– Independent Work Report Spring 2019 –

Chainwatch: Detecting 51% Attacks on Proof-of-Work Blockchains

Michael Man
Adviser: Arvind Narayanan

Abstract

While Proof-of-Work consensus protocols enable blockchains to be trustless, transparent, and decentralized, any miner that gains a majority share of the network's total computational power can launch a 51% attack to double spend cryptocurrency. Despite the exponential growth of blockchain and cryptocurrency technology, 51% attacks are a million-dollar problem that still remains unsolved. This paper examines how a combination of mechanisms, protocols, and incentives behind distributed ledger technology has enabled double-spending attacks to occur; furthermore, it presents multiple strategies for the real-time detection of such attacks. Through detecting chain reorganizations, tracking miner hash share, and analyzing network statistics, Chainwatch *implements these strategies to monitor blockchains for potential attacks. In addition to detailing the design and implementation decisions behind* Chainwatch*, this paper also presents a demonstration of its detection capabilities in the context of a controlled 51% attack on the Ethereum Ropsten test network. This application aims to address the current needs of real-time 51% attack detection by providing a detailed, transparent, and accessible overview of any vulnerable blockchain network.*

1. Introduction

In 2008, an anonymous author under the pseudonym of Satoshi Nakamoto published "Bitcoin: A Peer-to-Peer Electronic Cash System", laying down the foundations for blockchain technology [14]. Since then, blockchain technology have grown exponentially, with cryptocurrencies occupying a market capitalization of around $200 billion today. The core principle behind Bitcoin and similar blockchains that have followed lies in their decentralization — the concept that no corporation, organization, or user has the ability to control the workings of a distributed ledger. However, as the blockchain ecosystem has grown, dangers to this core tenet have also begun to manifest themselves.

To enable this decentralized quality, most blockchains rely on a distributed consensus protocol known as *Proof-of-Work*. Validating nodes, often referred to as *miners*, process user transactions and append transaction and state data to the distributed ledger. To maintain decentralization between the miners in a network, proof-of-work relies on miners performing computationally-expensive operations to delegate the append rights to the blockchain. This has given rise to the possibility of a *51% attack*, in which a malicious miner (or group of miners) that possess a majority share of computational power can append and revert blockchain data to defraud vendors and cryptocurrency exchanges.

This paper will examine the inner workings, enabling factors, and detection strategies behind 51% attacks and analyze the impacts these attacks have on the growing field of blockchain development as a whole. Furthermore, I will outline the design, implementation, and experimental results of my utility program *Chainwatch*, a blockchain statistics monitoring platform that I developed to detect such attacks.

2. Problem Background

To understand the workings of 51% attacks, we must first understand how Proof-of-Work blockchains work. This section will cover their design, implementation, and operation, as well as establish some key terms that will be critical for the rest of the paper. This overview will remain blockchain-agnostic, as while various blockchains (e.g. Bitcoin, Ethereum, Litecoin) all differ in terms of implementation, they still share certain core design principles. The following concepts apply to the vast majority of existing public blockchain networks in operation today.

2.1. Blockchain Basics

A blockchain, a distributed ledger of transactional and state data, can be considered analogous to a linked list of *block* data structures. Each block contains a *block header*, as well as a list of transactions.

The block header acts as a lightweight representation of the block. It contains a reference to the previous block in the form of a hash[1] of its block header, a timestamp dating when the block was created, as well as the root hash of a Merkle tree of the transactions included in this block [14]. A *nonce* value is also included in each header, as well as the unique hash identifier of the miner that created this block (covered in section 2.2).

The transaction data associated with each block tracks transfers of cryptocurrency between users on the network. Transactions of cryptocurrency are cryptographically signed by the sender's private key,[2] then this transaction data is broadcast to the network so that miners can include it in an upcoming block. Various blockchain implementations handle transaction verification differently, with Bitcoin relying on UTXO (unspent output, senders prove they can pay an output transaction by referencing an input transaction), and Ethereum maintaining a state database containing the cryptocurrency balance of each user. This chain of blocks is publicly known to all participants on the network, so it act as a complete record of every transaction that has ever occurred on this network.

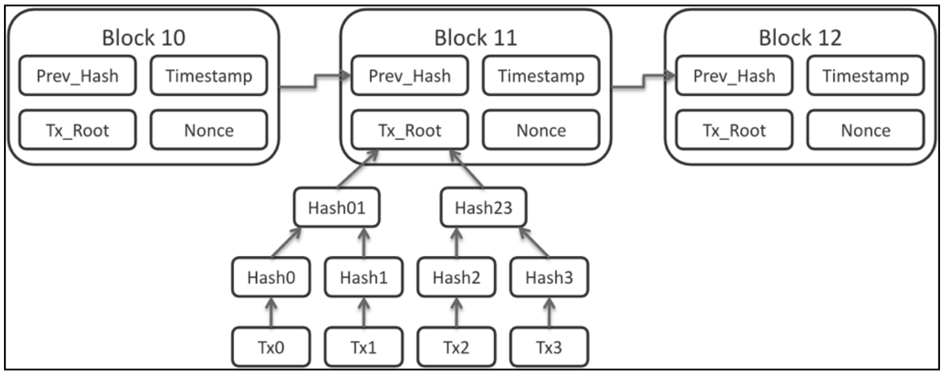


Figure 1: A diagram of three blocks (a subset of the longer blockchain).

2.2. Proof-of-Work & Mining

In order to append new blocks onto the blockchain, *miners* must first determine a list of transactions to include in a block, then recursively generate nonce values until the hash of the contents of block header is lower than a *target difficulty*. The difficulty value is set such that each nonce only has a

[1]Various blockchains use different hash functions, with Bitcoin using SHA-256 and Ethereum using Keccak-256.
[2]Blockchain technology relies heavily on public-key cryptography, which is beyond the scope of this paper

very small probability of resulting in a hash that satisfies the network requirement to add a block, and the difficulty value represents how many hashes must be attempted on average before a valid nonce can be discovered. This forms the basis for *Proof-of-Work* (PoW), where miners hoping to append a block must attempt a very large number of nonces until they generate a valid block [3].

Upon generating such a nonce, the miner will broadcast the block to the network. For other nodes on the network, verifying that a proposed block is simple and can be done in a single hash. If valid, the miner will receive a *block reward*, a predetermined amount of cryptocurrency that acts as an incentive for mining blocks. At this point, the block is officially included onto the blockchain, and miners will begin to mine on top of the newly discovered block.

As such, appending blocks onto the blockchain can be extremely computationally intensive.[3] This decentralizes the access right to append data based on the computational power of each miner, commonly referred to as each miner's *hashrate*. For a miner M that is capable of computing h_M hashes per second, they are competing with the other miners on the network to generate the next valid nonce. If the entire network had a combined hashrate of h_N, then for each block M has probability $\frac{h_M}{h_N}$ proportional to their hashrate to generate a valid nonce before the rest of the network.

For individuals miners (i.e. $h_M << h_N$), their expected probability of generating a valid block is close to zero. As a result, these smaller miners tend to join *mining pools*. In a mining pool, each miner that contributes hashes to the pool is paid proportional to h_M, such that even if h_M is a tiny fraction of the total hashrate of the network, the miner will still be rewarded. In this paper, the term *miners* will represent both large individual miners and these mining pools.

2.3. Forks & Longest Chain Rule

From the proof-of-work protocol described above, a miner can generate a block referencing any previous block header. However, nodes on the network will only recognize the longest blockchain to be valid. This is derived from the PoW consensus protocol, such that only the blockchain with

[3]For example, the Bitcoin network is currently capable of 50 exahashes per second. Nevertheless, a valid block is only discovered on average every 10 minutes.

the most computational work done on it (i.e. the most blocks) will be considered the *canonical chain*. Only transactions on the canonical chain are considered to be valid.

If the current number of blocks, the *block height*, of the blockchain is B, miners will generally attempt to mine the $(B+1)^{\text{th}}$ block. However, due to various reasons[4], a miner M could in theory mine a block at any position, including at position B (referencing in its header the $(B-1)^{\text{th}}$ block). The figure below shows an example of such a block being created.

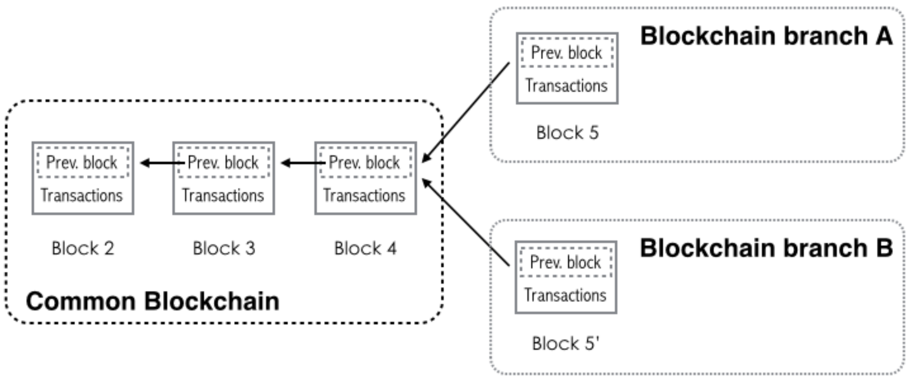


Figure 2: A miner can fork the blockchain, producing two blockchains of height $B = 5$.

This is an example of a *fork* in the blockchain, in which two valid blockchains have been created. Blocks 5 and $5'$ are both valid blocks, both sharing the same common history in blocks position < 5. However, blocks in branch A in Figure 2 will not be aware of the transactions in branch B (including the granting of miner block rewards), and vice versa. Nodes on the network that first hear about branch A will accept it as the canonical chain and reject block $5'$ (and vice versa) [17].

2.4. Chain Reorganizations

To resolve such a fork, the longest chain rule will apply as soon as one forked chain outpaces the other. If a miner discovers a block at height 6 on branch A, nodes that originally believed branch B to be the canonical chain will discard it in favor of branch A (the same applies for a block at height $6'$). The blocks on the shorter branch B that do not match those of A will be *orphaned* and

[4]This may be done maliciously or accidentally in the case where M has no knowledge of the B^{th} block (often as a result of block propagation delays)

discarded, and any transaction data stored within those blocks will be invalidated. This is termed a *chain reorganization*, in which a node will effectively "back up" its chain by discarding orphaned blocks and replacing them with a longer chain.

The longest chain rule incentivizes miners to mine on the longest chain they know about, as an orphaned or rejected block will not yield the miner any block rewards for their work. However, due to delays in block propagation, accidental forks and block orphaning do occur on a regular basis for most blockchains. In fact, chain reorganizations are a part of a protocol-level decision design that enables blockchains to survive network partitions and remain both consistent and fault-tolerant. Similar features can be found in other examples of distributed replicated state machines such as Paxos and Raft, in which log entries can be deleted and overwritten as a recovery mechanism from network partitions [16].

To mitigate transactions being lost due to accidental or malicious chain reorganizations, most vendors (e.g. cryptocurrency exchanges) that accept cryptocurrency impose what is known as *confirmation* requirements. In essence, they require that transactions be at least Ω blocks "old" before they are considered immutable. For example, Bitcoin confirmation requirements are generally set at $\Omega = 6$, so for a longest chain of block height B, only transactions in blocks $B - 6$ and before are safe to be considered immutable from a chain reorganization. [5] This is because the probability of a reorganization occurring at a certain block position decreases exponentially as more blocks are appended after it [13].

2.5. 51% Attacks

Finally, this brings us to the central topic of this paper. For each valid block that a miner produces, they have the option of broadcasting it to the rest of the network or hiding it. All honest miners are incentivized to broadcast a mined block as soon as a valid nonce is discovered, or else they risk another miner publishing a block at the same height and receiving the block reward (hence invalidating and wasting the mined block). However, since a miner (or mining pool) M has a $\frac{h_M}{h_N}$

[5]For Bitcoin results in an estimated 60 minute wait for a transaction to be confirmed, which severely impacts the user experience of Bitcoin transfers.

probability of mining a new block, if $h_M > \frac{1}{2}h_N$, then M will generate blocks at a faster rate than the rest of the network. Therefore, if M were to hide their blocks, they will generate a hidden chain that is guaranteed to outpace in size the honest miners' chain. Upon revealing that hidden chain, M can guarantee that other miners will accept their longer chain, forcing a chain reorganization.

To exploit this attack for gain, any miner M with $h_M > \frac{1}{2}h_N$, can commit what is known as a *double spend*. This involves M making a transaction to a vendor, then revealing a private chain that causes that transaction to be discarded. When targeting a cryptocurrency exchange, this attack enables M to effectively spend their cryptocurrency twice and defraud the exchange.

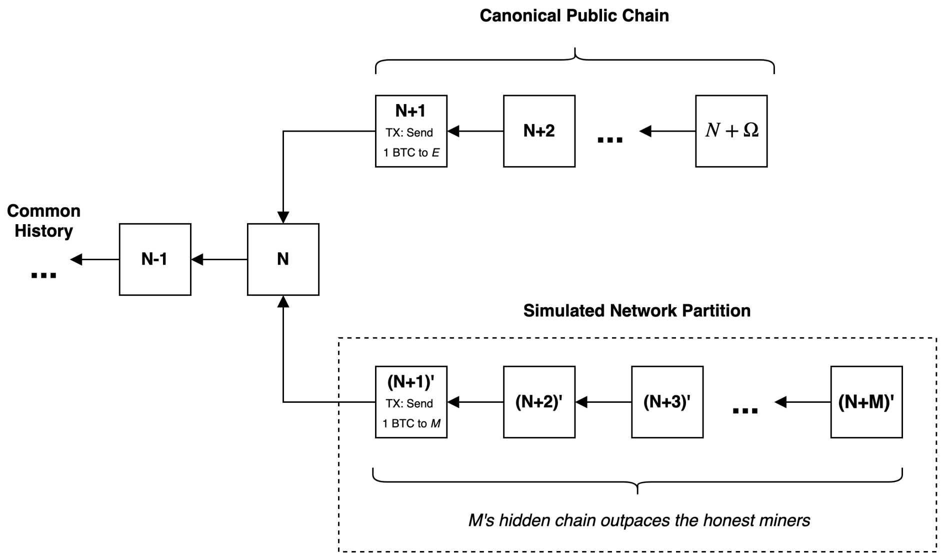


Figure 3: A attacker M can outpace the honest miners.

In Figure 3, M has created a transaction of 1 Bitcoin to E, the cryptocurrency exchange, in public block $N+1$. In secret, M begins mining a hidden chain, including a transaction to themselves in block $(N+1)'$. Since M has a majority hashrate, it is guaranteed to eventually outpace the canonical public chain. After the transaction has been confirmed (in block $N+\Omega$), M will have a private chain of length $N+I$, where $I > \Omega$. From the cryptocurrency exchange E, the attacker can withdraw their funds (either in a different cryptocurrency or in fiat currency). After cashing out, M can now broadcast their hidden blocks to the network to effectively reverse this transaction.

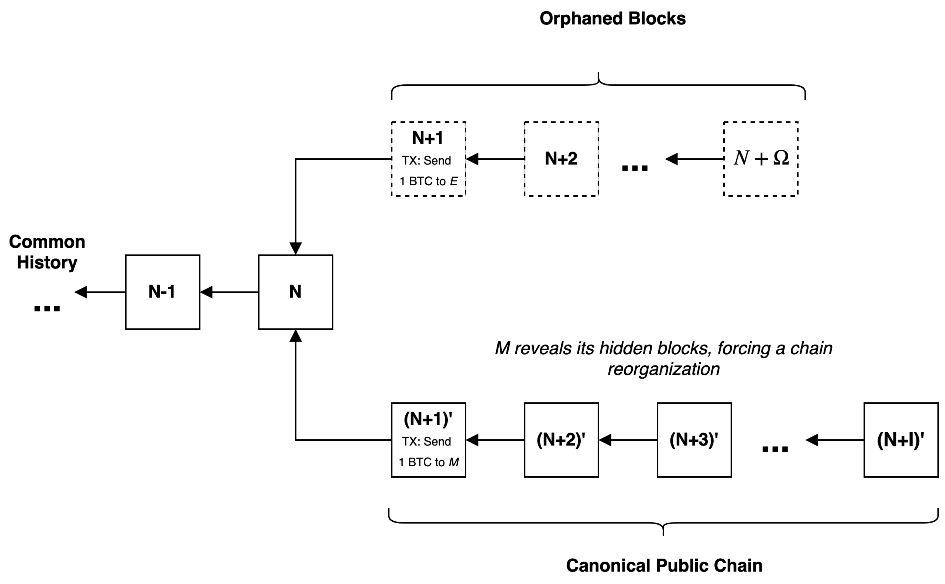


Figure 4: M reveals its hidden chain to commit a double spend.

After M announces blocks $(N+1)'$ through $(N+I)'$, the rest of the network will recognize M's chain as the canonical chain. This causes blocks $N+1$ through $N+\Omega$, originally mined by the honest miners, to be orphaned and discarded. The 1 Bitcoin is retained by M in the new public chain, and M will have successfully committed a double spend.

Note that while M retains their Bitcoin while also being credited for its deposit on E, no new Bitcoin was actually generated. The attacker was instead credited for cryptocurrency that, in the end, was never actually deposited to the exchange.

For honest miners on the network, there is little they can do to stop the double spend. After a chain reorganization is detected, miners are incentivized to mine on the new longest chain published by the attacker. While miners can choose to intentionally stay on their original (and now shorter) chain in a futile attempt of resisting the attacker, any block rewards they net via mining will not be recognized on the new canonical chain. Therefore it would be in each miner's best interest to accept the attacker's chain and begin mining on top of M's blocks.

2.6. 51% Attacks in Practice

While it seems unlikely that an attacker M can possess majority hashrate share for a given blockchain network, 51% attacks are not only feasible in practice, but occur with shocking regularity. For the largest networks like Bitcoin and Ethereum, it is true that gaining a majority hashrate is both economically and practically infeasible (but not impossible), smaller blockchains often do not share the security of the larger chains. Blockchains tend to have total network hashrates proportional to the market capitalization of the underlying cryptocurrencies, so smaller chains will have significantly fewer miners securing the network.

For example, an industry standard Antminer S9 ASIC (application-specific integrated circuit) has a hashrate of around 14 TH/s (for Bitcoin's SHA-256 hash algorithm)[4]. For the Bitcoin network, with total hashrate hovering around 50 exahashes per second[5], a single S9 miner will contribute around 0.00003% of the total hashrate. For a smaller network that's also built around the SHA-256 hash algorithm such as Zetacoin (ZET), which has a total hashrate of only 42 TH/s, the same S9 miner will be capable of 33% of the total. As a result, it is both cheap and incredibly simple to move hashing power from the Bitcoin network to launch a 51% attack and double spend.

NiceHash, an online hashing power marketplace founded in 2014, allows miners to sell access to their mining equipment to client pools for any hash algorithm of the customer's choice. Today, so much hash power is available for purchase on NiceHash such that dozens of cryptocurrencies are vulnerable to attack. The table below lists cryptocurrencies that have a total hashrate that is below the available NiceHash hash power for the associated hash algorithm as well as the cost of gaining a majority hashrate share for an hour [7].

Cryptocurrency	**Market Cap**	**Hash Algorithm**	**Total Hashrate**	**51% Hourly Cost**
Bitcoin Gold (BTG)	$327 million	ZHash	3 MH/s	$1,324
Bytecoin (BCN)	$177 million	CryptoNight	550 MH/s	$146
Verge (XVG)	$112 million	Lyra2REv2	4 TH/s	$143
Metaverse (ETP)	$40 million	Ethash	488 TH/s	$304
ZClassic (ZCL)	$8.8 million	Equihash	106 MH/s	$391
Zetacoin (ZET)	$375,000	SHA-256	42 TH/s	$0.44

Table 1: NiceHash costs for gaining a majority hashrate share for various smaller blockchains.

As evident above, the cost of using NiceHash to temporarily gain a majority hashrate share in a smaller blockchain network is an insignificant fraction of the total worth of the cryptocurrency.[6] Furthermore, renting hash power enables attackers to ignore the fixed costs of purchasing expensive mining equipment and the technical complexities involved in their setup. With NiceHash, an attacker simply can direct hash power to their own malicious mining pool. From there, they can create a temporary network partition (as seen in Figure 3) and begin mining a hidden chain.

As a result, 51% attacks have evolved from being a theoretical concern to becoming a practical reality of the blockchain space. In May 2018, attackers launched a 51% attack on Bitcoin Gold (BTG), defrauding exchanges of an estimated $18 million [10]. In January 2019, attackers used a similar attack on Ethereum Classic (ETC), double spending around $1.1 million [15]. Other cryptocurrencies that have been attacked include Verge (XVG), ZenCash (ZEN), Vertcoin (VTC), and many more. The status quo has made it very clear that the timely detection of such attacks, as well as the development of protocol-level changes to blockchain clients, has become imperative to protecting the health and future prospects of the blockchain ecosystem.

2.7. Current Approaches to Detection and Mitigation

As of today, the state of blockchain monitoring to detect malicious chain reorganizations is woefully lacking. While some exchanges (who are almost always the target of such attacks) run proprietary code on their servers to detect instances of fraud and double spends, this data is not accessible nor transparent to the public. This leads to a trust assumption being made, where ordinary users must trust centralized services to monitor the security of the public networks. For example, the public was first informed of the case of the Ethereum Classic 51% attack via a tweet from Coinbase, one of the largest cryptocurrency exchanges [6]. Upon discovering an attempt at a double spend, exchanges often freeze the affected cryptocurrency from all trading activity, effectively preventing the attackers from "cashing out." Considering the exchanges' interest in protecting themselves from fraud, it is understandable why the burden of detecting such attacks has fallen upon platforms. However, in

[6]Note that higher hashrates do not necessarily correspond to higher attack costs. This is because different hash algorithms have varying computational and financial costs.

the spirit of decentralization, it is critical that the public, who are just as easily victimized by such attacks, has the tools to detect 51% attacks without relying on third-party exchanges.

Block explorers are public websites that allow visitors to explore many facets of blockchain data, ranging from tracking transactions to reading block header data. Block explorers provide helpful information for the detection of chain reorganizations and double spends, such as tracking network hashrate and orphaned blocks. However, these utilities tend to be delayed in terms of their detection as they fail to provide real-time data, a property that is critical for timely detection of attacks. Furthermore, block explorers such as *Etherscan*, the most popular and well detailed platform for Ethereum, only serve to provide general overview of network statistics. Considering the latency and lack of resolution of this data, it is not very useful for attack detection purposes.

In short, the current state of blockchain monitoring is lacking in terms of detail, transparency and timeliness. Moreover, it is impractical and irresponsible to delegate the task of blockchain security monitoring to self-interested centralized exchanges like Coinbase. To counter this problem, I have designed *Chainwatch* with these limitations in mind, seeking to provide a real-time blockchain data analytics platform to act as a first line of defense against 51% attacks.

3. *Chainwatch*'s Approach

3.1. 51% Attack Detection Approach

In light of the current lacking state of most blockchain monitoring services, I have created a utility software, *Chainwatch*, that addresses the security needs of the blockchain space. *Chainwatch* is designed with the goal of timely 51% attack detection; to do so, I have decided to tackle the problem from three main directions. Each approach has its own limitations, so I have determined that these three strategies should be used in conjunction with each other.

3.1.1. Detecting Chain Reorganizations

The most straightforward form of detecting a 51% attack and double spend is to detect chain reorganizations when they occur. This can be done by tracking the current longest known chain, then detecting the case where a conflicting fork is broadcast to the network. While chain reorganizations do occur as a result of partitions and block propagation delays, any 51% attack and double spend requires a significant (length > Ω (confirmation requirement)) chain reorganization to commit a double spend. Therefore, a deep chain reorganization can signal an attempted attack. However, this approach detects attacks after they have occurred (i.e. after an attacker has revealed their hidden blocks[7]), so it fails to detect attacks while they are still in-progress.

3.1.2. Tracking Miner Hash Share

Hash share, the percentage of total hash power that a given miner possesses, can be used as a metric to identify attacks. Unlike detecting chain reorganizations, tracking miner hash share can be employed to detect 51% attacks as they occur. This works by tracking the distribution of hash power to the largest miners and mining pools on a network. If a miner were to gain near or above 50% of the total hash power, then that indicates that the network is at imminent risk of an attack.

In the case that a miner's hash share were to suddenly and dramatically increase, then that signals large quantities of hash power being moved onto the network (including from rental services such as NiceHash). In the opposite case, if a miner's hash share were to suddenly decrease, it could signal that miner is now hiding their blocks in preparation for triggering a malicious chain reorganization. This approach, however, is unable to conclusively detect attacks as they occur — significant changes in miner hash share can represent both benign and malicious intentions on the part of the miner. For example, a sudden drop in hash share could be a result of a software crash or an accidental network partition.

3.1.3. Tracking Changes in Network Hashrate

For the most part, the total hashrate of a network is directly correlated to the profitability of mining the underlying cryptocurrency. Profitability is a product of a multitude of factors, including the price of the cryptocurrency, variable costs of mining (i.e. electricity), as well as the network's current mining difficulty. In the case that there are sudden changes in network hashrate, however, then that could potentially signal either an influx of hash power or a sudden network partition. For example, an attacker that rents hash power from NiceHash for the malicious goal of launching

[7] See Figure 4

a 51% attack will at minimum double the total hash rate of the network. On the other hand, an attacker that begins to hide their blocks from the rest of the network will cause total observed hash rate to drop dramatically, as the hash power of the attacker will no longer be contributing to the public chain. This is similar to the strategy of tracking miner hash share in terms of effectiveness, but it has the added benefit of identifying cases where a multitude of miners are colluding to trigger a reorganization.

It has become common practice for miners and pools to change the blockchain network they mine on based on their profitability calculations. In that case, this approach may flag a sudden increase or drop in hash rate as suspicious, even though no attack is taking place. As with the other two approaches, these strategies must be used in conjunction to be accurate.

3.2. ***Chainwatch*'s Detection Approach**

Chainwatch combines the above three strategies in order to provide an accurate and comprehensive view of potential threats to a blockchain network. My primary approach in designing *Chainwatch* centered around providing the following three core functions: (1) detecting chain reorganizations, (2) tracking miner hashrate share, and (3) collecting real-time blockchain statistics and providing automated data analysis.

In order to achieve the three core objectives for *Chainwatch*, major design considerations were made to ensure the scalability, availability, and accuracy of the platform. *Chainwatch* was designed to highly modular — by abstracting blockchain-interfacing logic into separate adapter components,[8] the core *Chainwatch* engine is fully blockchain-agnostic. Through this layering of abstractions, *Chainwatch* is able to communicate with blockchain client daemons to access the data needed to detect attacks without regard for the specific differences between various blockchain client implementations. This enables the software to be "plug and play," in the sense that all PoW blockchains can easily be incorporated with relatively few lines of adapter logic. As a daemon program itself, *Chainwatch* is designed to continuously monitor attached blockchains to detect suspicious events and collect detailed statistics.

[8]See Section 4 for *Chainwatch*'s implementation details.

3.2.1. Detecting Chain Reorganizations: Sliding Window Approach

To detect chain reorganizations while keeping *Chainwatch* lightweight in terms of memory usage, I developed a sliding window approach. This approach was designed with the following fact in mind: the probability of a chain reorganization occurring at a given block number decreases exponentially the further back we go. As a result, *Chainwatch* only tracks in memory the previous W blocks from the block height. As more blocks are mined and the block height N increases, blocks are added and removed from the window.

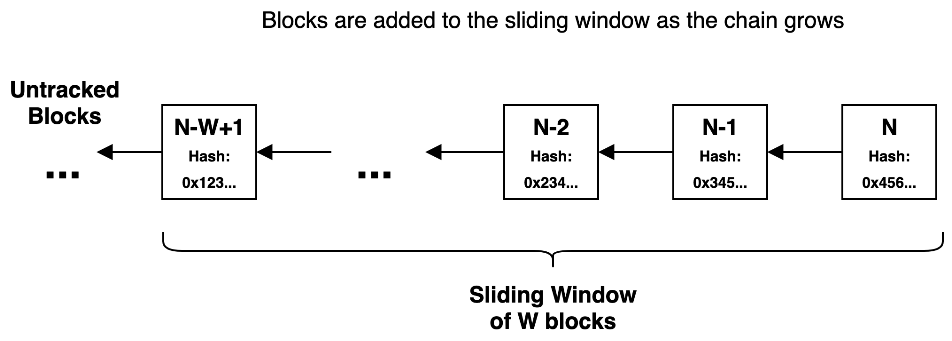


Figure 5: An example of a sliding window tracking the last W blocks.

For a blockchain at block height N, blocks $[N-W+1, N]$ are tracked by *Chainwatch*'s sliding window. For each blockchain, I have chosen a value for W that is significantly larger than the common confirmation requirement.[9] Each block is tracked in the sliding window by its block header data, such as its current block number, block hash and miner address.

For each timestep,[10] a new window is generated by adding and discarding blocks as necessary. As the window is moved forward, it is compared to the previous window starting from the highest shared block. Since the hash of the last block (position N above) is dependent on all previous blocks, a matching hash at a block position p implies that all blocks at position $< p$ also match. On the other hand, a hash mismatch indicates a chain reorganization, as some blocks must have been

[9]For example, Ethereum is tracked with a window of 200 blocks, whereas the common confirmation requirement is $\Omega = 30$ blocks.

[10]Exact timestep values are dependent on the underlying blockchain. For Ethereum, the window is updated every 15 seconds.

orphaned. In that case, the *Chainwatch* engine will process blocks in reverse order to determine the exact number of blocks that have been reorganized.

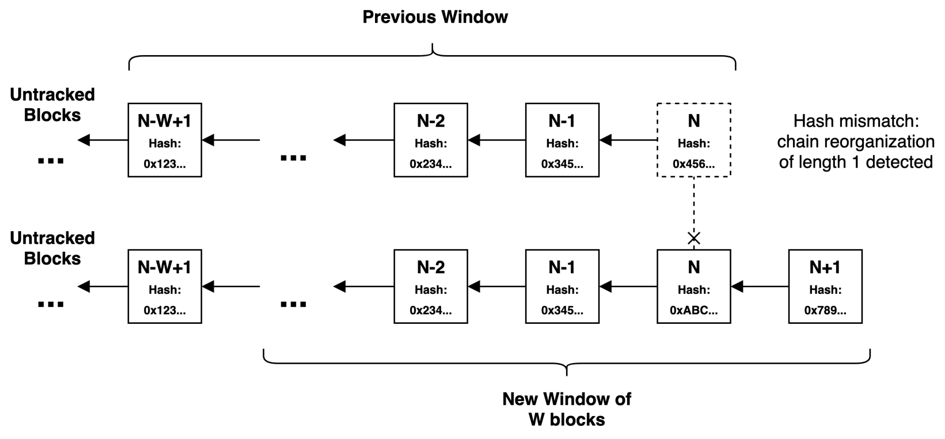


Figure 6: A mismatched hash indicates a chain reorganization.

In Figure 6, upon updating the window to include block $N+1$, *Chainwatch* detects that the hash of the original block N (0x456...) no longer matches the hash of the N^{th} block in the new window (0xABC...). Since the hashes of the $(N-1)^{\text{th}}$ block still matches, we can stop comparing hashes and conclude that a chain reorganization of length 1 has occurred, and the original block at position N block has now been orphaned.

This sliding window approach enables *Chainwatch* to detect chain reorganizations of almost any feasible depth. Furthermore, its ability to prune blocks enables it to be very memory-efficient and lightweight. On the other hand, keeping a detailed window allows the program to record a full picture of any orphaned blocks even after they are discarded from the canonical chain.

3.2.2. Tracking Miner Hash Share: Miner Density Analysis

In order to track the hash share of each miner over time, *Chainwatch* collects statistical data at regular block intervals.[11] While it is impossible to know exactly what hashrate each miner is contributing to the network, we can tally the number of blocks produced by each miner. If a miner M mines n blocks out of an interval of I blocks, we can determine that the expected value of n as a function of their hashrate h_M and the network's total hashrate h_N. Since M's probability of

[11]For Ethereum this interval is set at 20 blocks.

generating a given block is independent at $\frac{h_M}{h_N}$, the expected value of n is:

$$E[n] = I * \frac{h_M}{h_N}$$

This implies that the miner's hashrate share $\frac{h_M}{h_N}$ is equal to $\frac{n}{I}$. From this, we can easily compute each miner's expected hashrate,

$$h_M = h_N * \frac{n}{I}$$

In the case that a miner has produced over 50% of the blocks in the last interval, such that $h_M > 0.5h_N$, *Chainwatch* will flag this interval as a *miner density event*. Unless M had been incredibly lucky in generating these blocks, it is most likely that they possess the capability for a 51% attack.

3.2.3. Tracking Network Hashrate: Statistics Collection

Alongside collecting miner density statistics, *Chainwatch* also regularly collects data on the average blocktime, difficulty, and network hashrate. Difficulty adjustments in most blockchains tend to occur over long periods of time, so for each interval of I blocks, *Chainwatch* records the most recent difficulty and also averages the time to produce I blocks by comparing the timestamps of the first and last block. In order to compute the total hashrate of the network, we divide the difficulty value by the average block time. All of this data is recorded and logged, and I is chosen to be a small value such that the expected wait between each data collection event is around 5 minutes. This adds granularity and resolution to the collected statistics.

3.2.4. Data Presentation

The *Chainwatch Dashboard* is designed to provide a comprehensive view of all of the collected statistics as well as any detected chain reorganization events and miner density events. This enables the results of *Chainwatch* to be publicly accessible by anyone, which enables transparent monitoring (in contrast to the proprietary software used by exchanges). The main goals of this website were to display as much information as possible while still remaining user-friendly.

4. *Chainwatch*'s Implementation

4.1. Implementation Overview

Chainwatch's implementation centers around three major components: (1) the core Watcher engine, (2) the public API hosting web server, and (3) the *Chainwatch* Dashboard website located at chainwatch.info. The primary technologies and frameworks that *Chainwatch* was built upon include Node.js, Express, MongoDB, and HTML+CSS, all hosted on an Ubuntu Linux server. Much thought was devoted to implementing the ideas behind *Chainwatch* in a modular, available, and lightweight manner.

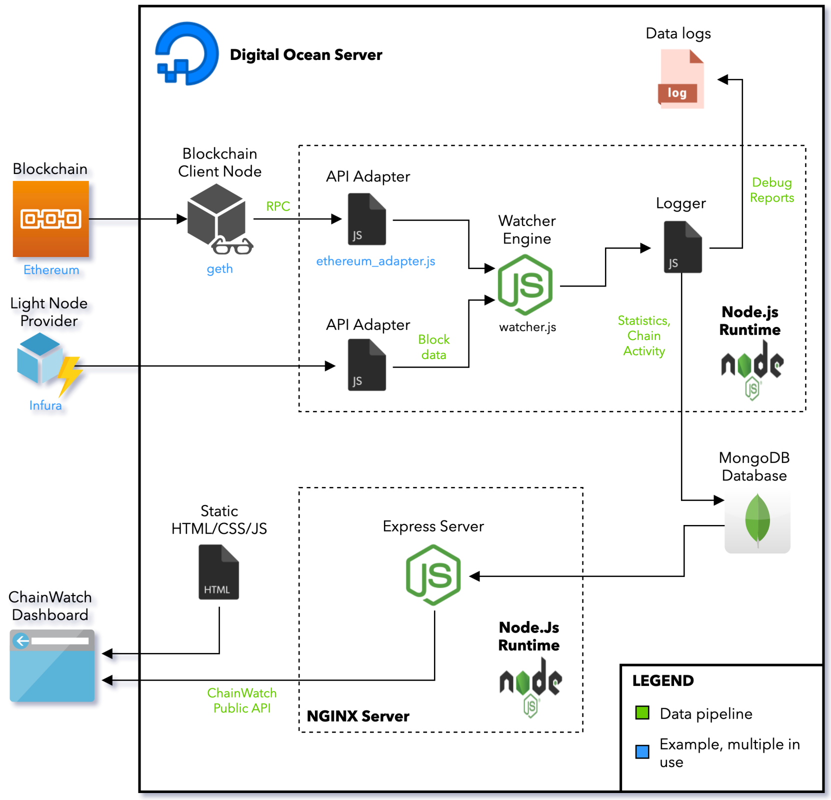


Figure 7: An overview of Chainwatch's system design.

4.2. Cloud Hosting

In order to support the various components of *Chainwatch* as well as support blockchain clients, I decided to host the core *Chainwatch* code on a DigitalOcean Ubuntu server. This was chosen over alternatives like Amazon Web Services EC2 because of the affordability of DigitalOcean, especially in terms of solid-state drive storage. This was essential in supporting blockchain client nodes. Hosting *Chainwatch* on the cloud enabled me to keep the client nodes and Watcher engine code running at all times.

4.3. Accessing Blockchain Data

To access data from various blockchain networks, I installed and configured client nodes that would sync up with their corresponding network. For this project, I decided to implement support for the Ethereum blockchain as well as Ethereum's Ropsten testnet. Using *go-ethereum* (*geth*) [9], the most common Ethereum client implementation, I was able to launch two full nodes on the Ubuntu server. As full nodes, these clients would sync up the full history of both Ethereum and Ropsten (totally hundreds of gigabytes in size), and continue to process new blocks as miners broadcast them to the network. Note that this client is not a miner; instead, having a comprehensive record of the two chains enabled the rest of *Chainwatch* to access block data efficiently.

Next, I needed adapter logic to interface with the API of these client nodes. *geth* supports an API known as *web3* through a remote procedure call (RPC) channel, so I developed Ethereum and Ropsten adapter code to connect to these ports in order to query the node for block header data. Communicating through a common API interface, the core engine could then query data through adapters with the following API:

- initProvider(): creates a connection to the blockchain client through RPC
- getBlockNumber(): queries the client to return the block height of the highest known block
- getBlock(n): returns the block header of the n^{th} block

This implementation enabled the core *Chainwatch* logic to be abstracted away from specific implementation details of various blockchain clients. Since most, if not all, clients can support such an interface, this adapter code enables a straightforward integration of additional blockchains.

Aside from connecting to full nodes hosted on the Ubuntu hosting server, *Chainwatch* adapter code also supports third-party blockchain data providers. A *light client* implementation involves querying full nodes that serve as a data provider. Services such as Infura [2], allow light clients to query for block data through a public API. However, this introduces a trust assumption into the quality of received data, and for security and performance reasons I decided against this.

4.4. Core Watcher Engine

The Watcher engine, developed in JavaScript, forms the core of the *Chainwatch* program. To ensure that *Chainwatch* remains monitoring the networks at all times, this engine is run in a Node.js runtime with the Unix screen command. Blockchain adapter-specific dependencies are abstracted away from the core engine code through dependency injection; in turn, this allows the Watcher engine to be completely blockchain-agnostic.

The engine's primary function is detecting chain reorganizations with the sliding window approach detailed in Section 3.2.1. this logic works by moving the window at a set interval (set at 15 seconds for Ethereum and Ropsten[12]), and recursively comparing hashes if a mismatch is detected. Upon a detection of a chain reorganization, the engine passes its window data onto the logger service, where it is processed and stored.

The second functionality of the engine is to detect miner density events, where a miner has an estimated hash share of over 50%. Since each miner has a unique address, the engine implements this by tallying the number of blocks each miner produces within the previous 40 blocks. This number was chosen because it is an average of the most common Ω value used by exchanges and vendors. If a miner density event is detected, relevant block header data is also passed onto the logger.

Finally, the Watcher engine is tasked with collecting the following statistical data every I blocks.

[12]This is because the expected block time is around 15 seconds, so we expect to shift the window by one block upon each timestep

This interval I is set as 20 blocks for both Ethereum and Ropsten (data is collected around every 5 minutes). The engine uses its stored window to calculate the average block time, difficulty, and hashrate for each interval. This data is also sent for processing through the logger.

4.5. Data Pipeline

The logger is responsible for processing and storing all data that is produced by the Watcher engine. Its primary function is to convert block header data into JSON, then storing it in a MongoDB database. This database is supported by a local MongoDB daemon hosted on the same Ubuntu server. MongoDB was chosen because of its ability to both store complex JSON data as well as its support of complex queries that process stored JSON objects.

This data pipeline design enables *Chainwatch* to be modular and compartmentalized. Data is supplied to the database on an append-only basis, and this data is delivered to the public API server on a read-only basis. This ensures that all data collected is immutable.

The data that this database stores is sorted into three collections (analogous to SQL tables): *reorg_events*, *density_events*, and *statistics*. The data stored in each collection adhere to the following format:

Table 2: The *reorg_events* database collection

Key	**Value**
"_id"	Unique document identifier
"network"	Blockchain network name, e.g. "ethereum"
"numBlocks"	Total number of reorganization, equal to the number of orphaned blocks
"detected"	ISO timestamp of when reorganization was detected
"start"	First block position of reorg
"end"	Last block position of reorg
"blocks"	JSON mapping of *block position* $\rightarrow$ *block headers* for affected blocks

Table 3: The *density_events* database collection

Key	Value
"_id"	Unique document identifier
"network"	Blockchain network name, e.g. "ethereum"
"numBlocks"	Interval size (currently set at 40 blocks)
"detected"	ISO timestamp of when miner density event was detected
"majorityMiner"	Unique address of miner with majority hash share
"start"	First block position of interval
"end"	Last block position of interval
"miners"	JSON mapping of *miner address* $\rightarrow$ *# of blocks mined* for each miner

Table 4: The *statistics* database collection

Key	Value
"_id"	Unique document identifier
"network"	Blockchain network name, e.g. "ethereum"
"timestamp"	ISO timestamp of collection
"blockTime"	Average time between blocks over the collection interval (unit: seconds)
"difficulty"	Most recent difficulty value in (unit: hashes)
"hashrate"	Estimated total hashrate of network (unit: hashes per second)

4.6. API Server

Using Nginx as a reverse-proxy and a web server, the API server is designed to process and serve client requests for *Chainwatch* data. Sandboxed in a separate Node.js runtime, this server uses Express as a web application framework to support HTTP requests. Hosted at chainwatch.info/api/*endpoint*, the API server supports three endpoints, one for each database collection. These endpoints enable clients to retrieve JSON formatted data stored by the Watcher engine.

4.7. Dashboard Web Client

In order to present the massive quantities of data produced by *Chainwatch*, I have created the *Chainwatch Dashboard* to act as a comprehensive overview of the monitored blockchains. Built off of a public Bootstrap template [8], The dashboard contains visualizations of the collected data to present the information in a user-friendly manner. Through graphs, charts, and tables, the dashboard enables users to interact with the generated data; furthermore, it acts as an alert mechanism by flagging potential attacks. The web application is currently publicly accessible at chainwatch.info.

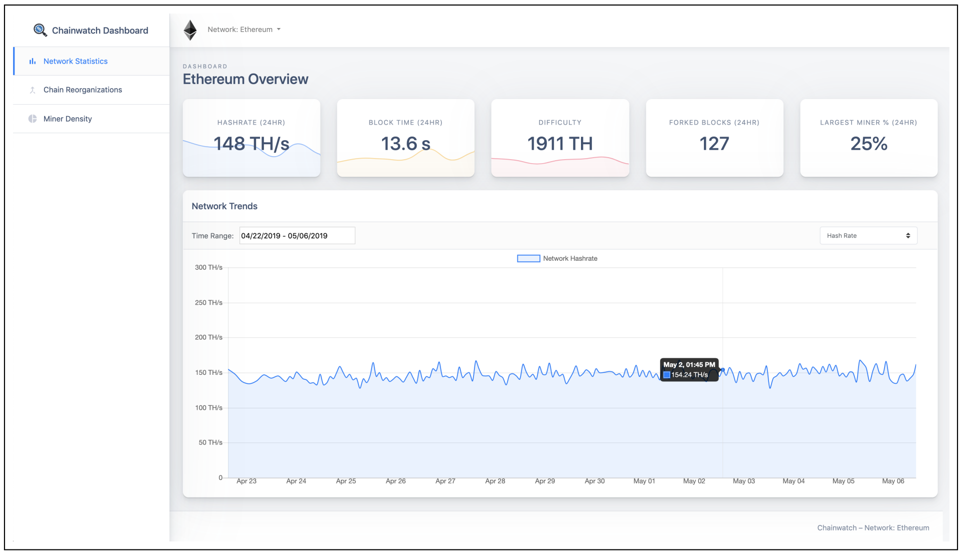


Figure 8: The statistics overview page for the *Chainwatch Dashboard*.

The statistics page presents the most important network statistics data. The main graph displays trend data for the network's hashrate, block time, and difficulty. Since *Chainwatch* collects data at a very high resolution ($<$ 5 minutes), the trend graph supports the display of smaller custom time frames. For longer ranges, I employed statistical smoothing techniques to eliminate the noise from individual data samples.

Other pages include details and visualizations of chain reorganizations and miner hash shares. In combination, these three features combine to form a complete overview of all of the data produced by *Chainwatch*. Combining the three strategies outlined in Section 3, the *Chainwatch Dashboard* provides all the functionality needed to apply this detection approach in practice.

5. Evaluation through a Controlled 51% Attack

In order to evaluate the effectiveness of *Chainwatch* in detecting 51% attacks, there is no better metric than observing how the software perform during such an attack. Because of this, I decided to launch a simulated attack on the Ropsten test network. As a testnet, Ropsten is nearly identical to the Ethereum mainnet in terms of implementation, but all cryptocurrency on Ropsten does not have any monetary value. Furthermore the total hashrate of the Ropsten network (200 MH/s) is orders of magnitudes smaller than that of Ethereum ($\approx$150 TH/s). This means that the total computational power being expended to secure the network is very low, thus making Ropsten a perfect candidate for such a controlled attack.

5.1. Experiment Design

The goal of this attack was to temporarily gain a majority hashrate in the Ropsten network, then use it to trigger chain reorganizations and execute simulated double spends. While doing so, I monitored the chainwatch.info dashboard as well as the *Chainwatch* engine logs to ensure that each of the three major detection strategies would be triggered. Since *Chainwatch* is blockchain-agnostic, a successful detection of an attack would show that the core engine logic is capable of handling the analysis of any other PoW blockchain.

To test detection of an actual 51% attack, I erected a temporary artificial network partition, hiding my mined blocks from the network. After gaining a sizable lead on the honest miners, I then removed this partition and allowed my blockchain client to announce my hidden chain to the network. This would then force a deep chain reorganization, which I expected *Chainwatch* to detect and flag.

5.2. Setup and Execution

To launch this attack, I decided to use a new Digital Ocean Ubuntu server to host a separate *geth* node. While syncing up to the blockchain was a tedious process, this was necessary because any artificial network partition would also impact *Chainwatch*'s ability to monitor the blockchain;

therefore, compartmentalizing these servers was the best choice. I installed the most commonly-used open-source Ethereum mining pool software, open-ethereum-pool [11] and opened a Stratum protocol port to connect additional hash power.

While I considered using NiceHash, I decided on using a powerful Amazon Web Services (AWS) p3.8xlarge EC2 instance instead, as AWS enabled me to install and monitor custom mining software. With 4 NVIDIA V100 GPUs, this instance was capable of computing around 500 megahashes per second for Ethereum and Ropsten's Ethash algorithm [1]. As the total Ropsten network hashrate was less than half of this amount, this single instance granted me over 70% of the total hashrate. After installing *ethminer*, an open-source C++ Ethash mining software,[12] I connected this AWS server to the open Stratum port on my DigitalOcean server and began mining.

After my *geth* client on my DigitalOcean server began registering that my generated blocks were being accepted, I was ready to launch the main attack. By blocking the inbound and outbound TCP and UDP ports 30301 and 30303, I essentially cut my *geth* client off from the rest of the network. While unable to discover peers or broadcast discovered blocks, *geth* continued to operate normally in the face of this apparent network partition. I began developing a hidden chain at twice the rate of the rest of the network. I tracked this by using the *geth* client on my main *Chainwatch* server (which was still connected to the network). After gaining a lead on the honest miners, I would then unblock the ports and allow *geth* to broadcast its chain. I repeated this a couple of times to collect more data, and I finished with a deep chain reorganization of 92 total blocks. While doing so, I placed a cryptocurrency transaction between two of my addresses on the unblocked server. Upon the reorganization, the block containing this transaction was orphaned and the balance returned to my address, which indicated a successful double spend.

5.3. Results

Chainwatch performed very satisfactorily during this attack. Besides triggering chain reorganization and miner density events as expected, the surprising resolution and detail of its statistics collection enabled me to monitor the attack in real-time.

As soon as I began mining with my AWS instance, the *Chainwatch Dashboard* view showed a significant spike in network hashrate. The estimated Ropsten hashrate rose from an average of 200 MH/s to around 500-700 MH/s. From the dashboard graph view, it immediately became apparent that an attack was occurring.

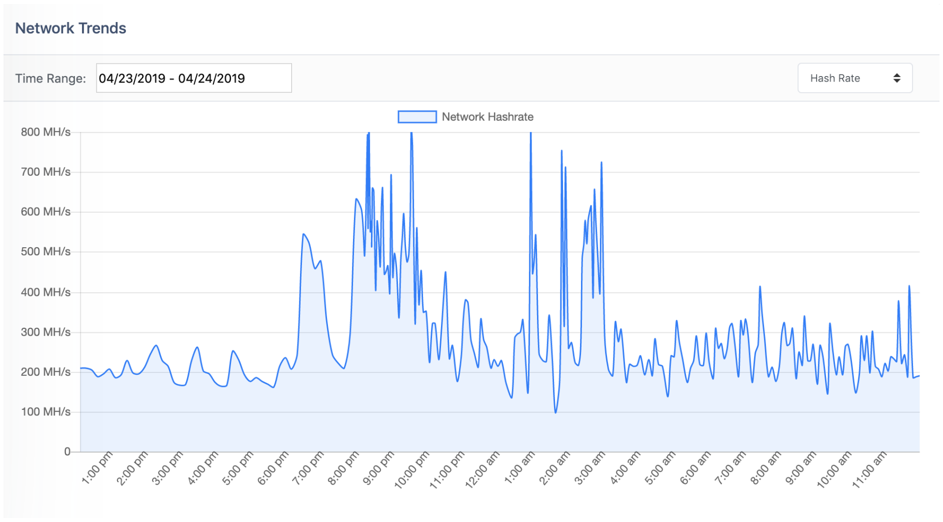


Figure 9: A *Chainwatch* graph showing the Ropsten hashrate between April 23 and 24.

In the above graph, the peaks and troughs roughly correspond to when my mining pool was broadcasting to the rest of the network (as the AWS miner was not running throughout the entire time range[13]. I used Amazon's *CloudWatch* monitoring software to produce a graph of my instance's CPU utilization over this same time range.

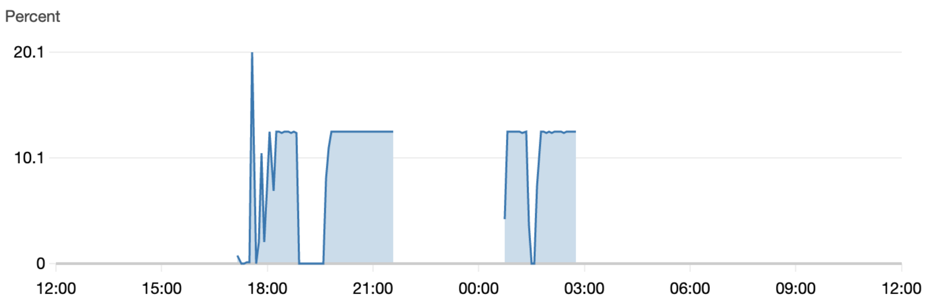


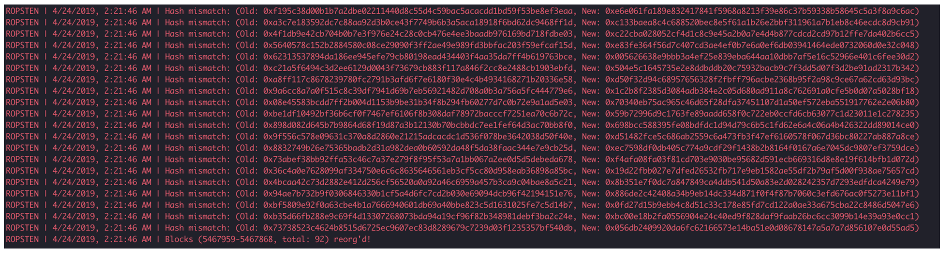
Figure 10: A *CloudWatch* graph of CPU utilization showing when the EC2 instance was in use.

The peaks and troughs of the measured hashrate and miner CPU utilization[14] are clearly matching.

[13]This is because I was also working on a problem set, so I turned off the EC2 when not in use.
[14]While I would have preferred to track GPU utilization, this feature cost money.

Moreover, the timing of these extrema also match up with the times at which I partitioned my mining pool off from the server. Dramatic peaks followed by shallow troughs roughly reflect the timing of each time the artificial network partition was placed and lifted.

On the topic of chain reorganization events, each time I unblocked the *geth* networking ports, *Chainwatch* was quickly able to detect the triggered chain reorganizations. While the Chain Reorganization section of the dashboard had not been completed at this point, this data still visible to me in real-time from my logger. As shown below, the logger output displays the block numbers and hashes for the blocks have been orphaned/reorganized. As deep chain reorganization events possess the lowest false negative rate in 51% attack detection,[15], this trigger demonstrated a high degree of confidence in detecting the attack.



```
ROPSTEN | 4/24/2019, 2:21:46 AM | Hash mismatch: (Old: 0xf195c38d00b1b7a2dbe02211440d8c55d4c59bac5acacdd1bd59f53be8ef3eaa, New: 0xe6e061fa189e832417841f5968a8213f39e86c37b59338b58645c5a3f8a9c6ac)
ROPSTEN | 4/24/2019, 2:21:46 AM | Hash mismatch: (Old: 0xa3c7e183592dc7c88aa92d3b0ce43f7749b6b3a5aca18918f6bd62dc9468ff1d, New: 0xc133baea8c4c688520bec8e5f61a1b26e2bbf311961a7b1eb8c46ecdc8d9cb91)
ROPSTEN | 4/24/2019, 2:21:46 AM | Hash mismatch: (Old: 0x4f1db9e42cb704b0b7e3f976e24c28c0cb476e4ee3baadb976169bd718fdbe03, New: 0xc22cba028052cf4d1c8c9e45a2b0a7e4d4b877cdcd2cd97b12ffe7da402b6cc5)
ROPSTEN | 4/24/2019, 2:21:46 AM | Hash mismatch: (Old: 0x5640578c152b2884580c08ce29090f3ff2ae49e989fd3bbfac203f59efcaf15d, New: 0xe83fe364f56d7c407cd3ae4ef0b7e6a0ef6db03941464ede0732060d0e32c048)
ROPSTEN | 4/24/2019, 2:21:46 AM | Hash mismatch: (Old: 0x62313537894da186ee945efe79cb80198ead434403f4aa35da7ff4b619763bce, New: 0x005626638e9bbb3a4ef25e839eba644aa10dbb7af5e16c52966e401c6fee30d2)
ROPSTEN | 4/24/2019, 2:21:46 AM | Hash mismatch: (Old: 0xc21a5f6494c3d2ee6129d0043f73679cb883f117a846f2cc8e2488cb1903ebfd, New: 0x504e5c1645735e2e8ddbddb20c75932bacb9c7f3dd5d07f3d2be91ad2317b342)
ROPSTEN | 4/24/2019, 2:21:46 AM | Hash mismatch: (Old: 0xa8ff117c8678239780fc2791b3afd6f7e6180f30e4c4b4934168271b20336e58, New: 0xd50f32d94c68957656328f2fbff796acbe2368b95f2a98c9ce67a62cd63d93bc)
ROPSTEN | 4/24/2019, 2:21:46 AM | Hash mismatch: (Old: 0x9a6cc8a7a0f515c8c39df7941d69b7eb56921482d708a0b3a756a5fc444779e6, New: 0x1c2b8f2385d3084adb384e2c05d680ad911a8c762691a0cfe5b0d07a5028bf18)
ROPSTEN | 4/24/2019, 2:21:46 AM | Hash mismatch: (Old: 0x08e45583bcdd7ff2b004d1153b9be31b34f8b294fb60277d7c0b72e9a1ad5e03, New: 0x70340eb75ac965c46d65f28dfa37451107d1a50ef572eba551917762e2e06b80)
ROPSTEN | 4/24/2019, 2:21:46 AM | Hash mismatch: (Old: 0xbe1df10492bf36b6cf0f7467ef6106f8b308daf78972bacccf7251ea70c6b72c, New: 0x59b72996d9c1763fe89aadd658f0c722eb0ccfd6cb63077c1d23011e1c278235)
ROPSTEN | 4/24/2019, 2:21:46 AM | Hash mismatch: (Old: 0x898d082d645b7b9864d68f19d87a3b12130b70bcbbdc7ee1fef64d3ac70bb8f0, New: 0x698bcc588395fe08bdfdc1d94d79c6b5c1fd62e6a4c06a4b426322dd89014ce0)
ROPSTEN | 4/24/2019, 2:21:46 AM | Hash mismatch: (Old: 0x9f556c578e09631c370a8d2860e21215adcacdc1d536f078be3642038d50f40e, New: 0xd51482fce5c686ab2559c6a473fb3f47ef6160578f067d36bc80227ab887a8ce)
ROPSTEN | 4/24/2019, 2:21:46 AM | Hash mismatch: (Old: 0x8832749b26e75365badb2d31a982dea0b60592da48f5da38faac344e7e9cb25d, New: 0xec7598df0db405c774a9cdf29f1438b2b8164f0167a6e7045dc9807ef3759dce)
ROPSTEN | 4/24/2019, 2:21:46 AM | Hash mismatch: (Old: 0x73abef38bb92ffa53c46c7a37e279f8f95f53a7a1bb067a2ee0d5d5debeda678, New: 0xf4afa08fa03f81cd703e9030be95682d591ecb669316d8e8e19f614bfb1d072d)
ROPSTEN | 4/24/2019, 2:21:46 AM | Hash mismatch: (Old: 0x36c4a0e7628099af334750e6c6c8635646561eb3cf5cc80d958eab36898a85bc, New: 0x19d22fbb027e7dfed26532fb717e9eb1582ae55df2b79af5d00f938ae75657cd)
ROPSTEN | 4/24/2019, 2:21:46 AM | Hash mismatch: (Old: 0x4bcaa42c73d2882e412d256cf56520a0a92a46c6959a457b3ca9c04bae8a5c21, New: 0x8b351e7f0dc7a847849ca4ddb541d50a83e2d028242357d7293edfdca4249e79)
ROPSTEN | 4/24/2019, 2:21:46 AM | Hash mismatch: (Old: 0x94ae7b732b9f0306846330b1cf5a4d6fc7cd2b030e69094dcb96f42194151e76, New: 0x886de2c42408a34b9eb14dc334d871f0f4f87b7060c3efd676ac0f5273e11bf1)
ROPSTEN | 4/24/2019, 2:21:46 AM | Hash mismatch: (Old: 0xbf5809e92f0a63cbe4b1a7666940601db69a40bbe823c5d1631025fe7c5d14b7, New: 0x0fd27d15b9ebb4c8d51c33c178e85fd7cd122a0ae33a675cba22c8486d5047e6)
ROPSTEN | 4/24/2019, 2:21:46 AM | Hash mismatch: (Old: 0xb35d66fb288e9c69f4d13307268073bda94a19cf96f82b348981debf3ba2c24e, New: 0xbc00e18b2fa0556904e24c40ed9f828daf9faab26bc6cc3099b14e39a93e0cc1)
ROPSTEN | 4/24/2019, 2:21:46 AM | Hash mismatch: (Old: 0x73738523c4624b8515d6725ec9607ec83d8289679c7239d03f1235357bf540db, New: 0x056db2409920da6fc62166573e14ba51e0d08678147a5a7a7d856107e0d55ad5)
ROPSTEN | 4/24/2019, 2:21:46 AM | Blocks (5467959-5467868, total: 92) reorg'd!
```

Figure 11: Truncated logger output of *Chainwatch* upon a deep chain reorganization.

Finally, I observed *Chainwatch* flagging miner density events throughout the entire 51% attack. After my deepest chain reorganization of 92 blocks, it triggered a miner density event for my pool as it had 100% hash share of the entire monitoring interval of 50 blocks. This showed that miner density events can be used as more than just preventative analytics – this event serves as confirmation that the deep chain reorganization was conducted by a malicious attacker and not the result of a natural network partition.

In conclusion, this controlled attack demonstrated that *Chainwatch* is fully capable of real-time 51% attack detection through a variety of data metrics.

[15]The further back in history a chain reorganization occurs, the probability of it occurring as a benign phenomenon decreases exponentially

6. Future Work

While *Chainwatch* has proven itself to be effective, there are still much that could be added to augment the platform. One of the most pressing needs is the ability to add support for additional blockchains, something that was unable to done due to storage space limitations and performance concerns. Due to the simplicity of integrating additional blockchains through implementing the adapter API, monitoring additional chains can be easily achieved with a few more lines of code (and an expanded SSD). *Chainwatch*, in its current state, requires uses to actively monitor the *Chainwatch Dashboard* in order to detect 51% attacks. Through some additional data analysis, the core detection logic could be extended to relay some form of a public alert to the broader ecosystem in cases where a 51% is highly likely. For example, automated Twitter posts could help spread awareness about potential threats and enable vendors and exchanges to take timely preventative measures. There are a myriad of ways that the scope of *Chainwatch* could be expanded to add additional functionality, but it has nevertheless achieved its core goal of detecting 51% attacks.

7. Conclusion

No proof-of-work blockchain is immune to a 51% attack — while larger networks are relatively more secure, 51% attacks still loom over the blockchain ecosystem as an ever-present threat. As a result, the timely detection of said attacks is critical in enabling preventative measures from being undertaken to mitigate their damage. This includes freezing cryptocurrency deposits on exchanges or increasing confirmation requirements, which can enable a vendor to protect themselves from double spends. Now more than ever in the wake of the high-profile Bitcoin Gold and Ethereum Classic attacks, 51% attack detection is desperately needed to secure the world's blockchain networks.

In the face of such security threats, *Chainwatch* has proven itself to be a utility capable of detecting attacks as they occur. The controlled attack on the Ropsten test network displayed the effectiveness of *Chainwatch*'s three-pronged monitoring strategy of detecting chain reorganizations, tracking miner hash share, and analyzing network statistics. In the context of real-time blockchain monitoring and attack detection, *Chainwatch* can do its part in protecting the security and growth of blockchain technology.

8. Acknowledgments

I would like to thank my advisor, Professor Arvind Narayanan, for his support, guidance, and feedback throughout this course of this semester. I could not have asked for a more helpful mentor for the last 12 weeks.

9. Honor Code

This paper represents my own work in accordance with University regulations.

References

[1] "Amazon EC2 P3." [Online]. Available: https://aws.amazon.com/ec2/instance-types/p3/

[2] "Scalable blockchain infrastructure." [Online]. Available: https://infura.io/

[3] "A survey of blockchain security issues and challenges," *International Journal of Modern Trends in Engineering Research*, vol. 4, no. 3, p. 57–61, 2017.

[4] Bitmain, "Antminer S9 specification," https://shop.bitmain.com/promote/antminer_s9i_asic_bitcoin_miner/specification, 2018.

[5] Blockchain.com, "Bitcoin network hash rate," https://www.blockchain.com/en/charts/hash-rate, accessed: 2019-05-05.

[6] Coinbase, "Coinbase detected a deep chain reorganization of the Ethereum Classic blockchain," Jan 2019. Available: https://twitter.com/coinbase/status/1082364714246582274

[7] Crypto51, "Cost of 51% attacks," https://www.crypto51.app/, 2018.

[8] DesignRevision, "Designrevision/shards-dashboard," Sep 2018. Available: https://github.com/DesignRevision/shards-dashboard

[9] Ethereum, "ethereum/go-ethereum," May 2019. Available: https://github.com/ethereum/go-ethereum

[10] Fortune, "Bitcoin spinoff hacked in rare '51% attack'," http://fortune.com/2018/05/29/bitcoin-gold-hack/, 2018.

[11] Github, "open-ethereum-pool," Nov 2017. Available: https://github.com/sammy007/open-ethereum-pool

[12] Github, "ethereum-mining/ethminer," Feb 2019. Available: https://github.com/ethereum-mining/ethminer

[13] A. Kiayias, N. Lamprou, and A.-P. Stouka, "Proofs of proofs of work with sublinear complexity," *Financial Cryptography and Data Security Lecture Notes in Computer Science*, p. 61–78, 2016.

[14] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009. Available: http://www.bitcoin.org/bitcoin.pdf

[15] M. Nesbitt and M. Nesbitt, "Deep chain reorganization detected on Ethereum Classic (ETC)," Jan 2019. Available: https://blog.coinbase.com/ethereum-classic-etc-is-currently-being-51-attacked-33be13ce32de

[16] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," *USENIX Annual Technical Conference*, 2014.

[17] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. e0b234a, pp. 1–39, 2014. Available: https://ethereum.github.io/yellowpaper/paper.pdf