techniques enabled by the Bitcoin smart contract language, *Script*. They are often referred to as *Layer 2*, because they operate on top of the Bitcoin on-chain transactions (referred to as *Layer 1*). These solutions operate on top of Bitcoin the same way TCP/IP stack operates on top of the Internet layer.

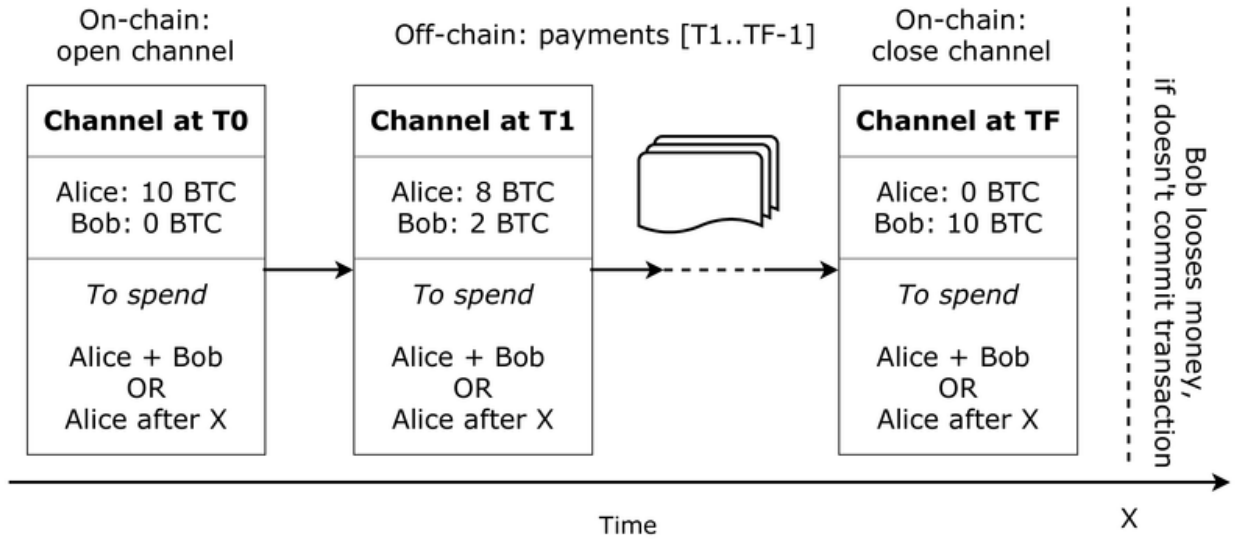
The security of off-chain protocols in Bitcoin differs from the security of the Bitcoin protocol because at least one of the following holds:

- they introduce an extra assumption about trusting third parties (e.g., a federation of operators)
- users of the protocol are assumed to timely react to base layer updates

In the Section 2.4, we further discuss the second assumption, and later use it as a basis for the attacks we demonstrate.

Payment channels

The high-level idea of payment channels, which were first suggested[3] by the creator(s) of Bitcoin in 2011, was to cache transactions between the peers (payer and payee in this context) instead of committing every transaction to the Bitcoin blockchain. Even though the described design was not secure, the high-level idea was further evolved and payment channels are now used in off-chain scaling protocols.



Spilman's channel. The state can only move forward (moving funds to Bob), when Alice gives Bob a signature for a new state.

Spilman's channel [3], or basic unidirectional payment channel (see Fig. 1) between two parties would work as following:

1. At time $T0$, Alice creates a transaction $TX0$, which locks $N0$ BTC. The transaction can be spent either collaboratively by Alice and Bob, or by Alice's only signature after X hours are passed.
2. At time $T1$ (less than X hours after $T0$), Alice constructs a transaction $TX1$ spending $TX0$, paying Bob $N0 - N1$ BTC and Alice $N1$ BTC. Both parties store transaction $TX1$.
3. At time $T2$ (less than X hours after T0), Alice creates a transaction $TX2$ spending $TX0$, paying Bob $N0 - N2$ (should be larger than $N0 - N1$) BTC and Alice $N2$ BTC. Both parties store transaction $TX1$

4. Now, before the X-hours time lock expires, Bob must commit the latest channel balance on-chain, in order to receive his funds.

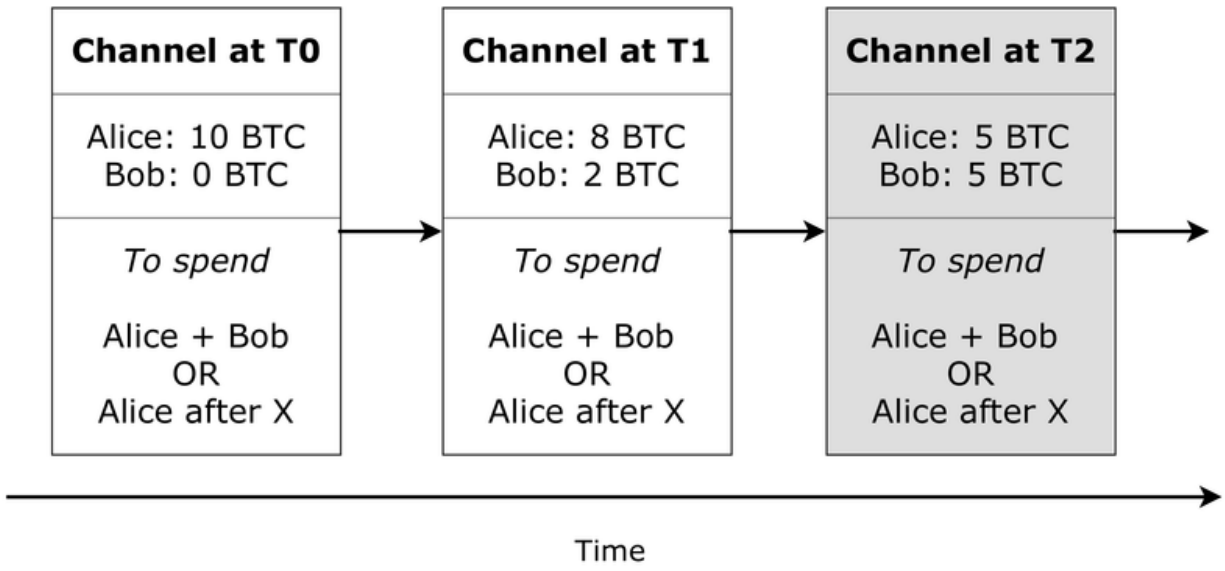
With this design, the worst thing may happen to Bob is missing the deadline X, and losing his funds because now Alice can finalize any state, including those, where she has all the money. The worst thing which may happen to Alice is getting her funds locked if Bob disappears. Due to the timelock, the funds may be locked for a limited, pre-defined time. These scenarios demonstrate that this protocol relies on the notion of time more than what's expected from the regular Bitcoin transactions.

This idea was further advanced with Decker-Wattenhofer [4], Poon-Dryja [2], and Decker-Russell-Osuntokun [5] mechanisms of updating a state of a channel with a goal to make channels bidirectional without making any trust assumptions on counterparty behavior. Since in a bidirectional case either participants may loose funds if a non-recent channel state is broadcast, both of participants have to bear the liveness requirement.

Another direction of research is enabling multi-hop payments, so that parties which do not have a direct payment channel can still transact. The most studied and currently used approach employs routing via Hash Time Locked Contracts. As an alternative, it has been proposed to use Point Time Locked Contracts for better confidentiality [6][7]. Now we will describe the protocols behind the Lightning Network, the most widely used layer-two Bitcoin scaling solution, currently based on the Poon-Dryja update mechanism and Hash Time Locked Contracts for routing.

Lightning Network

The most popular implementation of payment channels in Bitcoin is Lightning Network [2]. This system is designed so that independent payment channels between users form a network, where the users can transact both ways without necessarily having a shared channel, but through multi-hop payments.



A limitation of the basic channel design: single direction. Alice can never be sure she can finalize T2, because Bob can attempt to finalize T1 at any time.

The fundamental disadvantage of the design presented in Section 2.3 is inability to enable **bidirectional channels**, because even if Bob agrees to move funds towards Alice, he can always finalize the channel with a state, preceding the one where he paid Alice (see Fig. 2). In other words, there is no way for Alice to ensure that Bob won't revert his payment.

Another fundamental disadvantage of the Spilman's channel is the time bound: the channels can securely (for Bob) exist only until the initially chosen timelock expires. That timelock cannot be chosen too far in the future, because then Alice won't is at risk of locking her funds for too long, in case Bob is uncooperative.

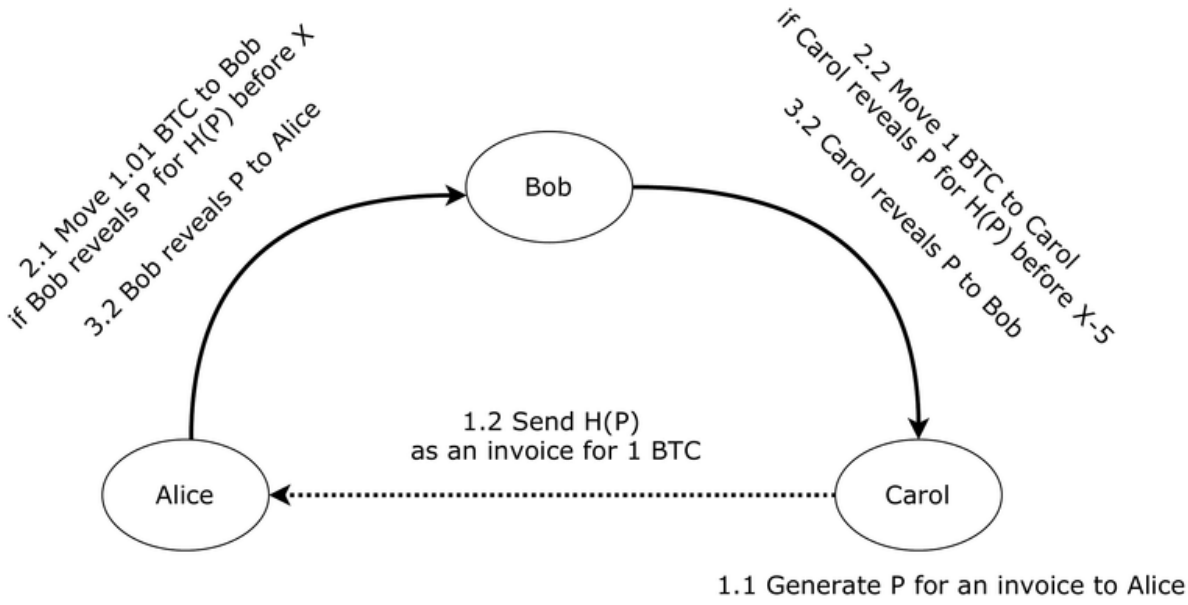
LN currently copes with both of these disadvantages by modifying the payment channel construction with **Poon-Dryja revocation mechanism**.

As we demonstrate in fig. , proceeding with the new state now reveals the secret, which makes the previous state invalid. If a malicious actor then decides to commit an outdated state on-chain, an honest user has a time window, during which they can "punish" the malicious actor for being dishonest and take all funds from the channel.

On the other hand, **multi-hop payments** are enabled in the LN by routing payments by **Hash Time Locked Contracts (HTLCs)**, as we demonstrate in fig. .

As its name suggests, an HTLC is built up from two primitives: a timelock and a hashlock. The contract semantics of an HTLC can be understood as "if a preimage P is

provided such as $hash(P) == H\_lock$, before timelock expiration T, increase the balance of the specified receiver".



Routing payments in Lightning Network.

While routing the payment through LN, a hashlock is going to be same value for the whole payment path. A timelock is going to decrease at every hop from payer to payee. Every multi-hop payment then would consist of three phases (see Fig. 3):

1. A payee sends an invoice of a payer, which would contain a hash to a preimage chosen by the payee.
2. Route setup, when every party agrees with the next hop to add a HTLC on their local channel sequentially in every channel, ordered from the payer to the payee.
3. Settlement phase, where every party agrees with the previous hop to remove the HTLC, once the preimage is known to the channel participant from the payee side. This way the preimage is propagated all the way to the first channel in the chain.

One notable detail relevant for the attacks is the timelock enforced by the user, via whom the payments are being routed. To allow a user in the middle to claim all the funds they have in a non-cooperative case, the user enforces the HTLC on the sender side be several blocks longer than on the receiver side. Along a payment path, timelocks are sorted in a decreasing order from payer to payee, to allow every intermediate router extra time for enforcing the smart contract rules.

Extra assumption

The cost of these solutions is a new assumption: a user should always have access to the recent blockchain history and should be able to broadcast transactions, in case of counterparty misbehavior.

More fundamentally speaking, LN introduces new security parameters. Instead of measuring the finality of the transactions with confirmations (a number of blocks after inclusion in the blockchain), security of payments in the LN can be measured in the chosen timelocks. The longer funds are locked in a channel, the better chance an honest user has to act on the misbehaviour of a counterparty, and get their funds back from the channel. At the same time, it makes the protocol less flexible in the case of an honest unilateral close triggered by an irresponsive counterparty.

Now that we described how LN works, it becomes apparent that processing the Bitcoin blockchain timely is required to transact in Lightning. The required monitoring can be done via running a full Bitcoin node, by relying on a trusted third party or, by using a *light client* (see Section 4.1), which is currently the most popular option.

In the next Section we will provide the background on the relevant attacks on Bitcoin, which is required to understand time-dilation attacks on the Lightning Network.

Attacks on the Bitcoin peer-to-peer network

All Bitcoin nodes, which participate in validation and propagation of blocks and transactions, constitute a peer-to-peer network.

Full Bitcoin nodes can be roughly split into two categories:

- reachable from most of the Internet and accepting inbound connections from other nodes
- non-reachable nodes behind NATs and firewalls

Reachable nodes act as a backbone, allowing other reachable and non-reachable nodes to join and relay transactions, blocks, and other necessary information.

As of March 2020, every Bitcoin Core node by default maintains 8 outbound connections to relay transactions, blocks, and network addresses of other nodes; and 2 extra connections to relay exclusively blocks. All connections in the Bitcoin network are bidirectional: if transactions are relayed in one direction, they are also relayed in the other one. Connections relaying only blocks are harder to infer due to less privacy

leaks, so they are more robust, and supposed to secure stable block relay even under certain (non-infrastructure) attacks.

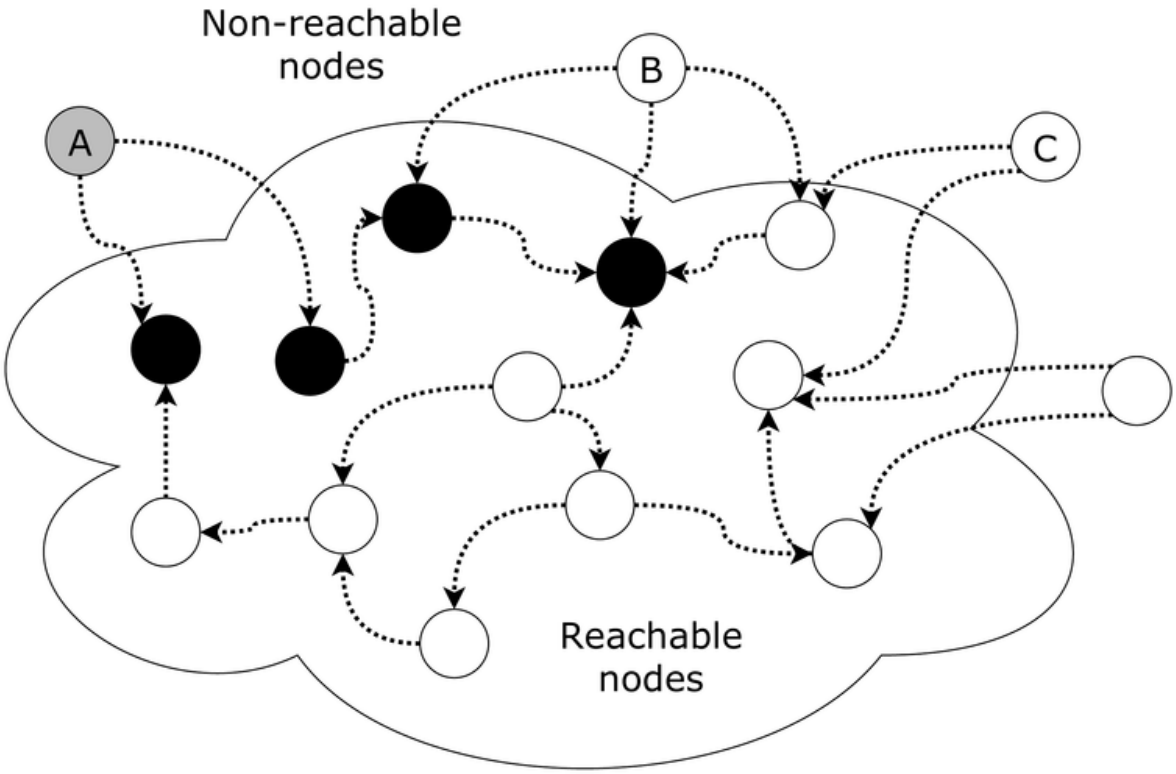
Although outbound peer rotation has been discussed multiple times [8][9], Bitcoin Core never deviated from the status quo approach, mainly because the alternatives were never thoroughly researched and analyzed. Due to this, the topology of the network is fairly static, and new outbound connections for an existing node are only made when there are issues with the existing connections.

Since the network is permissionless, it is naturally susceptible to a number of attack vectors. In this section, we will describe in detail two attacks vectors, which are relevant to time-dilation exploitation: eclipse attacks on Bitcoin and transaction origin inference.

These attacks are relevant to attacks on Lightning Network users, which choose to run their own full nodes or light clients for Bitcoin blockchain processing, instead of relying on third parties. We will first describe the current state-of-the-art relevant attacks targeted at full nodes, and then discuss applying them to light clients in more detail (see Section 4).

Eclipse attacks on Bitcoin full nodes

By definition, an eclipse attack implies preventing a victim's node from communicating with other honest participants of the network. It is usually done by occupying all of the victim's node connections by nodes or pseudo nodes belonging to an attacker. As a result of an eclipse attack, an attacker gains a complete control over *what* and *when* the victim sends and receives from the network, which is a crucial requirement for performing time-dilation attacks.



Eclipse attack on the Bitcoin network. Only node *A* is eclipsed, because all its connections lead to attacker.

Fig 4 demonstrates how an eclipse attack looks at the network topology level.

First eclipse attack on the Bitcoin network was demonstrated by Heilman et. al. [10], and was based purely on the high level protocols of the Bitcoin network, namely address management and relay logic. Further research [11][12] demonstrated that using a combination of BGP protocol exploitation and Bitcoin's address management can significantly reduce the cost of eclipsing a Bitcoin node.

Although a number of countermeasures (including suggested in 1,2,3) were integrated into Bitcoin Core and are believed to significantly reduce the attack cost, there was no study to demonstrate that it is not practical anymore to eclipse a Bitcoin nodes.

The studied consequences of eclipsing a Bitcoin node include monetary (double-spending attacks, attacks on mining) and non-monetary (enabling peer-to-peer layer deanonymization). Although Heilman et. al. briefly mentioned monetary consequences on second layer protocols [13], our work is the first to highlight that stealing funds in this case can be done without any hashrate and in an undetectable and practical way.

Transaction origin inference

Another building block of the attacks we demonstrate is linking transactions to the originating node. More specifically, an attacker has to find a Bitcoin node, from which a particular transaction was initially relayed. This would allow to link a transaction to a particular IP address, assuming a transaction sender uses their own node to submit transactions. We anticipate that this is a fair assumption for LN users which prefer a trust-minimized model compared to relying on a third-party backend.

It is possible that a transaction was relayed via a proxy node or Tor, in which case it would trigger a false positive observation, but this is currently not the default behavior and not the general case. Only 5 of 17 popular wallets[4] have the Tor feature. Furthermore, whether Tor should be used to do initial transaction relay by default remains an open question, due to the issues with Tor itself [14].

Transaction origin inference was previously explored in [15][16][17][18]. Most of the demonstrated attacks are a form of a Sybil attack and use the first-spy estimator. First-spy estimator technique relies on the assumption that the node which announces a transaction earlier than other nodes, is likely to be an originating node to the transaction or is directly connected to the originating node [16].

Countermeasures to the known transaction origin inferences techniques present in Bitcoin Core and the literature have two major drawbacks:

- They do not completely eliminate the risks of transaction inference, but only reduce the success rate.
- These solutions often focus on protecting against spying through inbound connections to victims, while spying by public well-connected Sybil nodes is less explored.

According to some of the experiments [18], an attacker can achieve 32-52% success inference rate when controlling 30% of the reachable Bitcoin nodes.

Based on these facts, we conclude that it is currently possible to infer transaction origin with a reasonable accuracy, although the requirements for an attacker grow with the maturation of Bitcoin protocol and the implementation.

Attacks on light clients

Several protocols have been proposed to reduce the requirement of running a full node and still use Bitcoin with a fairly trust-minimized model. All of them use client-server

architecture with multiple servers, assuming that at least one of the servers a client connects to is honest.

Light clients are often are used as a Bitcoin blockchain processing backend, notably on resource-constrained devices (like mobile phones). Because of that, understanding the security of these clients is important to evaluate the security of LN client implementations.

In this section we will demonstrate that every attack described in the previous section can also be carried against this class of Bitcoin nodes, and in many cases easier.

Light client protocols

One of the first non-standardized through BIPs (Bitcoin Improvement Proposal) implementation of the light client protocol is **Electrum**. Per this protocol, Electrum light clients connect to an Electrum server. Electrum server must have access to the chain processing backend, usually collocated on the same machine with the server.

Electrum itself provides configurations with different trade-offs. For example, an Electrum user can connect their light clients to Electrum Personal Server software run by themselves, or connect to multiple reachable ElectrumX Servers run by someone else.

Electrum is currently used as a Bitcoin chain processing backend by one of the most popular Lightning wallets, *Eclair*.

BIP 37 is historically the first light client standard specified with a BIP. This standard heavily uses Bloom filters: space-efficient probabilistic representations of a set [19] (in this case, Bitcoin addresses), with a possibility of false positives, but not false negatives. Per BIP 37, every light client will send a Bloom filter of addresses it is interested in monitoring to the full nodes it is connected to.

Then, those full nodes will be checking new confirmed transactions (from the newly mined blocks), against the filter and send only relevant transactions to the light client. More specifically, full nodes would make sure that all transactions, which send coins to or receive coins from the addresses specified by the Bloom filter are sent to the client.

In this case, the light client will receive only the transactions relevant to the client's address list (with some extra false positive transactions due to the Bloom filter probabilistic nature). To make it less obvious for the full nodes which addresses a light

client is interested in, a Bloom filter can be configured to have more false positives. This would, however, cause spending extra bandwidth on irrelevant transactions.

To the best of our knowledge, the only Lightning wallet which currently uses BIP 37 is *Bitcoin Lightning Wallet*.

BIP 157 was suggested as an alternative light client standard to provide a better privacy/efficiency tradeoff to its users. These light clients would connect to the full nodes in the network, receive a compact representation of Bitcoin blocks (filters, as defined in another related standard, BIP 158), and, if a filter detects relevant transactions on the client side, request a full block of transactions. Unlike BIP 37, this standard can operate with different types of filters, although the implementations we are aware of are based on Golomb-Coded sets (another probabilistic data structure [20], which is more compact than Bloom filters, but for the cost of longer queries).

Neutrino is currently one of the most popular light client implementations, and it is based on BIP 157. Neutrino is used by at least *Breez* and *Wallet by Lightning Labs*).

In the next sections we will demonstrate how the most popular implementation of BIP 157, Neutrino, is vulnerable to Eclipse attacks and transaction origin inference.

Eclipse attacks on Neutrino

It is crucial for BIP 157 light clients to be connected to the Bitcoin full nodes which provide the required server-side support by serving filters defined by BIP 158.

Currently, only one of the Bitcoin implementations (*btcd*) with 20-30 reachable nodes has released a build with the server-side support for these light clients. However, we found out that in addition to that, 20 nodes are running a custom version on Bitcoin Core, based on the work-on-progress implementation of server-side BIP 157 support.

According to our measurements, spawning 500 Sybil nodes with server-side BIP 157 support would allow trivially eclipse random 47% of newly deployed or restarted light clients.

The probability of a successful Eclipse attack can be then measured as

$$P_e = 1 - (attacker\_server\_nodes / all\_server\_nodes)^{out\_conns}$$

According to the equation, to improve the success of the attack, an attacker would have to either create more Sybil server nodes or reduce available honest server nodes. The former is a well-understood part of the threat model of the Bitcoin peer-to-peer

layer, and would require having a distinct IP per Sybil node. This requirement mak es it expensive to Sybil attack the whole Bitcoin network, but quite practical to attack the subset of nodes using BIP 157. The latter can be achieved via DoS, but is supposively detectable.

Additionally, an attacker might exploit that neither Neutrino nor btcd implement countermeasures discussed in Eclipse, Erebus papers, and those used in Bitcoin Core. For example, they don't employ the following methods:

- peer selection diversification based on an Autonomous System a peer belongs too, which would make it more difficult for an attacker to get victims exclusively connect to attacker's sybil nodes
- eviction on inbound connections when all the slots are occupied, which would allow new nodes to connect to honest reachable nodes even when their inbound capacity is exhausted.

Lack of these and other countermeasures allows more sophisticated eclipse attacks to succeed even at a lower cost.

In Section 9.1, we discuss additional measures required to make this attack more difficult.

Transaction origin inference on Neutrino

As it was described in Section 2, network-level transaction deanonymization in the suggested threat model usually relies on establishing multiple connections to honest nodes in the network and analyzing the messages coming from those nodes (first-spy estimation).

Research on this topic led to several techniques employed by the Bitcoin Core implementation to obfuscate transaction flow across the network. These include:

1. Random "diffusion" delays before announcing a transaction to peers
2. Increased diffusion delay for inbound connections
3. Shared diffusion delay timer for all inbound connections
4. Limiting connections to the same range of IPs

None of these are relevant to the Neutrino client, because those light clients relay only transactions submitted to them directly (from a corresponding Lightning node). Thus, it is enough for an attacker to make sure it has **at least one** direct connection from the Neutrino node of a victim.

As described in Section 4.2, Neutrino clients currently connect to a very limited number of public nodes. This allows an attacker with only 100 Sybil nodes to make sure that a victim is directly connected to an attacker at least once with 97

Attacks on Electrum light clients

Robustness to Eclipse attacks and transaction origin inference of Electrum light clients depend on the chosen mode of operation.

If an Electrum user runs their own Electrum Personal Server or ElectrumX Server connected to their own Bitcoin full node. In this case, since Electrum can be connected only to a full Bitcoin Core node, it inherits the security to Eclipse attacks and transaction origin inference from Bitcoin Core, partially described previously in this section.

If an Electrum user connects to ElectrumX Servers run by someone else, they face the same issues as Neutrino (as shown in the two previous sections): very low number of deployed nodes[5] makes it easy to eclipse honest users and infer their transactions.

Threat model

This work is built on top of the following assumptions about the Bitcoin peer-to-peer layer security:

- Honest users run unmodified Bitcoin and Lightning node software
- The network has liveness: blocks are constantly being mined
- The network provides confirmation-based safety of transactions
- The network generally forms a connected graph, except for the single nodes eclipsed by an attacker

When it comes to the capabilities of an attacker, we consider the following:

- An attacker does not control any hash-rate
- An attacker can deploy hundreds of sybil nodes with modified Bitcoin node software and analyze the messages coming from those nodes
- An attacker can create connections to, and accept connection from honest nodes
- An attacker can create and selectively relay transactions to other nodes in the network
- An attacker cannot influence initial bootstrapping of the honest nodes
- An attacker cannot force honest users to establish connections to the attacker, unless it happens according to the existing protocols

- An attacker cannot excessively abuse Internet infrastructure (e.g., does not directly cooperate with a victim's ISP), beyond the capabilities used in prior art (Erebus, Hijack-BTC)

This threat model allows an attacker to execute underlying attacks (eclipsing, transaction origin inference), which make possible the time-dilation attacks we discuss in this work.

Attack preparation

In this work we consider two practical scenarios for these attacks:

C1. A victim's Bitcoin node is first eclipsed as a part of a broader attack on the Bitcoin network, and an attacker attempts to find a corresponding Lightning node.

C2. A specific victim's Lightning node (identified by IP and the channels) is being targeted, and then an attacker attempts to locate and eclipse a corresponding Bitcoin node.

In both cases the attacker would have to eclipse the victim's Bitcoin node. We already demonstrated this is practical as of today in Section 4

In this section we will focus on two novel problems: verifying that a node is eclipsed and mapping a Lightning node to a Bitcoin node, both of which are also relevant in both cases.

Verifying Eclipse

Once an attacker suspects that a victim Bitcoin node is eclipsed, an attacker should verify that a node does not have another form of access to the Bitcoin network (e.g., after configuring extra outbound connections).

The easiest way to verify a node is eclipsed is **transaction probing** based on current transaction relay protocols. An attacker chooses a random transaction they received from the network, and not relaying it to the eclipsed node through any of the connections under control. Then, if a victim's node announces that transaction to an attacker, it means there is still a link between a victim and an honest part of the network.

It would be more difficult, however, if a node is connected to an external source of *blocks* (e.g., block-relay-only connections[6]), which do not relay transactions. In this case the proposed methodology would identify eclipsing only the transaction relay

aspect of the peer-to-peer communication, while this attacks require eclipsing all links relaying blocks.

In this case an attacker would have to apply **block probing**: delaying a block delivery through all links to the victim, and observing whether a victim relays that block to the attacker nodes. The only problem with this approach is that blocks arrive much less often than transactions. Thus, if the probing demonstrated that the victim is still not eclipsed, the next attempt will be possible not earlier than in 10 minutes on average (as opposed to every second with transactions).

Sometimes a victim may have an external source of blocks or transactions (e.g., Blockstream Satellite[7]). In this case, **a combination of block and transaction probing** would help an attacker to timely identify this is the case, deduce what kind of external service a victim is using, and whether an attacker is capable of disrupting it. This would ultimately help the attacker to choose a better strategy for proceeding with an attack (or abandoning it).

It is also possible for an attacker to do **inter-layer probing**, where eclipsing of a Bitcoin node is verified through communicating with a Lightning node. More specifically, an attacker withholds block X delivery to a victim, and sends a message related to the block X to the victim's Lightning node. If the victim's Bitcoin node is eclipsed and never received block X, the message would be rejected by the victim's Lightning node in an observable way.

Inter-layer probing is currenly not necessary, because it has the same properties as block probing. However, it would help the attacker if, due to a protocol change or software modification, a node does not relay blocks to the peers.

Mapping nodes

The easiest mapping technique would be to correlate Bitcoin and Lightning nodes which **operate under the same IP**. To measure how many users run their nodes under the same IP, we first scraped IP addresses of the Bitcoin nodes over a week, and attempted to correlate them to the list of lightning nodes with public (advertised) channels.

We were able to gather a list of 4,500 Lightning nodes and 52,000 Bitcoin nodes, and found 982 matches by IP. Notably, almost half of those Lightning nodes were represented by an onion address, making them even less likely to be traceable by this methodology. Only two pairs of nodes shared the onion address. These numbers do not

include Lightning nodes with *private* (not advertised) channels and non-listening Bitcoin nodes.

If Bitcoin and Lightning nodes **operate under different IPs**, an attacker would have to apply heuristics, some of which we will now discuss.

In case **C1** an attacker would have to find which Lightning channel funding transactions originated from the victim's eclipsed Bitcoin node, and map those transactions to channel announcements in the Lightning network. The most straightforward approach is to apply transaction origin inference against Lightning-related transactions coming from the victim's Bitcoin node. It can provide precise results because an attacker can analyze all the relevant messages coming from/to the victim node, acting as a Man-in-the-Middle between a victim and the honest part of the network.

Alternatively, an attacker can withhold a block from the victim's eclipsed Bitcoin node, and look for the nodes in the Lightning Network which don't accept and relay some of the channel announcements, which become valid *within the withheld block*.

For case **C2**, an attacker would have to:

1. Deploy sybil nodes in the Bitcoin network, both connecting to honest nodes and accepting connections from honest nodes
2. Apply transaction origin inference to the relay of the Bitcoin transactions corresponding to the victim's channels

In both cases, the approach involving transaction origin inference might take days or even weeks, if victim keeps not committing any channels on-chain.

Other techniques for correlating Bitcoin and Lightning nodes include timing analysis of nodes bootstrapping/restarting, or forcing a Lightning node to close a channel to speed up transaction origin inference mentioned above. We will leave this research for future work.

Time-dilation and the attacks

After the node is eclipsed, an attacker has to perform time-dilation: slowing down block delivery to the victim's Bitcoin node. Time-dilation is possible because, as we discussed in Section 2, block mining is an exponentially distributed process.

For example, even though it is expected to see blocks every 10m on average, it is expected to see 7 blocks which take longer than 30m every day. As of today, no dedicated countermeasure is implemented to let a victim distinguish between deferred block propagation and a random event.

This undistinguishability from normal operation enables an attacker to trivially make a victim behind in time, in terms of the knowledge about the latest block. An attacker has to simply introduce a delay between receiving a block and feeding it to a victim. Since the victim is eclipsed and doesn't have honest source of blocks, an attacker can single-handedly decide, when a victim receives a new block.

There is, however, one Bitcoin Core feature, which unintentionally sets the upper bound (30m) for undetectable delay per block. We explore that feature how the upper bound affects the attack strategy in Section 8.

Once the victim's Bitcoin node is confirmed to be eclipsed and an attacker is able to slow down block delivery to that node, an attacker can start launching attacks based on time-dilation. In the following descriptions of the attacks, pseudonyms "Alice", "Bob", "Carol" and "Mallory" represent users of the Lightning Network.

	Implementation	CSV delta	CLTV delta	Incoming HTLC Onchain Claiming Policy
	C-lightning	144	14	7
	LND	144-2016	40	10
	Eclair	720	144	11
	Rust-lightning	144	72	6

[tab:config_impl] Default timelock (in blocks) configurations for various Lightning implementations.

A1. Targeting Channel State Finalization timelock

This attack targets payment channels between an honest Lightning Network user (Alice) and an attacker (Mallory).

Let's say Alice and Mallory have a payment channel, which was opened within a block of height N. The channel is configured with a CSV timelock of M blocks for contestation (as per the channel design discussed in Section 2.4). The default choice of M in major LN implementations is summarized in 1.

To start exploiting an attack, Mallory should make Alice be M blocks behind the actual tip of the blockchain by performing time-dilation. As a result, Alice's block height is pinned at $P - M$, where P is height of the actual latest block in the network.

Once the difference in heights is achieved, Mallory can effectively double-spend Alice. To do so, Mallory negotiates with Alice a new state. Per this new state, Malory, for example, pays Alice and receives something (in an irreversible way, like a physical or digital good) from Alice. Then Alice commits the previous state on-chain with a balance near-channel-value paying back to her. Malicious revoked commitment transaction is committed at $P + 1$.

Since the latest block Alice has corresponds to time $P - M$, she won't detect the channel revocation until reaching $P + 1$. At that time, honest network and Mallory are already at height $P + M + 1$. The contestation period is expired for the rest of the honest network, and the "malicious" spend is fully valid.

A2: Targeting Per-Hop Packet Delay timelock

This attack is based on exploiting the HTLC-based routing through Bob, where Alice and Carol are two malicious entities of the same attacker having channels established with Bob.

The attacks starts with two lightning channels being opened: Alice-Bob and Bob-Carol. Bob enforces a $cltv\_delta$ (see Section 2.4) of M blocks on incoming HTLCs. We summarize how different LN implementations choose the $cltv\_delta$ in Table 1. Carol enforces

Alice and Carol eclipse Bob's Bitcoin node and perform time-dilation until they gain a lead of $M + 1$ blocks on Bob.

Once Alice and Carol have managed to be $M + 1$ blocks ahead of Bob, they route a payment through him with a final timelock delta of N. On Bob-Caroll channel HTLC timelock expires at $H + N$. On Alice-Caroll channnel HTLC timelock expires at $H + M + N$, therefore satisfying Bob's $cltv\_delta$ of M.

Once the actual Bitcoin blockchain tip is at height $H + M + N$, Carol provides a required preimage to Bob, and gets from Bob a signature for a new state.

At the same time, Alice finalizes the state of her channel on the Bitcoin blockchain and broadcasts a HTLC-timeout transaction to get back the offered payment. This prevents Bob from re-negotiating the state of that channel via the preimage he just got against a full-payment to Carol, making him effectively robbed.

A3: Targeting Packet Finalization timelock

This attack is based on exploiting the incoming HTLC safety delay on a channel.

When a party knows the preimage for an incoming HTLC but remote peer doesn't reply timely to update channel state, the party goes onchain to claim the incoming HTLC for which its preimage is known, I blocks before expiration. Incoming HTLC onchain claiming policy for each LN implementations is defined in Table 1.

Mallory, the attacker, starts by time-dilating Bob, the victim, by $I + 1$ blocks. Attacker waiting an offset of one compare to policy is to avoid broadcast race condition between a honest HTLC claiming transaction and a malicious timeout.

At H, an HTLC is routed from Alice via Mallory to Bob and will expire at $H + N$, with N final timelock delta. There is no collusion between Alice, a honest payer, and Mallory. Bob reveals the preimage to Mallory, and Mallory deliberately doesn't reply back to update channel state. When blockchain tip reaches $H + N$ on the non-eclipsed network, Mallory broadcasts her HTLC-timeout, therefore making revealed preimage invalid to claim offered HTLC on the Mallory-Bob channel.

Finally, when Bob reaches $H + N - I$ on his blockchain view, he attempts to claim the incoming HTLC by broadcast a preimage transaction. This one is going to be rejected by other network peers, HTLC output has already been finalized by Mallory's HTLC-timeout transaction. Then, Mallory claims offered HTLC on Mallory-Alice channel, presenting Bob's preimage, therefore earning a routed payment for which she hasn't send fund forward.

This attack differs from the previous one because here attacker only needs one channel with victim, but needs to be selected for a payment path. It also differs in a way an attacker finalizes the channel on-chain while stealing funds, i.e by timing out stolen HTLC.

Evaluation

The practicality of time-dilation attacks once the node is already eclipsed can be measured in *the time it takes to perform them* and *failure rate*. The timing aspect is relevant because if the attack takes several days, it can be more easily disrupted by a random event (e.g., a scheduled restart making a victim node connect to new peers).

Both of these metrics depend on the countermeasures software uses to disrupt time-dilation attacks. We previously mentioned that there is currently no *dedicated* countermeasure implemented for this purpose. In this section we explore how one mechanism, origininally designed for another purpose, bounds the practicality of time-dilation attacks.

The only heuristic employed by Bitcoin Core implementation which may help to break free from eclipsing is *potential stale tip detection*. If a block hasn't arrived during the last 30m, a node attempts to establish one extra outbound connection and sync tips with a new peer, and then repeat it in 10m if a block was not found during that time. This feature was originally introduced to handle *non-malicious* failures of honest nodes to provide latest blocks.

This countermeasure does not guarantee mitigating the eclipse attack, because it is possible that a victim's chosen extra outbound connections will be the attacker's sybil node.

For example, an attacker can degrade the effectiveness of this extra connection by poisoning the victim's address manager while the node is eclipsed. Although it was demonstrated to be impractical with current attacks, it shoukd be more feasible when a node is eclipsed.

Because of this, we will further refer to the probability of this event as *failure rate*. It will represent a baseline, upon which an attacker can improve.

Modelling optimal attack strategy

To fully eliminate the possibility of victim's Bitcoin node de-eclipsing due to the stale tip detection, the attacks we demonstrate would never intentionally trigger potential stale tip detection. An attacker should never exceed 30m delay between delivering blocks. It is possible, however, that stale tip detection may be triggered naturally by the randomness of the block mining process. According to our estimates, this happens with a probability of 5% ($e^{-(30/10)}$), so on average 7 times a day.

The optimal strategy for an attacker in this case would be to delay every block by 29.5m. This approach works best because it allows the fastest time-dilation without triggering stale tip check. At the same time, it reduces the probability of "natural" de-eclipsing as much as possible, because an attacker accumulates the time which can be used to amortize naturally slow (>30m) blocks in the most efficient way. If an attacker does combine it with address manager poisoning, the delay can be increased.

We created a simulation-based model accounting for exponential nature of block generation and stale tip detection. In our model, we simulate a generation of 1,000 blocks, and model an attack per which an attacker delays *every* block by a constant chosen interval of 29.5m, so that the stale tip check is never triggered. We repeat this experiment 100,000 times for every configuration we explore.

Results

According to our model, the time it takes to become ahead of a victim by 144 blocks is 36h. We summarize the estimated time of keeping a node eclipsed required to perform time-dilation attacks based on the configurations of Lightning Network implementations in Tables 2, 3. These results are based on the configurations presented in Table 1.

	Implementation	Eclipse time (N)	Eclipse time (BC)	
	C-lightning	24h	36h	
	LND	24-336h	36-508h	
	Eclair	120h	182h	
	Rust-lightning	24h	36h	

[tab:timings_a1] The time a node has to remain eclipsed to allow attack A1 on various Lightning implementations with Bitcoin Core and Neutrino backends.

	Implementation	Eclipse time (N)	Eclipse time (BC)	
	C-lightning	2.3h	4h	
	LND	6.6h	10h	

	Eclair	24h	36h	
	Rust-lightning	12h	18h	

[tab:timings_a2] The time a node has to remain eclipsed to allow attack A2 on various Lightning implementations with Bitcoin Core and Neutrino backends.

These results confirm the following intuitive formula: *Given targeted timelock to break and his eclipsing capability for a time-period, an attacker can compute block slow-down rate (in hours)*: $Y = b * X/(Z * W)$

However, it is possible that the stale tip check will be triggered naturally, even under the most optimal attack strategy. It would happen when it took *very long* to mine a block. In this case, the *delayed delivery time* of a particular block will be behind the actual block generation time. According to our model, with a chosen strategy of delaying for 29.5m every time, the probability of this natural de-eclipsing (attack failure rate) is around 7%.

Intuitively, the probability of successful de-eclipsing via the stale tip check rapidly goes down with every maliciously delayed block. For a first block to trigger a stale tip detection (while a node is under attack), the natural mining time of that block should exceed 30m, while for a fourth block it should exceed 80m. The probability of these events are 5% and 0.03% respectively.

Since Neutrino does not implement stale tip detection, there is no such upper bound, and the time it takes to dilate a node by a chosen number of blocks is constrained only by the natural time to produce those blocks. At the same time, without this check the attacks on Neutrino *always* succeed.

If an attacker had (or chose) to use 19.5m delays instead of 29.5m, it would increase the attack failure rate from 7% to 22%, while also increasing the time it takes to perform time-dilation from 25-32h for reaching a difference of 100 blocks.

One way to force an attacker into this for Bitcoin Core developers is to reduce the stale tip threshold from 30m to 20m. This would, however, also significantly increase false positives. Currently this stale tip warning is already triggered on average 7 times a day naturally (not under attack), and with this change it would be 20 times a day. Since this would make stale tip detection even less effective for its original purpose, we do not recommend modifying it to be used against time-dilation.

In the next section we will discuss what additional measures can be taken by the protocol and implementation developers to significantly increase the cost of time dilation attacks.

Countermeasures

We split countermeasures into preventing time-dilation itself, preventing the exploitation of it, and the issues related to reacting once the exploitation is detected.

Preventing Time-dilation

Preventing eclipse attacks on Bitcoin nodes would make time-dilation impossible. The cost of Eclipse attacks can be increased via the following measures.

Increasing connectivity and the number of honest reachable nodes. As it was shown in Section 4.2, the probability of Eclipse attacks goes down when any of these two parameters are growing. There are effectively three ways to achieve this:

- encourage users to provide more resources (bandwidth, computational) to the network
- make the use of those resources more efficient
- increase adoption and deployment of BIP 157 (especially server-side) across the ecosystem

Increasing peer diversity. Since both eclipse and topology origin inference often involve an attacker accepting connections from victim nodes to, increasing the cost of sybil attacks is an effective countermeasure. If honest nodes decide which nodes to connect to based on some scarce property, it would make it more difficult for an attacker to influence those decisions. Complementing anti-Sybil mechanisms (e.g., peer diversification based on the peer's Autonomous System) with proactive topology improvements through peer rotation is likely to help breaking-free from ongoing eclipse attacks [12].

Increasing link layer diversity. Since the attacks we demonstrated employ exploiting the peer-to-peer layer, a natural countermeasure to them would be adding a redundant method of communication: using several interconnected multi-homed nodes, a VPN, a mesh network, or the Lightning Network itself for block and transaction relay. If any of these methods are employed to receive blocks and transmit transactions, an attacker would be required to disrupt those as well. In case of

bandwidth-constrained communication channels, transmitting block headers would be enough to detect an anomaly.

Instead of adding redundancy at the p2p layer, LN clients may serve each other to increase their security in a web-of-trust-style deployment. "Friendly" client could be asked to watch a list of outputs spending belonging to another client and notify this one in case of a match against their filters. Therefore an attacker would have to control all chain providers of every client part of the swarm. Given resources and incentives for the watching client and the privacy leak for the beneficiary client, this scheme would only lay on some social trust assumption (like a set of mobile wallets belonging to a family).

Preventing transaction origin inference has been shown to be non-trivial. All the mentioned above can both increase (if the relayed information gets to the honest non-tracking nodes faster) and reduce (because they reveal more data about the node) the cost of transaction origin inference.

Peer-to-peer protocols anonymity is a standalone research topic. Integrating ideas from prior work [17][18] into Bitcoin Core, as well as improving on the existing features may make time-dilation attacks impractical.

Although **running Bitcoin nodes over Tor** was demonstrated to be vulnerable to certain attacks [21], other designs of transaction relay mechanisms involving various mixnets should be explored.

Preventing Time-dilation Exploitation

If an attacker was successful to make all the preparations required to execute a time-dilation attack (listed in Section 6), there are countermeasures which can be integrated to disrupt the attack.

Detecting time-dilation. Although in Section 8 we demonstrated that the current stale tip detection technique is limited against time-dilation attacks, implementing a specialized time-dilation detection could be useful.

It is possible to use local system clock to detect the absence of new blocks at an unlike-enough interval. Alternatively, time present in the header of a mined block may be used, although this field is only moderately enforced by consensus, preventing accidental block invalidation due to system time lagging of a miner. Local clock of the

potential victim can also be a subject to manipulation or system errors. These methods can be expanded to consider series of blocks instead of just one block.

All these solutions, unfortunately, have a fundamental trade-off: security against false-positive detection rate.

WatchTowers approach imply that chain monitoring is replicated with different computers, each of them maintaining the channel state [22][23][24][25]. Every watchtower may rely on a different full-node, each connected to a different set of peers and running on different ISPs, which considerably increase the bar for a successful eclipse attack.

Lightning protocol-level warnings in case of the observed anomalies may help a node operator identify that a node is currently under attack. For example, if a Lightning node receives channel announcements related to the blocks “from the future”, it should at least issue a warning.

Similarly, **abnormal failure rate detection** may be used. If a Lightning node is behind in terms of the Bitcoin blockchain view, but Lightning payments between honest nodes are still flowing through it, this node will have a high routing failure rate. This would happen due to the fact that honest nodes on the routing path would reject the forwarded HTLC for being too close to be expired. This observation can be used to detect time-dilation.

Reaction

Even if an LN node detected it is under a time-dilation attack and it’s not too late, it still cannot easily prevent the loss of funds. The issues with stopping the attack include:

1. If there are multiple channels opened, it is unclear which of them should be closed to prevent the loss
2. For a given channel, it may be unclear which state was committed by the attacker. This is relevant because this must be taken into account by a victim when constructing a justice transaction.
3. If fine-grained management is implemented, revocation key may be stored on a different (less "warm") storage than channel and HTLC key holder. In that situation, interaction with this storage should be included in security timelocks computation.
4. A justice transaction should have a proper fee and be transmitted to the miners via honest peer-to-peer network, or otherwise it won’t be confirmed.

The challenges (1), (2), (4) are critical in the context of a victim's Bitcoin node being eclipsed. Thus, even if the attack was detected, the only solution is to apply the same anti-eclipsing mechanisms we suggested above.

Discussion

Other time-sensitive protocols While the attacks we demonstrated were specifically targeted at the Lightning Network protocols, we believe that a wide variety of bitcoin protocols [26][27][28][29] may be susceptible to time-dilation attacks. This applies to any of them where timelocks are used to arbitrate between parties willing to commit concurrent onchain transactions. We believe that designers of those protocols should take these threats into account while arguing about their security.

N-hashrate attacker The attacks we presented do not require an attacker to control any hashrate. It is possible that having some minor hashrate would allow an attacker to execute the demonstrated attacks faster or more successfully, or exploit time-dilation in other ways. We leave this research for future work.

Combining attack with mempool spam The attacks we discussed may be prevented by a victim timely detecting it and submitting a revoke transaction timely. Due to the attack, a victim has to act in a very limited time-frame, less than the one anticipated in the original timelock. An attacker may make it even harder for a victim by running a DoS against Bitcoin, so that victim's transaction is not getting confirmed. Considering the high cost of known DoS-attacks, this may make sense only when several high-capacity channels are under attack. If LN implementations employ dynamic fee-bumping, this may help a victim to prioritize their transactions and overcome DoS.

Attacker controlling broader infrastructure The attacks we demonstrated work under limited capabilities of the adversary. If an adversary also controls the victim's ISP, or has ways to influence DNS responses, or have other ways to exploit the infrastructure, the attacks may be done at a much lower cost. It also makes countermeasures we suggested much less efficient.

Switching to Initial Block Download after 24 hours Although we mentioned that there is currently no dedicated mechanism for time-dilation detection, one relevant feature of Bitcoin Core software is switching to Initial Block Download. It happens if the time defined in the latest known block header is 24 hours behind current system time. This feature is not efficient against time-dilation attacks, because, as we demonstrate in this paper, those need to dilate is less than 24 hours. We do not

recommend modifying it to be useful in this context, because it was not originally designed to prevent attacks.

DoS risks due to anomalies detection Even though we suggested anomalies detection as a safety mechanism, it should be deployed very carefully. Any wide scheme of anomalies detection may be used to trigger false-positives by an attacker and lure a LN node to unilaterally close its channels by safety prevention. Contrary to the base layer, where one may just lockdown their payment system, under Lightning security model, not acting is a risk of money loss on its own.

Explore tradeoff between higher-security and funds liquidity One way to make time-dilation attacks harder is to increase timelocks. This would require an attacker to keep a victim eclipsed for longer, and will give a victim more time to prevent the attack or react to it. Unfortunately, this makes more secure channels less dynamic, because in a non-cooperative case it takes longuer to settle them.

Going further, attack cost may be spread on exploiting all links of single-LN node, so sum of all channel values should be used to assess operational risks. Reasoning on time-value only is an incomplete method to argue about sophisticated attacks. We believe that every LN operator should separately consider their own risks related to time-dilation, in addition to the regular time-value trade-off.

Finding the proper balance between the systemic risk caused by *liquidity market for routing payments*[8] and security is a open area of research.

Related work

Attacks on the Bitcoin peer-to-peer network usually result in eclipsing honest nodes and transaction deanonymization.

The first Eclipse attack on Bitcoin was based purely on the high-level Bitcoin network protocols [10], while the latter exploited BGP and Internet infrastructure [11][12]. These attacks only briefly mentioned consequences on second-layer protocol [10], while focusing on reducing the stability of the network and making mining less efficient.

Even though after applying some countermeasures Eclipse attacks on Bitcoin Core became more expensive, it is not the case with light clients, on which LN impelemntations often rely.

Bitcoin was also considered as a target for attacks on NTP [30][31], although the consequences for second-layer protocols were also not explored in this case.

Attacks on the privacy of the Bitcoin peer-to-peer protocols demonstrated that transaction deanonymization is fairly feasible both with simple techniques like first-spy estimator and more advanced strategies [16][17][15][18][32]. These attacks focused on clustering transactions together or finding a transaction origin.

Some of the privacy improvements were proposed and deployed, but all of them only increase the cost of deanonymization. And, same as with anti-eclipsing, most of these countermeasures are not integrated in light clients.

The novel time-dilation attacks we demonstrate heavily rely on this prior work, because they require both eclipsing and transaction deanonymization.

Using Tor "natively" at the peer-to-peer layer was demonstrated to be an inadequate solution to these problems [21].

Attacks on the Lightning Network may be split in two groups: privacy-related or DoS-related.

Prior work on privacy mainly explored real-time balances of channels in the network via probing[33]. Known DoS attacks achieve a cheap congestion of the network and preventing the flow of honest payments [34][35][36].

It was also explored how an attacker can steal routing fees [7], and exploiting Bitcoin Core transaction propagation and mempool policies to get an advantage in LN settlement [37].

Conclusions

Even though Lightning Network have a promise of solving the scalability limitations of Bitcoin, it introduces new security assumptions. In this work we explored what can happen to an LN user, if their view of the Bitcoin blockchain is lagging behind due to an attack.

The three attacks we discuss for the first time demonstrate that it is possible to steal money from an eclipsed user without any hashrate. The attacks we demonstrate are practical and very hard to detect with the existing Bitcoin and Lightning software. Time-dilation in some cases allows an attacker to steal funds from Lightning channels after eclipsing a victim for only several hours.

We suggest a wide variety of countermeasures to increase the cost of the discussed attacks. We believe that the threat we discuss is fundamental to a broader family of second-layer protocols. The suggestions we make are relevant to both modern and future implementations of those protocols.

Acknowledgments

Footnotes

1. From https://luke.dashjr.org/programs/bitcoin/files/charts/software.html ↩

2. https://www.metzdowd.com/pipermail/cryptography/2008-November/014814.html ↩

3. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002417.html ↩

4. Listed at bitcoin.org as of 01/10/2020 ↩

5. 61, as listed at https://1209k.com/bitcoin-eye/ele.php as of 04/13/2020 ↩

6. feature introduced in Bitcoin Core 0.19 ↩

7. https://blockstream.com/satellite/ ↩

8. https://blog.bitmex.com/the-lightning-network-part-2-routing-fee-economics/ ↩

Citations

1. Nakamoto, S. (n.d.). *Bitcoin: A peer-to-peer electronic cash system, http://bitcoin.org/bitcoin.pdf*. ↩
2. Poon, J., & Dryja, T. (2016). *The bitcoin lightning network: Scalable off-chain instant payments, https://lightning.network/lightning-network-paper.pdf*. ↩
3. Hearn, M., & Spilman, J. (2013). *Anti dos for tx replacement*. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002417.html. ↩
4. C.Decker, & Wattenhofer, R. (2015). A fast and scalable payment network with bitcoin duplex micropayment channels. In *International symposium on stabilization, safety and security of distributed systems*. ↩
5. C.Decker, R. R., & O.Osuntokun. (2018). *Eltoo: A simple layer2 protocol for bitcoin*. https://blockstream.com/eltoo.pdf. ↩

6. Poelstra, A. (2019). *Lightning in scriptless scripts, https://lists.launchpad.net/mimblewimble/msg00086.html*. ↩
7. Malavolta, G., Moreno-Sanchez, P., Schneidewind, C., Kate, A., & Maffei, M. (2019). Anonymous multi-hop locks for blockchain scalability and interoperability. In *NDSS*. ↩
8. Naumenko, G. (2018). *Peer rotation, https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2018-may/016014.html*. ↩
9. Pustogarov, I. (2014). *Add rotation of outbound connections, https://github.com/bitcoin/bitcoin/pull/4723*. ↩
10. E.Heilman, A. Z., A. Kendler, & S.Goldberg. (2015). Eclipse attacks on bitcoin's peer-to-peer network. In *24th usenix security symposium*. ↩
11. M. Apostolaki, A. Z., & L.Vanbever. (2017). Hijacking bitcoin: Routing attacks on cryptocurrencies. In *IEEE symposium on security and privacy*. ↩
12. M.Tran, I.Choi, G.Moon, A.Vu, & Kang, M. S. (2019). A stealthier partitioning attack against bitcoin peer-to-peer network. In *IEEE security and privacy*. ↩
13. Marcus, Y., Heilman, E., & Goldberg, S. (2018). *Low-resource eclipse attacks on ethereum's peer-to-peer network*. Cryptology ePrint Archive, Report 2018/236. ↩
14. R.Jansen, T.Vaidya, & M.Sherr. (2019). Point break: A study of bandwidth denial-of-service attacks against tor. In *USENIX security symposium*. ↩
15. A.Biryukov, D. K., I. Pustogarov. (2014). Deanonymisation of clients in bitcoin p2p network. In *ACM conference on computer and communications security*. ↩
16. Venkatakrishnan, S. B., G.Fanti, & P.Viswanath. (2017). Dandelion: Redesigning the bitcoin network for anonymity. In *ACM sigmetrics international conference on measurement and modeling of computer systems*. ↩
17. G.Fanti, Venkatakrishnan, S. B., S.Bakshi, B.Denby, S.Bhargava, A.Miller, & P.Viswanath. (2018). Dandeliton++: Lightweight cryptocurrency networking with formal aonymity guarantees. In *Proceedings of the acm on measurement and analysis of computing systems*. ↩
18. G.Naumenko, G.Maxwell, P.Wuille, A.Fedorova, & I.Beschastnikh. (2019). Erlay: Efficient transaction relay for bitcoin. In *ACM conference on computer and*

communications security.↩
19. Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, *13*(7), 422–426. https://doi.org/10.1145/362686.362692 ↩
20. Golomb, S. (1966). Run-length encodings (corresp.). *IEEE Transactions on Information Theory*, *12*(3), 399–401. ↩
21. Biryukov, A., & Pustogarov, I. (2015). Bitcoin over tor isn't a good idea. In *2015 ieee symposium on security and privacy* (pp. 122–134). ↩
22. Dryja, T. (2016). *Unlinkable outsourced channel monitoring, https://milan2016.scalingbitcoin.org/transcript/milan2016/unlinkable-outsourced-channel-monitoring*. ↩
23. Khabbazian, M., Nadahalli, T., & Wattenhofer, R. (2019). Outpost: A responsive lightweight watchtower. In *Proceedings of the 1st acm conference on advances in financial technologies* (pp. 31–40). ↩
24. Avarikioti, G., Litos, O. S. T., & Wattenhofer, R. (2020). Cerberus channels: Incentivizing watchtowers for bitcoin. *Financial Cryptography and Data Security (FC)*. ↩
25. McCorry, P., Bakshi, S., Bentov, I., Miller, A., & Meiklejohn, S. (2018). *Pisa: Arbitration outsourcing for state channels*. Cryptology ePrint Archive, Report 2018/582. ↩
26. Dryja, T. (2017). Discreet log contracts. *URl: Https://Adiabat. Github. Io/Dlc. Pdf*. ↩
27. I.Kuwahara, T. N., T. Le Guilly. (2020). *Discreet log contracts channels and integration in the lightning network*. https://github.com/p2pderivatives/offchain-dlc-paper/blob/master/offchaindlc.pdf. ↩
28. Möser, M., Eyal, I., & Sirer, E. G. (2016). Bitcoin covenants. In *International conference on financial cryptography and data security* (pp. 126–141). Springer. ↩
29. Gibson, A. (2017). *Coinswaps, https://joinmarket.me/blog/blog/coinswaps/*. ↩
30. A.Malhotra, I.Cohen, E.Brakke, & S.Goldberg. (2015). Attacking the network time protocol. In *NDSS symposium*. ↩

31. J.Herrera-Joancomarti, G.Navarro-Arribas, Pedrosa, A. R., C.Perez-Sola, & J.Garcia. (2019). On the difficulty of hiding the balance of lightning network channels. In *ACM asia conference on computer and communications security*. ↩

32. Biryukov, A., & Tikhomirov, S. (2019). Deanonymization and linkability of cryptocurrency transactions based on network analysis. In *2019 ieee european symposium on security and privacy (euros&P)* (pp. 172–184). IEEE. ↩

33. S.Tikhomirov, R.Pickhardt, A.Biryukov, & M.Nowostawski. (2020). *CoRR, abs/2004.00333*. Retrieved from https://arxiv.org/pdf/2004.00333.pdf ↩

34. Pérez-Sola, C., Ranchal-Pedrosa, A., Herrera-Joancomartí, J., Navarro-Arribas, G., & Garcia-Alfaro, J. (2019). *LockDown: Balance availability attack against lightning network channels*. ↩

35. Rohrer, E., Malliaris, J., & Tschorsch, F. (2019). Discharged payment channels: Quantifying the lightning network's resilience to topology-based attacks. In *2019 ieee european symposium on security and privacy workshops (euros&PW)* (pp. 347–356). IEEE. ↩

36. Mizrahi, A., & Zohar, A. (2020). Congestion attacks in payment channel networks. *ArXiv Preprint ArXiv:2002.06564*. ↩

37. Corallo, M. (2020). *RBF pinning with counterparties and competing interest, https://lists.linuxfoundation.org/pipermail/lightning-dev/2020-april/002639.html*. ↩