Security and Privacy of Bitcoin's Lightning Network

Prince Yaw Gharbin - prince.gharbin.002@student.uni.lu
Yann Hoffmann - yann.hoffmann.002@student.uni.lu
Supervisor : Sergei Tikhomirov - sergei.tikhomirov@uni.lu
University of Luxembourg
Belval Campus, Luxembourg

ABSTRACT

As digital currencies enjoy renewed attention, Bitcoin takes once again center stage as the leading technological breakthrough of the last decade. Lightning Network draws its popularity from Bitcoin and holds the promise of increasing the speed and capacity of Bitcoin, and sustainably decreasing transaction fees. Lightning Network has succeeded in establishing itself as a reliable off-chain scaling solution for Bitcoin, allowing it to achieve commercial adoption and breadth. However, Lightning is not impervious to exploits, some of which can have dire consequences. We study a wide range of known vulnerabilities from time dilation to lockdown attacks and try to better understand how to guard against these weaknesses. Moreover, we implement a probing attack on Lightning channel and grapple with its consequences in the latter part of the paper.

KEYWORDS

Lightning Network; Bitcoin; Blockchain; Network; Security; Off-Chain Scaling; Payments Channels; Privacy; Channel probing

1 INTRODUCTION

Bitcoin is a compendium of rules and software specifications that gives rise to a distributed network for conducting financial transactions in an irreversible and anonymous manner. Bitcoin relies on the blockchain as its consensus mechanism where miners convert hash power and electricity into proof of work to confirm transactions. This innovative mechanism, which made Bitcoin unique when Satoshi Nakamoto first published the bitcoin whitepaper in October 2008 [10], is incidentally a significant speed bottleneck. For reference, SWIFT, the society for worldwide interbank financial telecommunications, is able to carry out transactions at a whopping rate of 33.6 million per day between 11,000 institutions [6] only limited by demand , whereas Bitcoin is closer to 604,800 transactions per day (55 times less). The main reason behind this is that blocks, which contain hundreds of transactions, are generated on average only every 10 minutes and have a fixed size of 1MB. The growing demand and restricted supply have brought about a natural market for transactions. Miners charge a transaction fee based on the demand for transactions at any point in time (the fee sits at 68 sat or 3 USc per byte of data as of January 2021).

Lightning Network (LN) addresses the problem of speed and capacity by letting users settle transactions among themselves thanks to entirely trustless channels with shared capital. The Lightning Network is a peer-to-peer (P2P) routed payment network running as an additional layer on top of the Bitcoin blockchain. Even though LN members transact in bitcoin currency (BTC), the exchanges themselves bypass almost entirely the blockchain – and hence the size limit. Two nodes on the network can create a payment channel by first committing a certain amount of BTC to a shared blockchain address. The amount of committed BTC defines the capacity of the channel. A payer and payee can then transact together even if they don't share a direct channel; they simply reroute the transaction through the Lightning Network. From there on, the Lightning Network becomes independent from the blockchain and unlocks astronomical transfer rates (theoretically as fast as your bandwidth can support) and only resorts to the blockchain in case of disputes. The concept of a channel is essential and will be further explored in the later sections.

To preserve some degree of privacy, LN uses an onion-routing protocol named Sphinx to chaperone multihop transactions. Nodes only publish the minimal amount of information required to establish the payment routes. One cannot escape disclosing the total capacity of a channel as it is broadcasted on the blockchain at the birth of a channel. However, the distribution of funds between two owners of the channel remains private; or so it should be. We will show in this paper how a nefarious actor can probe a channel and discover the distribution of funds unbeknownst to its users. This naturally poses severe threats to the security and privacy of users because this information can then be used to conduct targeted denial-of-service (DOS) attacks.

In this paper, we first go into more details about the technical underpinnings of the blockchain and the Lightning Network to better understand their potential weaknesses. Then we turn our attention to a survey of the vulnerabilities of the Lightning Network. After this, we introduce a case study and the approach behind behind probing channels. We study its implementation, while emphasizing the simplicity of the attack. Finally, we discuss the importance of our results and summarize the key findings of this research.

2 LIGHTNING NETWORK

2.1 The blockchain

When creating a digital financial medium, one of the biggest preoccupations is how to avoid double spending. In the physical world, this concern doesn't make much sense because objects can't be duplicated. In contrast, forgery of transaction is commonplace when computers are involved. This issue is illustrated by the classic Byzantine Generals' conundrum. Bitcoin solves double spending by synchronizing the whole network and enforcing a unique view of the distributed ledger. On the one hand, Bitcoin incentivizes miners to add blocks only to the longest and most stable chain of blocks. Going against this rule as a miner runs the risk of successfully mining a block but not receiving the reward because the shorter chain would get rejected by the wider network. On the other hand, users are encouraged to monitor the longest chain for a sufficient amount of time (6 blocks or 1 hour) before considering a transaction as

definitive. The state of the UTXO set (unspent entries in the ledger) older than 6 blocks is considered immutable in practice since block rearrangements of length 6 are negligeable events, probabilistically speaking. After an hour, the money is safely expended and can't be "double spent" anymore. For reference, there is a theoretical 51% attack on the blockchain that could subvert the entire network. The attack requires that a malicious entity own a majority of the computing power present on the network and manipulate the consensus mechanism.

The blockchain is, in essence, how Bitcoin provides a single source of trust to its users and can be leveraged by other applications like Lightning Network.

2.2 Transaction Rates and Scalability

As alluded to in the introduction, Bitcoin suffers from an inherent trade-off between security and swiftness of settlement. Bitcoin has stricken the balance of clearing transactions at in infrequent rate but does it in a very secure manner. Developers are therefore left with the task of improving the transaction rate and speed of the network by other means.

The main mechanism to evolve Bitcoin is known as BIP or bitcoin improvement proposal. BIP 141 is a famous proposal which enabled segregated witnesses (SegWits) and laid the technical foundation for the Lightning Network to grow [5]. Countless arguments have been waged across the community on how to improve the scalability of Bitcoin. Armed with the success of SegWit, proponents of on-chain scaling argue that these technical solutions can be seamlessly incorporated into the network. Critics counter that changes to the Bitcoin protocol puts unnecessary pressure on the end users. Since nodes have to carry a copy of the blockchain on their computer, an on-chain increase of the block size would blow the amount of space required out of proportion. What is more, on-chain forks (especially the hard ones) could have unintended repercussions and risk splintering the network. Another faction favors off-scale scaling for the reason that it leaves Bitcoin completely untouched and can be all the more flexible. LN falls into the latter category.

Judging solely on how LN solves the problem of speed and capacity, it is a brilliant success. The only transaction latency is that of carrying packets on the network and as such LN can theoretically execute millions of transactions per seconds, orders of magnitude more than Bitcoin. Let us get better acquainted with the Lightning Network by connecting to it and performing our first transaction.

2.3 Connecting to the Lightning Network

The Lightning Network (LN) as defined in the BOLT (basis of lightning technology) whitepapers has several software incarnations. As of January 2021, the 3 most prevalent implementations are c-lightning (C), LND (Go), and eclair (Scala). We have chosen the c-lighting [1] for the purpose of this paper. However, the implementations were created with interoperability in mind. Hence, results in this paper can easily be reproduced with other frameworks. The steps to establish a connection are summarized as follows:

(1) Compile/Install bitcoin-core and c-lightning
(2) Synchronize the blockchain (400GB)
(3) Connect to the lightning network
(4) Create a channel with one or more LN nodes
(5) Route transactions, collect fees, and make payments

In order to connect to the Lightning Network, a user must first host a synchronized full blockchain node. It can then connect to the lightning network and receive common information called gossip. Subsequently, the user creates his or her first channel with a lightning node. The sum committed to the channel logically places an upper bound on how much can be expended by the user. A by-product is that once a channel is depleted, the user can't collect routing fees anymore.

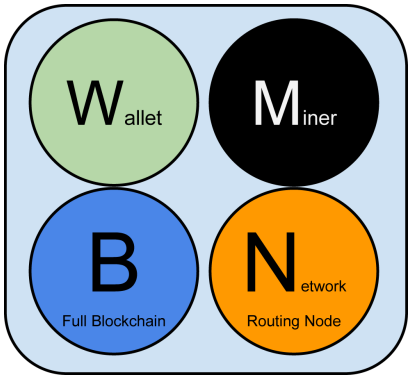

Figure 1: Functions of the Reference Client (Bitcoin Core) [7]

Figure 1 illustrates the 4 essential functions of Bitcoin Core. Network is necessary for all of the participants. From there, a user can choose to run an SPV node (simplified payment verification) with Wallet, a solo Mining rig with Miner, or a full node with Wallet and Full Blockchain. To connect and participate to the Lightning Network, the latter is required.

2.4 Channels and payment mechanisms

We now turn our attention to the trustless channels that make the Lightning Network (LN) possible. At heart, a channel is when 2 participants, Alice and Bob, put a predefined BTC amount on a shared address and broadcast it to the Bitcoin network. After waiting a few blocks, we now have the equivalent of a shared bank account and the blockchain has kept track of the initial distribution of funds. Alice or Bob can withdraw their money at any moment and close the channel in so doing. If they don't close the channel, the owners can start moving funds within the channel either through a simple transaction between them or as part of a more complex chain of transaction that happened to be routed at this location.

The biggest concern to anyone willing to receive BTC through a lightning channel is how to ensure that this newly advantageous balance is reflected on the blockchain and the other side can't just cash out with the previous state still on the blockchain. This scenario is the motivation behind state update enforcement. A channel state is nothing else than the ownership of funds at a given time. The HTLC (hash time-lock contract) solves all these problems at once and is the basis of state updates in channels.

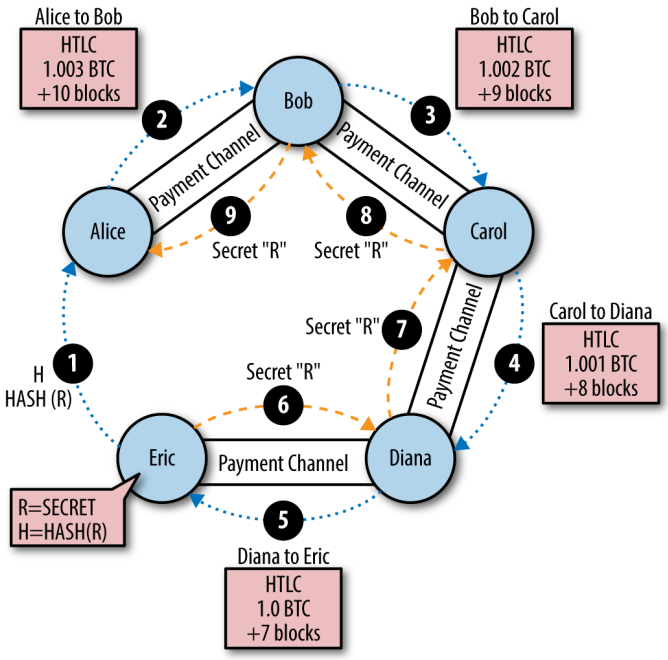

Figure 2: Routing mechanism

2.5 Routing transactions

The background key point is the multihop approach that allows two users without a direct channel to perform a payment. In the multihop approach, payments at each individual payment channel can't be performed exactly in the same way as with a single hop. An intermediate user has to enforce the exchange of the payment before the receiving node would accept the payment from the preceding node. That can only happen once it has performed the payment to the next one, otherwise the node would lose the amount of the payment. The enforcement of this type of atomic exchange between all the nodes of the path is performed using Hash Time-lock Contracts. In other words, all simple one-hop payments have to be completed or none can be processed. (HTLCs)[8].

2.6 Key features

- Privacy: Lightning Network payments are more private than Bitcoin transactions because the transaction is not broadcasted on a blockchain.
- Fungibility: Since Lightning adds a new layer of exchange on top of Bitcoin, the currency becomes all the more interchangeable.
- Speed: Lightning Network transactions are cleared in milliseconds in contrast to Bitcoin transaction that take 5 minutes on average.
- Granularity: Interestingly, Lightning allows transaction under the millisatoshi treshold which gives a new lease on life to bitcoin dust. This also side effects on the deflationary characteristic of Bitcoin.
- Capacity: Above and beyond speed, Lightning increases the capacity of the transaction and virtually removes any limit to transaction.
- Flexible Service : The blockchain technology provides flexible service through its smart contracts feature and this smart contract pool and the corresponding addresses of each smart contract creates an opportunity for service providers to program and operate the data in the blockchain and the system to deploy new services quickly and efficiently [9].
- Security : The blockchain technology provides strong security techniques, such as distributed ledger and validation, proof of work and encryption. This ensures the data provided by users cannot be tampered with, by other users in the networks. It achieves this through its encryption mechanism which also provides access control mechanism [9].

[7]

3 NEW SECURITY THREATS

Just as the technology matures and an increasing amount of money navigates the Lightning Network, so too grows the incentives for attackers to exploit its weaknesses. Nisslmueller et alii, Gudgeon et alii, Perez-Sola et alii have shown a host of methods relying on seemingly benign protocol routines to deanonymize targets, eclipse nodes, gain private information, and more. Some of these threats are not economically profitable while others are hard to execute if the target node is adequately prepared. However, all of the attacks have shown to work and can easily be reproduced.

3.1 Lockdown attack

The lockdown attack is the process of an attacker blocking middle nodes or victims in multipath payments. Upon a successful attack on the victims, an attacker may obtain a dominant position in the LN. By blocking some selected nodes the adversary becomes the main gateway to route payments allowing him to have an active position that can be exploited either for data and information-gathering purposes or just for increasing his collection of fees as an LN gateway node [13]. The multihop payment system holds that all intermediate payment nodes in the multihop route lock the balance of the payment, all of which until the route construction is completed so that all payments can be performed together. Malicious attackers take advantage of this mechanism within this period to lock a total amount of p balance in a channel AB. While the payment is being constructed, it then sends a payment of value p through the channel AB [13].

A demonstration of lockdown attack is shown where a victim A is a hub between two users, B and C, as depicted in Figure 1. Capacity values are Capacity $AB = p1 + p4$ and Capacity $AC = p2 + p3$ being p_i the balances in each direction for each channel. A simple scenario of the LN network is shown in figure 3. The objective of the adversary James (J) is to disrupt the availability of A by either blocking incoming links or outgoing ones. That is rendering $p1 = 0$ and $p3 = 0$ or $p2 = 0$ and $p4 = 0$ [13]. To perform this attack James (J) only has to open a channel with A as shown in figure 4 to block A from communicating with B and C.

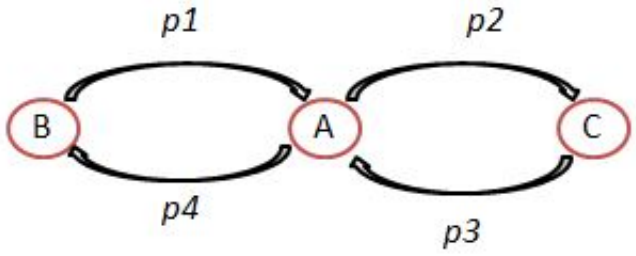

Figure 3: Simple scenario of LN network [13]

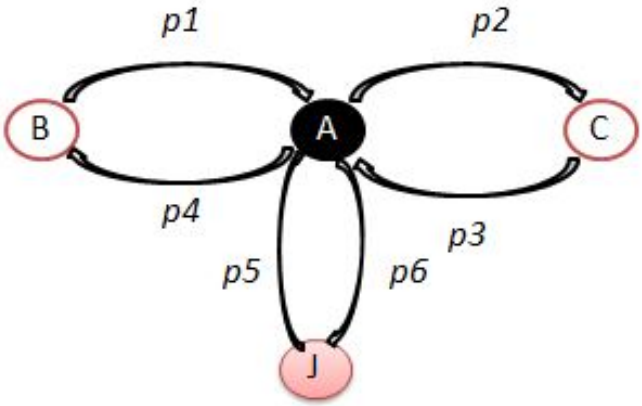

Figure 4: Simple scenario LN network with adversary J [13]

3.2 Eclipse attack

An Eclipse attack is where an attacker gains complete control over the content and timing of what a targeted node or victim sends and receives from the network [15]. This implies preventing a the node of a victim from communicating with other honest nodes in the network. It is usually done by occupying all of the victim's node connections by malicious nodes or pseudo-nodes [15]. This attack leads severe consequences for the victim up to real losses of coins. The procedure goes as follows.

After an attacker eclipses the node of a victim, the attacker performs time-dilation. This is the slowing down of block delivery to the victim's Bitcoin node [15]. Indeed, block announcement on the blockchain is a Poisson process of mean 10 minutes. The mining of the blocks regularly deviates from 10 minutes and trailing of hours happens frequently, a phenomenon colloquially known as bad luck. Yet time dilation is a very serious issue that is hard to detect [15]. Therefore, to strike a victim with time dilation, an attacker has to simply delay receiving a block and feeding it to the victim. Since the victim is eclipsed and does not have an honest source of blocks, the attacker can decide when the victim receives a new block [15]. If the attacker decides to litigate a channel on the blockchain after having accumulated a big time lead on its victim, the attacker may decide at any time to unleash a flurry of real blocks and render its victim helpless before a fait accompli: the coins will have already been stolen. The demonstration of an eclipse attack can be seen on figure 5.

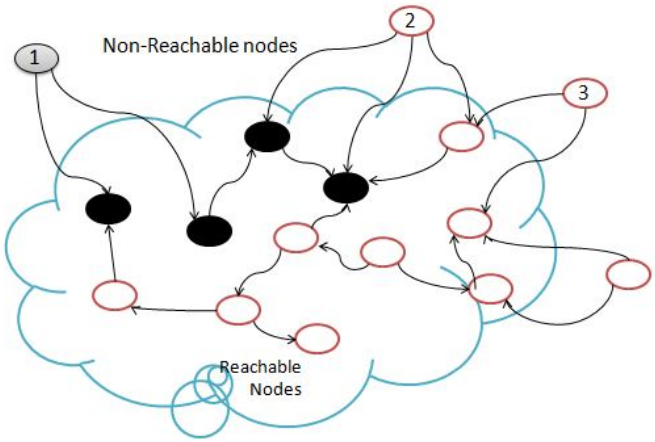

Figure 5: Eclipse attack on the Bitcoin network. Where only node 1 is eclipsed, because all its connections lead to the attacker [15]

3.3 Timing attack

The Lightning Network (LN) performs payments over multiple hops thanks to the use of HTLCs (hash time-lock contracts) [14] [11], a special bitcoin transaction whose unlocking conditions effectively rid the Lightning Network and its users of all trust requirements [11]. Cryptographically speaking, it is infeasible to try and determine where along the route a forwarding node is located thanks to the Onion Routing properties of the Lightning Network. Additionally, each node can only decrypt the layer which was intended for it to decrypt [11]. Attempts to analyze the remaining length of the routing packet have been thwarted at the protocol level by implementing a fixed packet size with zero padding at the final layer [11].

The main goal of Timing attack is finding out how far away nodes are from each other in the network. To achieve timing attack, one can start analyzing the encrypted traffic between the nodes to extract time-related information from the messages[11]. One possible way of doing this would be to analyze the *cltv-expiry-delta* field (analogous to passed", measured in mined blocks since the establishment of the HTLC). By looking at the delay of both the incoming and the outgoing HTLC, a node could infer how many hops are left until the payment destination [11]. However, this possibility has been accounted for by the adding of "shadow routes" to the actual payment path, with each node fuzzing path information by adding a random offset to the *cltv-expiry-delta* value, hence effectively preventing nodes from guessing their position along the payment route. [4]. Lightning Network messages are end-to-end encrypted, meaning that even if you know the target node IP address and port number, you cannot detect the exact nature of the messages exchanged [11]. Therefore, the only option left to carry this attack is to time messages at the network level, rather than at the protocol level, thus allowing nodes to listen to response

messages from other nodes, since there is currently no mechanism in place to add delay to update-fulfill-htlc responses [11].

4 PROBING ATTACK

4.1 Preliminaries

The Lightning Network uses several protocols to perform payments. It uses *channel announce* and *channel update* messages as critical protocols for triggering correct payment routing by other nodes on the network. The *channel announce* prompts the creation of a new channel between two LN nodes and is broadcasted exactly once.

The *channel update* is generated at least a single time by each endpoint, since even initially each of them may have a different fee schedule and thus, routing capacity may differ depending on the direction the payment [11]. There are two layer messages required to establish a payment chain;

- **update-add-htlc**: This message signals to the receiver, that the sender would like to establish a new HTLC (Hash Time-Locked Contract), containing a certain amount of msat (millisatoshis), over a given channel. The message also contains an onion-routed packet field, which contains information to be forwarded to the next hop along the route. The sender initially sets up an HTLC with Hop 1. The onion routing field contains another update-add-htlc (set up between Hop 1 and Hop 2), which in turn contains the ultimate update-add-htlc (set up between Hop 2 and Destination) in the onion-routing-field as shown in figure 6 in step 1, 2, 3 [11].
- **update-fulfill-htlc**: Once the payment message has reached the destination node, it needs to release the payment hash preimage in order to claim the funds which have been locked in the HTLCs along the route by the forwarded update-add-htlc messages. To achieve this, the preimage is passed along the route backwards, thereby resolving the HTLCs and committing the transfer of funds [11] as shown in step 4, 5, 6 on figure 6.

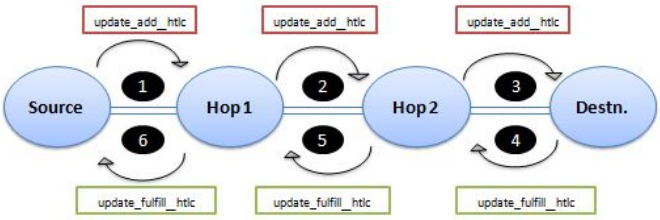

Figure 6: Transaction from source to destination involving 2 intermediate nodes [12]

4.2 Implementation

The Lightning Network handles payments through the invoice system. A LN invoice consists of a destination node ID, a label, a creation timestamp, an expiry timestamp, a CLTV (Check Lock Time Verify) expiry timestamp and a payment hash. An invoice can also be paid with a randomized payment hash. (since the routing nodes at this point do not know the actual hash) and will route the payment successfully to its' destination, which forms the basis for this attack. Alternatively, an invoice can contain an amount, a verbal description, a BTC fallback address in case the payment is unsuccessful, and a payment route suggestion. This invoice is then encoded, signed by the payee, and finally sent back to the payer [11]. When the payer receives a valid invoice, the payer can now either use the route suggestion within the invoice or query the network himself, and then send the payment to the payee along the route which has been determined. The payer can use *getroute()* function and *sendpay()* function, which takes two arguments: the return object from a *getroute()* call for a given route, a given amount and a given riskfactor, as well as the payment hash [11]. Using *sendpay()* can result in two error codes (This means using random payment hash instead of data from a corresponding invoice);

- **Error 204 (failure along the route)**. It is an error indicating failure of one of the hops to forward the payment to the next hop. This could be either due to insufficient funds or a non-existent connection between two adjacent hops along the specified route [11] as shown in figure 7.

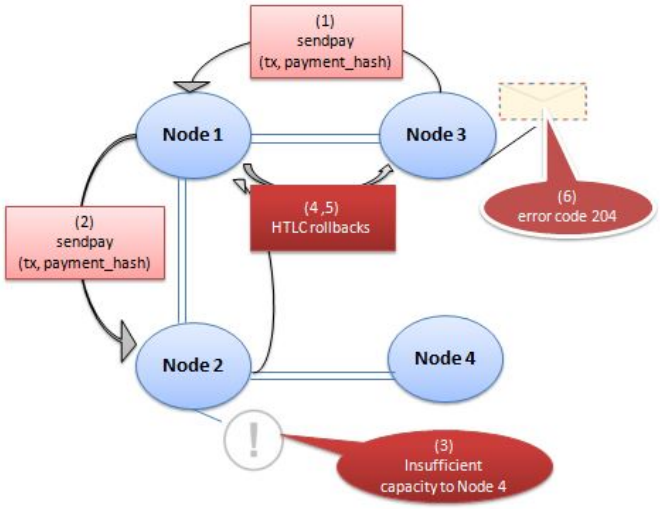

Figure 7: Creating 204 error by trying to send payment to Node 4, which Node 2 is unable to perform.[12]

- **Error 16399 (permanent failure at destination)**. It is assumed at this point that the attempted payment has reached the last hop if no error 204 is received. Using a random payment hash, certainly the destination node will send an error, signalling that no matching preimage has been found to produce the payment hash [11] as shown in figure 8.

4.3 Formalization

We can discriminate 16399 failed transactions that have reached their targets but returned empty-handed because the invoice was scrambled from the get-go, from 204 failed transactions that couldn't cross a channel along the way for some as-of-yet unknown reason. In reality, failcodes 204 are a superset of the -1 and 4103 failcodes. All of which occur when one of the channels along the route had insufficient funds to carry out the transfer.

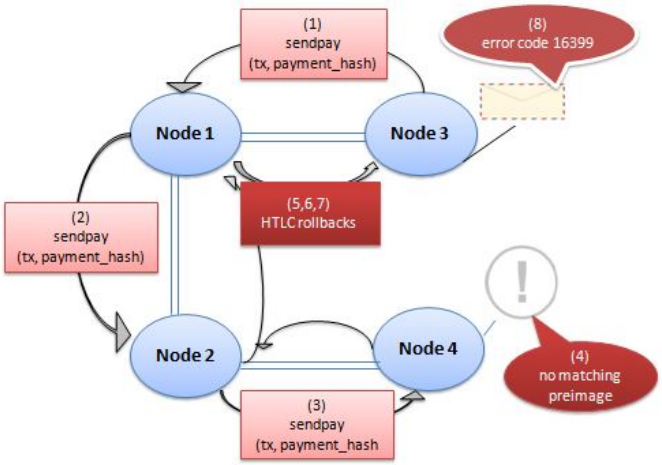

Figure 8: Causing 16399 error by trying to send payment to Node 4, who cannot produce a matching preimage and thus fails the payment [12]

Let R be the route composed of nodes $\{N_i\}$ such that $R = \{N_i\}_{i \in [[0;M]]}$ where M is the length of the path and $1 <= M <= 20$. N_0 corresponds to the attacker's node and N_M to the target node.

Let $R(x)$ be the return code received after sending the amount x over route R.

Finally, let C_i be the channel between N_i and N_{i+1} whose capacity $cap(C_i)$ is publicly known. We would like to probe the distribution of funds $dist(C_i) = \{s_i; r_i\}$ where s_i is the balance on the sender's side, r_i the balance on the receiver's side (from the attacker's perspective) and $s_i + r_i = cap(C_i)$.

If we receive -1 along route R, we have in effect discovered that one of the channels had in imbalance in its distribution of funds and couldn't forward the transaction. This principle is summarized by the following relationships:

$$R(x) = -1 \implies min(s_i) < x : i \in [[0; M-1]]$$

$$R(x) = 16399 \implies min(s_i) > x : i \in [[0; M-1]]$$

In practice, we created a test channel between a target node (N_T) and a well-connected node (N_I). We simplified the deduction process and fixed the distribution of funds for the duration of the attack by deactivating fee collection. Next, we simulated an attacker (N_A) wanting to probe this channel. To convert the bound on the funds of the whole path to a bound on the probed channel, we created routes of length 2: $R = \{N_A, N_I, N_T\}$ by connecting the attacker directly to N_I. This leaves us with the following:

$$R(x) = -1 \implies s_2 < x$$

$$R(x) = 16399 \implies s_2 > x$$

If s_2 is the unknown, the simplest way to get to its value is to perform a binary search on the x space.

4.4 Reproducibility

We have chosen to automate the probing procedure by coding a Lightning Network plugin in Python. Plugins are launched at the same time as the lightning daemon and communicate with it by using RPC. As can be seen in the annex, the script has access to the same commands as the usual lightning CLI. The Python library that acts as the interface is called pyln-client (0.9.2). Example plugins created by the community can be found on the c-lightning github page [2].

4.5 Experimental results

After creating a target channel and setting up a 2-hop route between the attacker and the target, we get the following results:

Epsilon (msat):	10	100	1000	10000
Nb. of iterations	27	23	20	17
Convergence (msat)	153,890,003	153,889,988	153,889,845	153,890,990
Probe #1 (s)	42.991	44.291	32.674	29.092
Probe #2 (s)	43.243	39.725	31.589	31.745
Probe #3 (s)	47.818	41.849	32.375	31.538
Probe #4 (s)	46.289	37.218	36.379	33.076
Probe #5 (s)	44.44	34.29	37.111	31.742
Mean probe duration (s)	1.67	1.72	1.70	1.85

The algorithm converges around 154,000 sat, slightly undershooting the actual s_2 balance. Knowing that the total capacity $C_1 = 0.045BTC$, we get $s_2 = 154,000sat$ and $r_2 = cap(C_1) - s_2 = 4,350,000sat$. We have thus successfully estimated the distribution of one channel. This process can be replicated to all the channels established by a particular node to find the total lightning wealth of its owner.

We can empirically observe that the duration of convergence scales linearly with the number of probes. Whereas the probes generated by the binary search algorithm scale algorithmically as $O(log(n))$, n being the capacity of a channel. This is corroborated by the mean probe duration roughly constant regardless of the value of ϵ. To reduce the number of probes required before convergence, it is sufficient to gradually decrease the precision $\alpha = \frac{1}{\epsilon}$. Note that increasing ϵ past a certain point is bootless as it gives exponentially diminishing time savings because of how the binary algorithm works.

As a rule of thumb, results will significantly depend on the state of the network, and the latency. An epsilon of 1000 msat will work quite well in practice.

4.6 Interpretation

The previous results show that an attacker with sufficient information on a given node can easily conduct a probing attack and discover how funds in a given channel are distributed. The attacker can then scale up this attack for a virtual cost of zero to cover all the channels stemming from the target. In a few minutes, an attacker can then generate a profile of the total amount of funds owned by a target. This technique suffers from mild imprecision because of the dynamic nature of the lightning network. But all in all, the

lightning protocol discloses a worrying amount of information to any user savvy enough to prompt it to.

4.7 Further works

Left unanswered by our current research are the possibilities of dynamically changing channels and creating routes longer than 3 nodes. The traceroute plugin [3] can be used to iterate over prefixes of any given route and thus isolate the min s_i from our previous equation. This addresses the question of navigating more complex routes. Although in practice there are no limitations to what nodes you can create a channel with and a target can always be reached with routes of size 2.

Channels are not static entities. The capacity is fixed but the funds slosh around on a frequent basis when rerouting lightning transactions and collecting fees. This can pose a challenge to attackers wanting to ascertain the distribution with a high accuracy. As it stands now, the attacker can give up a certain amount of accuracy to still have broadly application results.

5 CONCLUSION

In many regards, the Lightning Network has kept its promise of delivering huge speed and capacity improvements, by supplementing the Bitcoin network. This off-chain solution has gathered a lot of popularity in recent years but suffers from undeniable weaknesses. As we have seen in this paper, creative approaches like time dilation and lockdown have proven effective means to tamper with the network's security. Not only do these vulnerabilities threaten everyday users but they also have the potential to stunt the development and adoption of Lightning Network at large if not dealt with. Furthermore, a concrete implementation of a probing discovery attack has shown that – short of stealing the funds in a channel – an attacker can reveal the distribution of funds and take advantage of it.

For all its vulnerabilities, the Lightning Network is still in its infancy and in continuous development. Luckily many of the vulnerabilities do have technical solutions, which is a subject that would warrant another paper by itself.

6 ANNEX

Original repo: https://github.com/Yann21/lightning-probe

```
# Probing distribution of channels on the Lightning Network
# using a binary search.

# If the error code is 16399, then the channel has a
     sufficient entry balance to accommodate the
     transaction; we can probe higher.
# If the error code is -1 or 4103, then the transaction
     amount exceeds the entry balance; we must probe lower.
from tail_recursion import tail_recursive
from pyln.client import Plugin, RpcError

plugin = Plugin()

## Binary search part of the probing.
# Epsilon-precision in satoshis
epsilon = 10

@plugin.method("probe")
def bin_search(plugin, node_id, capacity, **kwargs):
    @tail_recursive
    def loop(lower, upper):
        middle = (lower + upper) // 2
        error_code = failcode(node_id, middle)

        if (upper - lower) < epsilon:
            return middle

        elif error_code == 16399:
            return loop(middle, upper)

        elif error_code == 4103 or error_code == -1:
            return loop(lower, middle)

        else:
            raise Exception("UnknownErrorCode")

    return loop(0, capacity)

## Lightning Network RPC calls
def failcode(node_id, amount, **kwargs):
    probe = {
        'destination': node_id,
        'started_at': str(datetime.now()),
        'probes': [],
        'payment_hash':
             ''.join(random.choice(string.hexdigits) for _
              in range(64)),
    }
    try:
        probe['route'] = plugin.rpc.getroute(
            probe['destination'],
            msatoshi=amount,
            riskfactor=1,
        )['route']
        probe['payment_hash'] =
             ''.join(random.choice(string.hexdigits) for _
              in range(64))
    except RpcError:
        probe['failcode'] = -1
        return probe

    plugin.rpc.sendpay(probe['route'],
         probe['payment_hash'])

    try:
        plugin.rpc.waitsendpay(probe['payment_hash'],
             timeout=30)
        raise ValueError("The recipient guessed the preimage
             and cryptography is broken.")
    except RpcError as e:
        probe['finished_at'] = str(datetime.now())
        if e.error['code'] == 200:
            probe['error'] = "Timeout"
        else:
            probe['error'] = e.error['data']
            probe['failcode'] = e.error['data']['failcode']

    return probe["failcode"]
```

REFERENCES

[1] [n.d.]. https://github.com/ElementsProject/lightning.
[2] [n.d.]. https://github.com/lightningd/plugins
[3] [n.d.]. https://github.com/lightningd/plugins/tree/master/probe
[4] [n.d.]. BOLT 7: P2P Node and Channel Discovery. https://github.com/lightningnetwork/lightning-rfc/blob/master/07-routing-gossip.md
[5] 2015. BIP 141. https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki.
[6] 2019. SWIFT in figures. https://www.swift.com/sites/default/files/documents/sif_201912.pdf.
[7] A.M. Antonopoulos. 2014. *Mastering Bitcoin: Unlocking Digital Cryptocurrencies*. O'Reilly Media. https://books.google.lu/books?id=IXmrBQAAQBAJ
[8] Christian Decker and Roger Wattenhofer. 2015. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems*. Springer, 3–18.
[9] Yuhong Li, Kun Ouyang, Nanxuan Li, Rahim Rahmani, Haojun Yang, and Yiwei Pei. 2020. A blockchain-assisted intelligent transportation system promoting data services with privacy protection. *Sensors* 20, 9 (2020), 2483.
[10] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System.
[11] Utz Nisslmueller, Klaus-Tycho Foerster, Stefan Schmid, and Christian Decker. 2020. Toward Active and Passive Confidentiality Attacks On Cryptocurrency Off-Chain Networks. arXiv:2003.00003 [cs.CR]
[12] Utz Nisslmueller, Klaus-Tycho Foerster, Stefan Schmid, and Christian Decker. 2020. Toward Active and Passive Confidentiality Attacks On Cryptocurrency Off-Chain Networks. *arXiv preprint arXiv:2003.00003* (2020).
[13] Cristina Pérez-Solà, Alejandro Ranchal-Pedrosa, Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, and Joaquin Garcia-Alfaro. 2020. Lockdown: Balance availability attack against lightning network channels. In *International Conference on Financial Cryptography and Data Security*. Springer, 245–263.
[14] J Poon and T Dryja. [n.d.]. The Bitcoin Lightning Network: Scalable O-Chain Instant Payments, 2016. *Online manuscript* ([n. d.]).
[15] Antoine Riard and Gleb Naumenko. 2020. Time-Dilation Attacks on the Lightning Network. *arXiv preprint arXiv:2006.01418* (2020).