12-2022

Detecting Selfish Mining Attacks Against a Blockchain Using Machine Learing

Matthew A. Peterson

DETECTING SELFISH MINING ATTACKS AGAINST A BLOCKCHAIN USING MACHINE LEARNING

A Dissertation

Submitted to the Graduate Faculty of the
University of South Alabama
in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

in

Computing

by
Matthew Alan Peterson
B. S., Pensacola Christian College, 2013
M. S., University of South Alabama, 2017
December 2022

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ASIC Application-specific integrated circuit

BTC Bitcoin

BSV Bitcoin Satoshi's Vision

ETC Ethereum Classic

GDP Gross Domestic Product

MDI Mean Decrease in Impurity

PoW Proof of Work

PoS Proof of Stake

ABSTRACT

Peterson, Matthew Alan, Ph.D., University of South Alabama, December 2022. Detecting Selfish Mining Attacks Against a Blockchain Using Machine Learning. Chair of Committee: Todd Andel, Ph.D.

Selfish mining is an attack against a blockchain where miners hide newly discovered blocks instead of publishing them to the rest of the network. Selfish mining has been a potential issue for blockchains since it was first discovered by Eyal and Sirer. It can be used by malicious miners to earn a disproportionate share of the mining rewards or in conjunction with other attacks to steal money from network users. Several of these attacks were launched in 2018, 2019, and 2020 with the attackers stealing as much as $18 Million. Developers made several different attempts to fix this issue, but the effectiveness of the fixes is currently unknown. Despite the known vulnerability, there is little researching into detecting these attacks either historically or in real-time. In this research, we build a program to gather data from known selfish mining attacks against the Ethereum Classic blockchain. We then use this data to train a machine-learning algorithm to discover the important features for detecting selfish mining.

CHAPTER 1

INTRODUCTION

Cybersecurity plays an important role in an increasingly connected world. In this modern age of connectedness, technology disrupts previously untouchable industries and drives society forward. The financial industry has felt the disruptive influence of technology with the introduction of blockchain technology. The first and most popular application of blockchain technology is cryptocurrency. Bitcoin was invented in 2008 by a person or group of people working under the pseudonym of Satoshi Nakamoto who published the original Bitcoin white paper to the metzdowd.com cryptography mailing group [1]. Bitcoin adoption started with crypto hobbyists interested in the technology and idealists who wanted a form of money outside of government control. Since its humble beginnings, Bitcoin has soared in value to possess a market cap greater than the GDP of many first-world countries [2], [3].

As blockchain adoption increases and the value stored in cryptocurrencies rises, attackers are incentivized to target them. There have been several high-profile attacks against various cryptocurrencies in the past decade and the first high-profile attack was against the MtGox cryptocurrency exchange. MtGox, once the biggest cryptocurrency exchange in the world, was attacked in 2011 and then again in 2014 [4]. The attackers stole what at the time was $460 Million worth of Bitcoin [5]. While these attacks were

not an attack against the blockchain technology itself, they showed the world how much value was stored in blockchains and the potential gain for the attackers.

Many potential blockchain attacks exist, but one of the attacks becoming more common is the selfish mining attack which can be coupled with double-spending. Cryptocurrencies such as Monacoin, Ethereum Classic, Bitcoin Gold, Bitcoin Satoshi's Vision, Litecoin Cash, Verge, and ZenCash have fallen victim to this attack and have lost over $18 million [6]–[8].

1.1 Problem Statement

The increasing adoption of blockchain technology makes it a larger target for criminals and the unique characteristics of blockchains opens them up to attacks not present in other systems. Since most blockchains are decentralized, any method that allows attackers to control the majority of the hashing power can devastate a blockchain by compromising its integrity. Controlling 51% of the hash rate allows the attacker to reverse transactions by rewriting the blockchain history or control which transactions are allowed onto the blockchain. The lack of a central authority means that successful attacks cannot be reversed once the blocks are part of the chain. Through an attack known as selfish mining, Eyal and Sirer proved that reversing transactions is possible for attackers controlling less than 51% of the blockchain [9].

Selfish mining works by manipulating the way the Proof of Work (PoW) mining protocol works. The protocol states that when new blocks are discovered, they are to be released to the rest of the network participants. Miners engaged in selfish mining behavior keep the newly discovered blocks hidden instead of releasing them. The end

goal of selfish mining is to earn a disproportionate amount of mining rewards by making honest miners waste their mining power. Selfish mining can also be combined with other attacks to steal money from cryptocurrency exchanges or other network participants.

As selfish mining attacks become more popular, blockchain users must have a way to detect them. Detecting selfish mining poses a difficult and unique problem as selfish mining attacks exist in three potential stages. The first stage is when a miner or group of miners first launches the attack. The miners will keep their newly discovered blocks hidden and only release them when the honest miners find a new block or the selfish miners decide the attack is ready. This is the most difficult stage to detect as it is impossible to tell if miners are keeping blocks secret or simply have not found a block yet. Detecting an attack at this stage is the most proactive solution as the network can stop the attack before it takes place.

The second stage is when the selfish miners launch the attack by releasing their blocks on the network. Once the attacker releases their blocks, it will create a temporary fork in the network as the miners either accept the selfish miner's blocks or stay with the honest blocks. The miners will choose the selfish block if the chain is longer or, in the case where both chains are the same height, accept whichever block they hear about first.

There is some existing research on detecting attacks at this stage, but it is largely untouched. Detecting attacks at this stage is largely reactionary as the attacks have already been launched.

The third stage is detecting attacks historically. Once an attack has succeeded, it is preserved in the blockchain data. The problem with the historical data is that it only preserves the final blocks, and the orphan blocks (which help signify an attack) are

discarded. Unfortunately, there is currently no heuristic to tell when these attacks have occurred. Detecting attacks historically can help developers verify the results of security patches designed to mitigate this threat or help model selfish mining with historical examples.

Currently, there is no known way to proactively detect these attacks and there is limited research on detection after the attack is completed. To successfully defend against selfish mining, the network needs to detect the attack before the attacker's blocks are released on the network. By the time the attacker releases their blocks, it is already too late. Adding to this difficulty is the fact that accurately detecting all network forks is very challenging. Unless a node connects to all the other nodes on the network, it will not be able to detect all forks.

Limited research exists that discusses how to detect selfish mining. The existing research centers on how the fork rate can be used to detect an attack after the attacker releases the blocks onto the network [10]. The authors of this paper found that they could detect an ongoing attack by examining the block height of the forked chain. To date, this is the only paper that we know of that looks at how to detect a selfish mining attack. While this research is a step in the right direction, the total number of significant factors that indicate a selfish mining attack is unknown. Several additional factors such as relative revenue rate and timings between successive blocks have been suggested by different authors but remain largely unexamined [9], [11]. Research is needed to discover the significant factors that indicate a selfish mining attack.

1.2 Research Objective

To fill the current research gaps in selfish mining, our research objective is to find the significant factors that identify an attack. We plan on looking not only at the already discovered factor of block height but also at other factors such as the ones suggested by other authors. Our research centers on analyzing known historical attacks. While there may be more value in detecting ongoing and unlaunched attacks, these phases pose a data source problem and would require simulation.

Several authors attempted to prove that specific factors identify selfish mining by researching them individually. In our research, we let the data decide what the significant factors are gathering data from known attacks and then using machine learning to analyze the data. To date, we have not been able to find examples of anyone using machine learning to analyze selfish mining attacks. A few authors have discussed using Big Data techniques such as map/reduce to search through a blockchain's transactions but attacks stay untouched [12], [13].

To run the machine learning algorithm, a vast amount of data is needed. Generating data by attacking the live blockchain is generally frowned upon and monetarily prohibitive. To examine historical attacks, we used the Ethereum Classic blockchain. Criminals successfully used a selfish mining attack to steal money on the Ethereum Classic (ETC) blockchain in 2019 and then two more times in 2020 [8], [14], [15]. These attacks left examples of selfish mining in the blockchain and will allow us to examine historical data and verify detection methods.

The layout of the rest of this paper is as follows: Chapter 2 presents the background of the Proof of Work blockchain while Chapter 3 discusses the existing

blockchain research into attacks and machine learning. Chapter 4 goes into more detail on our research objective and the factors we chose to research while Chapter 5 details our research methodology and data sources. Chapters 6-9 discuss the tests we ran while chapters 10-13 discuss our results, limitations, and future work.

CHAPTER 2

BACKGROUND

Blockchains are the result of many different technologies and concepts that were combined to create the current system and each piece is important to understand how blockchains work. Blockchains, at their core, are blocks of data that are chained together by including a reference to the previous block in each new block. This reference is a hash of the previous block which prevents someone from replacing a block as it would invalidate every block from that point forward. In this chapter, we will review the block data structure and how new blocks are created.

2.1 The Blockchain Structure

Each block in the blockchain has several required pieces of information and is created through a process called mining which will be covered in section 2.2. Blockchains start with a block known as the genesis block. The genesis block is the only block that does not contain a reference to a previous block and all future blocks are built on top of this one. Although the information in the following sections applies to Bitcoin, most PoW blockchains follow a similar structure with slight variations. The size of a Bitcoin block is limited to 1MB and contains a header section and a body that holds the transaction data. The header takes up the first 80 bytes of the block and contains the

version, previous block hash, Merkel Root, timestamp, difficulty target, and nonce [16].

These fields are summarized in Table 1 and Figure 1.

Table 1. Block Header Fields. Shows the fields in the block header of the Bitcoin blockchain including their size in bytes and a description.

Field	Bytes	Description
Version	4	Version of the block. Defines what validation rules to apply to the block
Previous block hash	32	Double SHA256 hash of the previous block's header
Merkel root	32	Double SHA256 hash of all the transactions in the current block
Timestamp	4	Unix timestamp of when the miner started hashing the header
Difficulty target	4	Numeric value that the block hash must be equal to or less than to be deemed valid
Nonce	4	Random number used to change the hashed value of the block to make it conform to the difficulty target

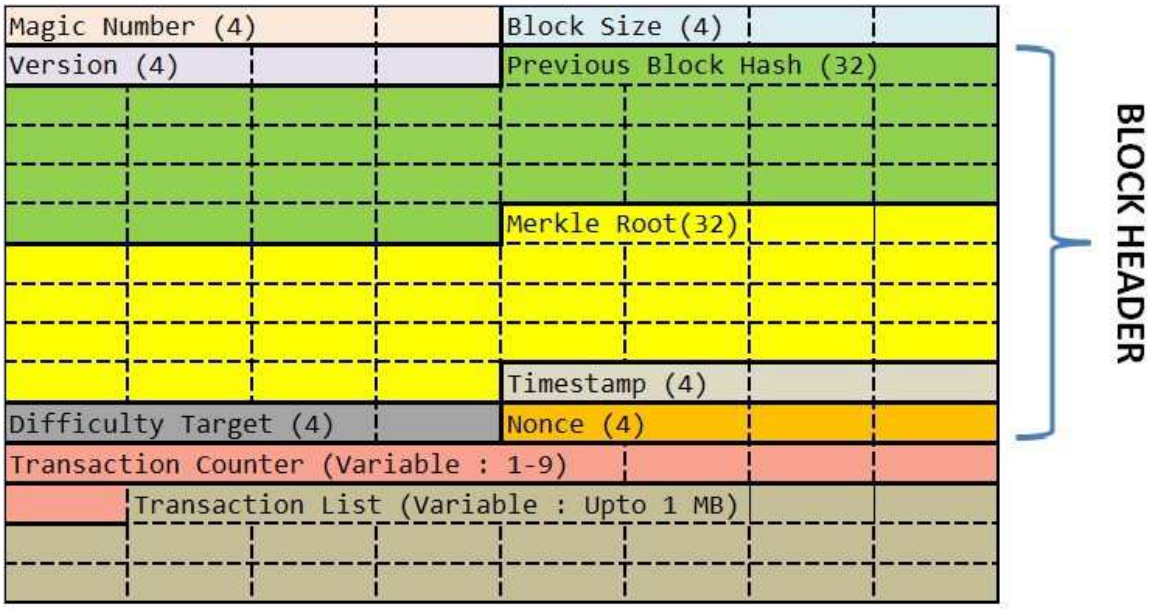

Figure 1. Block header block structure for Bitcoin. The number in brackets is the size in bytes. Each individual cell is 1 byte. Hence a field of 4 bytes occupies 4 cells. Fields from Version till Nonce form the block header (Total 80 bytes). [17]

2.1.1 Header Fields

The version field in the header indicates what version the blockchain was on when the block was added. Since Bitcoin is actively being developed, the set of rules to validate the block can change. The version field lets the network participants know what set of rules to use to validate the block.

The previous block hash field is a double SHA256 hash of the previous block's header which links the blocks into a chain. Hashing the header creates a unique number that gets embedded in the next block. If someone tries to change the data in the previous block, the hash will no longer match, and all the descendant blocks would be invalidated.

The Merkel Root is a common data structure used in many applications to track changes. The Bitcoin Merkel Root creates a bottom-up binary tree of hashes. All the transactions in the block are ordered and then hashed together two at a time. Each resulting hash pair are then hashed together again until you are left with one hash of all the hashes. The Merkel Root is used to verify that the transactions are valid.

The timestamp is a Unix timestamp of when the miner started creating the block. This timestamp can vary between network participants and the only rule is that it must be greater than the median time of the last 11 blocks and less than two hours in the future.

The difficulty target is a variable number that is shared by the network and changes every 2016 blocks. The difficulty target defines a number that the hash of the block header must be equal to or less than. According to the Bitcoin protocol, blocks are produced every 10 minutes. As more miners join the network, blocks get created faster. To keep the block rate at one every 10 minutes, the protocol will lower the value of the difficulty target, thereby making it harder to produce a block. If miners leave the

network, the protocol will increase this number to make it easier. The difficulty target is adjusted every 2 weeks (every 2016 blocks).

The nonce is a number that the miners will increase to create a new hash. When a miner hashes the block header and the resulting value is not less than or equal to the difficulty target, he must try again. To create a new hash, the miner increments the nonce value by one and tries again. The miner continues this process until an appropriate value is found. This algorithm is reviewed in section 2.2.2.

2.1.2 Transactions

After the header comes the blockchain transaction data. Transactions send money from one wallet address to another. Bitcoin uses elliptic curve cryptography to secure transactions and a wallet address is the public key of an ECDSA public/private key pair. When someone wants to start using Bitcoin, he must first create a new wallet address to receive money. Money is sent to the public key address which ensures that only the holder of the private key can access the money.

2.2 Proof-of-Work

The main problem that blockchains solve is achieving consensus in a distributed environment. As network participants send money back and forth, it is important that everyone agrees on the balance of the accounts and that no one tries to illegally modify them. In a network, it is difficult to make sure that all the nodes agree on a specific state and this difficulty is best represented by the Byzantine General's Problem [18].

2.2.1 Byzantine General's Problem

The Byzantine General's Problem is a fictional scenario where multiple generals are encamped around a fortress that they wish to attack. If the generals attack the fortress at the same time, they will win the battle but will lose if they attack separately. In this time before telephones, the generals coordinate their attack by sending messengers to confirm the attack time. If General A wishes to attack at 8 AM he will send a message with the attack time to General B. After General B reads the message, he will need to send a confirmation message back. Maybe, instead of agreeing to attack at 8 AM, General B thinks it would be better to wait, so he sends back a message telling General A to retreat. The problem is that the messengers might be captured by the enemy or one of the other generals might be a traitor and send the wrong response message to confuse the attack. What the generals need is a way to achieve consensus about the right time to attack in a distributed environment.

Instead of exchanging messages that tell armies when to attack a fortress, imagine a system where the participants exchange messages that spend money. This system needs to make sure that everyone agrees on account balances and that no one tries to steal someone else's money. In more succinct terms, the system needs a consensus mechanism. Bitcoin solves this problem and achieves consensus through a process called Proof-of-Work.

Several consensus methods exist in modern blockchains including Proof-of-Stake, but the most widely used and successful one is Proof-of-Work (PoW). Proof-of-Work was incrementally developed and existed many years before Bitcoin was invented. PoW started as a way to throttle abuse of unmetered Internet systems such as email. Invented

in 1997 by Adam Back, the process was originally called Hashcash but was adopted by Satoshi Nakamoto as a way to achieve consensus in Bitcoin [19], [20].

2.2.2 PoW Algorithm

If a user on the Bitcoin network wants to purchase a cup of coffee from his local coffee shop, he will sign a transaction with his private key that sends $5 from his wallet to the wallet controlled by the coffee shop. This transaction is broadcast to the entire network and is picked up by special nodes called miners. The miners add the transaction to their transaction pool and bundle multiple transactions together and attempt to create a new block. Every time the miner hashes the block, he compares the resulting hash to the target value. If the hash is not less than or equal to the target, the miner changes the nonce value and rehashes the block. Incrementing the nonce value results in a completely different block hash. This process is repeated until a valid hash is found or another miner publishes a new block. The simplified version of this process is shown in Figure 2.

```
Init:
    GetTransactionsFromPool()

HashTransactions:
    Hash = RunTransactionHash()

IF NewBlockPublished
    New block has published by the network
    Accept the new block
    GOTO Init

IF Hash > TargetHash
    Hash not in acceptable range
    Increment nonce value
    GOTO HashTransactions
ELSE
    This miner found a valid block
    PublishBlock()
    GOTO Init
```

Figure 2. Pseudo code of the Proof-of-Work algorithm.

This process of constantly rehashing the transactions is computationally expensive and can take a long time. On average, this process takes 10 minutes to create a new block in Bitcoin but varies for other blockchains. As miners join the network it shortens the time to mine blocks or when they leave, it takes longer. The protocol automatically adjusts the difficulty target to keep the average block rate at 10 minutes. Miners are financially incentivized with a reward for finding a new block and charging small fees for each transaction included in the block. Each block includes a special transaction known as a coinbase transaction that creates new cryptocurrency and sends it to whatever wallet address the miner chooses.

After the miner finds the new block, he broadcasts it to the rest of the network. Each network participant will validate the block information and if the block height is

greater than his last known block, he will add it to his blockchain. Miners who were working on finding a block will adopt the new block and start the mining process again with new transactions.

2.2.3 Fork Resolution

A situation can arise where two miners find a new block at the same time. In this case, each node participant will accept the block they hear about first and reject the other. A disagreement between the network nodes about the latest block is known as a fork. When the network is in a forked state, the miners carry on as if nothing is wrong and attempt to build the next block on whichever block is at the head of their blockchain. The fork is resolved by whichever miner mines the next block due to the principle of the longest chain. The Bitcoin protocol states that in the case of a network fork, a node will accept the longest chain it knows about. The longest chain is the one with the most blocks or, congruently, the most proof-of-work. A fork is resolved by the miner who creates the longest chain by finding the next block. Since this miner's chain is now the longest, all the nodes will switch to his when they hear about it.

Since a network fork can switch the latest accepted block, merchants generally do not trust a payment until two additional blocks have been added to the chain. Recalculating blocks becomes exponentially cost-prohibitive since one would need enough computing power to recalculate the target block and every block built on top of it to create a longer chain.

This process of hashing transactions and adding them to the block allows Bitcoin to be Byzantine Fault Tolerant. Any disagreements in the state of the blockchain are resolved by the principle of the longest chain. The chain is safe from alteration since each

block contains a hash of the previous block. No miner or group of miners can replace blocks by creating a longer chain since PoW makes it computationally infeasible to create blocks faster than the rest of the network.

CHAPTER 3

RELATED WORKS

Any system that contains sensitive or costly information will eventually come under attack by individuals who wish to exploit them for personal gain. Blockchains are no outlier and many different types of attacks have been discovered. In this section, we will go over the attack vectors used to launch a selfish mining attack. Following the convention of Hasanova, Baek, Shin, Cho, and Kim, we categorize these attack vectors into General Risk and Private Forking and Pool Attacks [21].

3.1 General Risk

3.1.1 Double Spending Attacks

One of the possible attacks against blockchains, and an important piece in the 51% attack, is a double-spend attack. In a double-spend attack, a malicious user seeks to spend the same funds more than once. Whenever a participant in the peer-to-peer network wishes to spend money, he sends a transaction to all his connected peers. These peers forward this transaction to all their peers until the transaction has been propagated across the entire network. Each node on the network receives broadcasted transactions and miners will bundle multiple transactions together and mine them into a block that gets put

on the blockchain. On average, it takes 10 minutes for a batch of transactions to be mined and included on the Bitcoin blockchain. While it is impossible for duplicate transactions to be included in these mined blocks, a malicious user can take advantage of the slowness of the mining process to double-spend funds.

While the Bitcoin protocol keeps the average block time at 10 minutes, many merchants need to process payments faster. For many service-oriented businesses, waiting 10 minutes is an unacceptable amount of time. Researchers defined Bitcoin transactions where the merchant does not wait for a block transaction as *fast payment* transactions [22]. In fast payments, the customer queues a transaction in the blockchain and then immediately receives the purchased product without waiting for a block confirmation. In their paper, Karame, Androulaki, and Capkun examined fast payments to see how vulnerable they are to double-spend attacks [22]. They researched how successful double-spend attacks were against vendors who had varying numbers of connected peers. The test involved a vendor node, an attacker node, and one or two helper nodes for the attacker. When the attacker node purchased an item from the vendor, he immediately broadcasted a double-spend transaction that routed the money back to himself. The attacker would send this double-spend transaction to his helper nodes to propagate the malicious transaction faster than the vendor node could propagate the legitimate one. Since the Bitcoin protocol does not allow duplicate transactions, whichever transaction got propagated to the majority of the Bitcoin network would be the one included on the blockchain. By performing various tests, they were able to determine that a double-spend attack with two helper nodes had a high probability of succeeding (>

50%). These tests showed that the attack could succeed even when the attacker waited up to two seconds to send the malicious transaction.

Research has been done to come up with ways to prevent double-spend attacks. Ruffing, Kate, and Schröder, suggested a solution that involved deposits and smart contracts to create what the authors called accountable assertions [23]. Two blockchain users would deposit funds into a channel governed by a smart contract. If the paying party attempts to double-spend any of the funds, a cryptographic function called an accountable assertion would transfer the funds to a beneficiary and penalize the attacker by taking away his deposited funds. This design fell apart if the attacker colluded with the beneficiary to get his money back and had the additional problem that the person attacked is not compensated for his loss [24]. To fix these flaws, additional research came up with a way to ensure the attacker was penalized while compensating the victim. The protocol proposed in this paper solves these two problems while adding the ability to handle more than one transaction with a deposit. A possible limitation of these solutions is the required time for creating a deposit on the blockchain and then processing a consumer-to-vendor transaction.

Another possible attack vector for double-spending is using selfish mining to launch an attack. In selfish mining, a miner, or a group of miners, withholds newly discovered blocks from the rest of the network. These malicious miners try to mine a longer chain faster than the rest of the network can. Consensus is achieved in the Bitcoin network by accepting the longest chain as the correct chain. If the miners can forge a malicious chain that is longer than the honest one, the rest of the network will accept the malicious chain as the correct one when it is broadcast to the rest of the network. These

malicious miners can launch a double-spend attack by spending money on the honest chain while including a transaction in the secret chain that routes the money back to themselves [25]. Rosenfeld researched the probability of launching a successful selfish mining attack given a miner's hashing power and the number of block confirmations the merchant waits. Rosenfeld also calculated how much money the attacks would end up spending to commit an attack by creating a matrix that showed how much money the attacker would have to double-spend the make the attack worth his time. The more economical the attack is, the more likely it is to be carried out [25]. We cover selfish mining in more depth in section 3.2.1.

3.1.2 51% attack

PoW blockchains work as intended if no one group controls more than 50% of the total computing power on the network. Once someone crosses this threshold, they completely control the network and can launch double-spend attacks at will or control which transactions are allowed in each block [6]. While it is extremely cost-prohibitive for any one person to control 51% of the total hashing power, it is theoretically possible for a mining pool to achieve this. While the honest network participants still have a chance to mine the next block, the attackers can control the network by choosing to mine on their blockchain instead of adopting the longer chain from the honest miners. Since they control more than 50% of the network's hashing power, the attacker's chain will eventually overtake the honest miners which allows them to have total control over which transactions are included in the blocks.

3.2 Private Forking and Pool Attacks

3.2.1 Selfish Mining

Selfish mining attacks manipulate the way PoW systems mine blocks onto the blockchain. Blocks are added to the global blockchain by computers running the Bitcoin client software that use their computing power (referred to as hash-rate) to solve a difficult mathematical problem. As discussed in section 2.2.2, a miner creates a new block by selecting transactions and hashing them until the result conforms to the target value. The miner broadcasts this new block to the rest of the nodes on the network and then restarts the process. If the transaction hash is not below the target difficulty, the nonce value is incremented and the cycle repeats. Table 2 shows an example of the target difficulty, nonce value, and resulting hash [26].

Table 2. Example block.

Target difficulty	Nonce	Hash
12973235968799.78	4152620663	000000000000000000142f9df821e08b2713775812efef61bacf440afdc3afca

If two miners (A and B) find a block at the same time the network can split when some nodes adopt block A1 and some adopt block B1. Network participants will always adopt the first block they hear about and ignore any others. In Figure 3, miner C hears about block B1 first and subsequently adopts it. The network will resolve this split and merge back together as soon as the next block is found. If miner A finds block A2 before B or C finds a block, then blocks A1 and A2 will become part of the chain, and block B1

will be discarded. This works because the blockchain always accepts the longest chain as the valid one and discards all others. In Figure 4, both blockchains are valid but since miner A's chain is longer than B's, all the nodes on the network will adopt A's chain and discard B's.

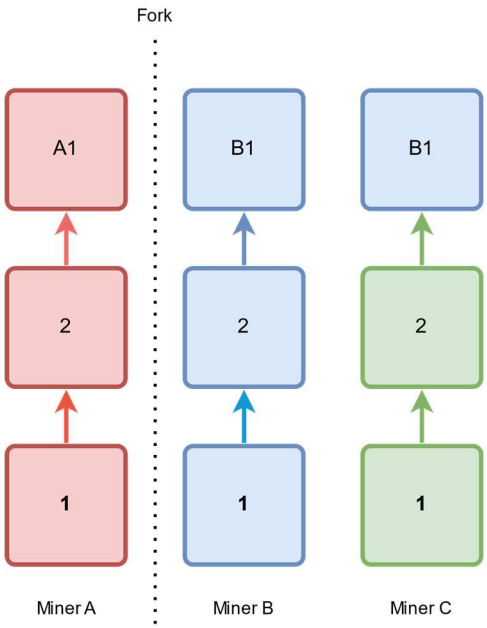

Figure 3. Blockchain fork. Miner C accepts block B1 since it was notified about it first. This creates two chains of equal length.

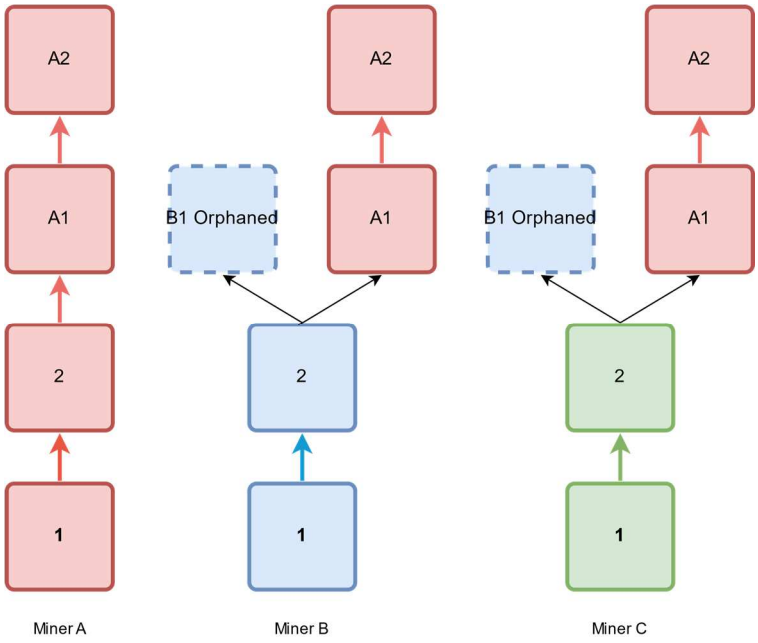

Figure 4. Blockchain fork resolution. When miner A mines and releases block A2, it causes block B1 to be orphaned for miners B and C because chain (A1 + A2) is longer than B1.

In the foundational research paper on selfish mining, Eyal and Sirer detail how a miner can earn more than their fair share of mining rewards by keeping newly mined blocks hidden instead of sharing them with the rest of the network [9]. This paper challenged the thought that a blockchain was secure if honest miners owned at least 50% of the total hashing power on the network. To launch a selfish mining attack, a malicious node tries to outpace the honest miners by building a private chain. The malicious miner will continue to work off this private chain until the rest of the network starts to catch up. Once his lead is threatened, the malicious miner reveals his private chain to the rest of the network. This causes the rest of the nodes on the network to discard the blocks they mined and lets the selfish miner reap all the block rewards. Eyal and Sirer showed that

malicious users could launch a selfish mining attack with less than 25% of the network's hashing power [9]. Furthermore, if a mining pool uses this strategy to obtain more than their fair share of the mining rewards, other miners will be incentivized to join the malicious miner, further increasing their mining power. The authors proposed a modification to the Bitcoin protocol that would change the way miners handle conflicts. Currently, when a node hears of a chain equal in length to the chain it is currently mining on, it discards the incoming chain. Their proposal would change this by having the miner randomly choose one of the two equal-length chains to mine on. This modification would require the attacker to obtain at least 25% of the hashing power to launch a selfish mining attack. The authors noted the difficulty of detecting selfish mining attacks due to inaccuracies in orphan block rates.

Building on the original selfish mining paper, Sapirshtein, Sompolinsky, and Zohar developed an algorithm to find optimal selfish mining strategies [27]. Using this algorithm to analyze Eyal and Sirer's proposed fix, they discovered that when a model takes propagation delay into account there is always a successful selfish mining strategy. If the selfish miner can propagate their block to half of the nodes on the network before the honest miner can, then they need as little as 15% of the hash rate to earn more rewards than they would from following the protocol. Since the Bitcoin network operates as a peer-to-peer network, it can naturally split when two miners broadcast a newly discovered block at the same time. This natural division of the hashing power of the network increases the probability of a selfish mining attack to the point that a profitable selfish mining attack is possible for a miner with any amount of hashing power [27].

Contrary to the established research on selfish mining, Gobel, Keeler, Krzesinski, and Taylor published research that suggested that selfish mining, as defined by Eyal and Sirer, was not optimal in the presence of network delays [11]. By using a Markov chain model, they show that strategies such as selfish mining increases the rate of orphan blocks and then run a simulation of the network with active selfish mining.

3.3 Simulation of the Bitcoin Network

Blockchain research has shown the need to simulate blockchain workings outside of the production network. Researchers need a way to create specific network conditions quickly and reliably without harming normal operation. We initially wanted to simulate selfish mining attacks and reviewed the current literature for a simulator. Although several simulation programs exist, most are too complicated or have fallen into disrepair. Shadow-Bitcoin was a program that built the Bitcoin client on top of a parallel discrete-event network simulator known as Shadow [28]. Despite its potential, support for Shadow-Bitcoin was dropped in February 2020. Gervais et al. created a simulator to examine the security and performance of blockchains but did not open-source the code [29]. They modeled the geographical download latencies and miner distributions from Verizon and bitnodes.21.co respectively [30]. They compared the results of their simulator to the Bitcoin network and found that it produced comparable results. Two other simulators were released in 2019 with similar names of BlockSim and SimBlock [31], [32]. The creators of BlockSim saw the limitations of Shadow-Bitcoin and wanted to create a simulator that had the flexibility to simulate many different blockchain protocols and consensus mechanisms. BlockSim was written in Python and the program

was validated using block and transaction measurements taken from the Ethereum network [31]. The creators of SimBlock also noted the difficulties in properly simulating a blockchain network and used Java to create their simulator. To validate the program output, the authors compared the simulator to measurements taken from the Bitcoin network along with measurements by Gervais et al. [29].

3.4 Bitcoin Network Topology

Any extension to a simulator will require validation to ensure the generated data matches the results from the real network. Several published papers have already made significant progress in measuring the way the Bitcoin network performs. Decker and Wattenhofer wrote the foundational paper on how the Bitcoin network propagates blocks and transactions. The authors performed their measurements by connecting a research node to a large sample of network nodes and captured transaction and block messages over 70 days [33]. Their research allowed them to measure the time between block discoveries and the network fork rate. Another frequently-cited Bitcoin network paper examined the network topology and how nodes are connected [34]. Miller et al. discovered that rather than behaving as a random graph, the network is filled with super influential nodes and 2% of the nodes are responsible for 75% of the total network mining power [34]. Attackers that use network information to obtain a broadcast advantage have more opportunities to cheat the system.

CHAPTER 4

RESEARCH OBJECTIVE

The object of this research is to discover what the important factors are for detecting a selfish mining attack. As mentioned in Chapter 1, there are three different phases of a selfish mining attack. The three phases are the initialization of the attack, the launching of the attack, and the aftermath of the attack. We wanted to find the significant factors for each phase of the attack.

4.1 Feature Selection

To find the significant factors, we started by reviewing several features that are available from the blockchain data or that have been suggested as factors by previous authors. Table 3 at the end of this section shows a summarization of the analyzed features that are presented below.

4.1.1 Fork Height

As shown by Chicarion, Albuquerque, Jesus, and Rocha, the block height of a fork can be used to detect a launched selfish mining attack [10]. They based their detection algorithm on observations from the data gathered from selfish mining simulations. If a blockchain fork had a block height greater than or equal to two, they classified it as an attack. Unfortunately, this resulted in some false negatives from forks that only had one

block and it cannot detect historical or unlaunched selfish mining attacks. Additionally, Decker and Wattenhofer noted the difficulty in detecting all blockchain forks [33]. While this feature does have some limitations, the authors showed that it is worthwhile to include in research into selfish mining.

Finding block heights is only possible by accessing a network node. In Bitcoin, one must host a network node, or have access to a long-running node and run the getchaintips command. Getchaintips returns a list of all the network forks that the node has seen since it joined the network. Getchaintips has the inherent limitation that the node needs to have been notified about a fork in the network. As noted by Decker and Wattenhofer, only nodes on the edges of a fork will know the fork exists. This makes it difficult to accurately capture all forks in the network.

In the Ethereum blockchain, miners are incentivized to include information about blockchain forks in a mined block. These orphaned blocks (called "uncles' in Ethereum) grant miners a reward for producing the block even if it is not accepted in the final blockchain. While this does make it possible to find information on honest blockchain forks, there is no guarantee that a selfish miner will include uncle blocks.

4.1.2 Fork Rate

Multiple papers have discussed the importance of the fork rate in the detection of selfish mining. The original selfish mining paper by Eyal and Sirer noted that an increase in the orphaned blocks (caused by a fork) could indicate selfish mining [9]. Likewise, Göbel et al. noted that the fork rate could potentially be used to detect selfish mining [11]. Decker and Wattenhofer did extensive research into the way blockchains communicate and observed that detecting all forks is easier said than done [33]. Forks

create a natural rift in the network where only the nodes on the edge of this rift know about the fork. To detect all network forks, one would need to connect to all the nodes on the network. Decker and Wattenhofer also observed that the fork rate would increase as the network grew and the size of the blocks increased. Most research that references the fork rate uses Decker and Wattenhofer's 1.69% fork rate, but this has a potential problem. The problem lies in the fact that the fork rate is an average over 70 days of data. Some days had higher spikes in the number of forks and some days had hardly any. A high number of forks in a single day is expected and not necessarily an indication that selfish mining is taking place. The average fork rate will not be properly calculated until well after the attack has taken place.

Implementation of this feature shares the same difficulty as the fork height. It is difficult to measure this feature and no known historical records exist.

4.1.3 Timings between blocks

Block announcements in the Bitcoin blockchain are spread using a gossip protocol in which each node tells its neighbors about the new block it just heard about [35]. The Bitcoin consensus algorithm keeps block production at the rate of one block every 10 minutes which it does by adjusting the mining difficulty every 2016 blocks. The discovery of new blocks is Poisson distributed which means that the time it takes to discover a new block is independent of when the last block was found. A miner is just as likely to discover a new block at time=0 as they are at time=20 minutes [33]. Although the act of finding a new block is an independent variable, the average time it takes is kept at 10 minutes per block. This exponential distribution makes it difficult to tell if two blocks released back-to-back were selfishly mined or just part of the normal mining

process. This problem is complicated by the fact that the block timestamp is not always correct since the nodes do not synchronize their clocks. A block with a height of $h + 1$ can have an earlier timestamp than the earlier block with a height of h.

Many PoW blockchains implement slightly different difficulty adjustments but the underlying principle stays the same. The Ethereum blockchain mines a block every 13 seconds on average and adjusts the difficulty every block instead of waiting for a set number of blocks.

Blockchain timestamps are stored in a UNIX format, and we measured this feature by taking the timestamp of a block and subtracting the previous timestamp. While no previous literature has suggested that the block time could indicate an attack, we decided to capture it and how much of a role it plays. We hypothesize that selfish mining could briefly shift the Poisson distribution due to the extra hashing power held by the attacker.

4.1.4 Miner Revenue-per-Hour

As stated in Chapter 2, miners earn cryptocurrency for finding a new block and including transactions in a block. As miners conduct mining operations, they can calculate how much money they expect to earn over a given period. Göbel et al. calculated the revenue-per-hour as (block rewards/percentage of the network hashing power). As selfish mining increases and the attackers earn more than their fair share of the block rewards, the honest nodes' revenue-per-hour will drop. As implied by the name, using the revenue per hour is a slow detection process because it works on the hourly average. There is an added difficulty to this since the network hashing power is calculated from the current difficulty [36], [37]. The total, real network mining power can

fluctuate but the difficulty itself is only adjusted once every two weeks. This makes it difficult to know how much mining power there is on a network at any given point.

Although it is possible to measure this feature, we chose not to include this feature as the detection interval of one hour was outside of the time frame we wanted to examine.

4.1.5 Transaction Count/Block Size

To the best of our knowledge, the transaction count has not been studied by anyone as it relates to selfish mining. Decker and Wattenhofer measured the effects of block size on propagation speed and noticed a strong correlation between the two [33]. Smaller blocks propagate through the network faster than larger ones. The selfish miner's goal is to propagate their blocks as fast as they can to ensure that most of the nodes build on top of theirs. Honest nodes, on the other hand, want to include as many transactions as possible to gain more money from transaction fees. This could result in a large disparity between the transaction counts in honestly mined blocks and the selfishly mined blocks.

An added difficulty for selfish miners is that transactions are broadcast to all nodes and stored in the transaction pool but would be based on the honest miner's blockchain. Unless the selfish miners and honest miners chose to include the same transactions, selfish miners could spend too much time trying to find valid transactions that do not depend on previous transactions that were included in the honest miner's chain and not the selfish one.

4.1.6 Receiving Wallet Address/Coinbase Wallet Address

A launched selfish mining attack could double-spend cryptocurrency if the miners so choose. When the attackers double-spend money, they will send it back to a wallet

address that they control. While it is best practice to create a new wallet address for every new transaction, it is not required. A selfish miner could continue to selfishly mine and send transactions to the same wallet address. Reusing a wallet address is not unique to selfish mining but it could help identify an attack when combined with other factors.

The coinbase transaction sends the reward for mining a new block to the miner-specified wallet address. This is typically the address that identifies a miner or mining pool. Regardless of if the attackers double-spend money, they want to receive the financial reward for mining a new block. Just like in the other transactions, the miner can supply a new wallet address for every new coinbase transaction. However, in some observed selfish mining attacks, the attacker repeatedly sent the coinbase transaction to the same wallet address [38]. Reuse of the coinbase wallet address could help identify selfish mining when combined with other factors.

The difficulty in implementing these features is that they could lead to an artificially high detection rate due to our selfish mining algorithm thinking that a specific address always indicates an attack. While this may be true, it does not lead to the discovery of new attacks. A possible way to implement this feature would be to keep track of all known addresses and just flag an address as new or previously known. As this would require a database of all previously used addresses, we decided not to implement this feature.

4.1.7 Current Difficulty

An increasing mining difficulty means that more miners are working to solve the PoW algorithm in hope of earning the block rewards. If the price of Bitcoin falls, it becomes financially unfeasible for some miners to continue operating. This results in

some miners going offline leaving their computing power dormant. This change in mining power will result in a drop in the mining difficulty and this lower mining difficulty represents an opportunity for selfish miners to use their current mining power more effectively or buy up the dormant mining power. The correlation between the mining difficulty and selfish mining is currently unresearched and there may be ranges of mining difficulties that present better opportunities for selfish miners.

4.1.8 Difficulty Delta

A late addition to our feature set was the difficulty delta which tracks the changes in difficulty between blocks. A positive number means that the difficulty increased between blocks while a negative number means that the difficulty decreased. We hypothesized that selfish mining would result in an increase in difficulty due to more hash power entering the system.

4.1.9 Hash Price

Blockchains have given rise to several new businesses, one of which is the rental of hash power. When it first started, mining Bitcoin was as simple as downloading the client software and running it on a desktop computer. The total mining power of the network was low enough that consumer CPUs could mine competitively. As the network grew, mining companies sprung up whose sole purpose was to mine Bitcoin. They developed application-specific integrated circuits (ASIC) designed specifically to run the Bitcoin hashing algorithm. The hash power arms race led to miners banding together into pools that share the new block rewards according to how much hashing power the individual contributed. Furthermore, businesses sprang up that allow people to sell their hashing power instead of using themselves which allows anyone to buy hashing power

online without the overhead of purchasing hardware. The company NiceHash is one such online retailer of hashing power [39]. This online marketplace also lowers the entry cost for attackers who wish to launch a selfish mining attack. NiceHash provides pricing history for most of the popular mining algorithms beginning in July of 2019 and attackers may have used NiceHash to purchase hashing power. A spike in the price of a mining algorithm may indicate an attack.

4.1.10 Hash Price Delta

Another late addition to our feature set was the hash price delta. Like the difficulty delta, it captured the changed in the hash price as opposed to the actual price. We hypothesized that a malicious miner could launch an attack using rented hash power and so the hash price would increase when an attack was launched.

Table 3. Feature selection summary.

Feature	Significance
Fork height	The difference in block height between chains in a fork may indicate an attack
Fork rate	A higher rate of forks may indicate selfish mining
Timings between blocks	Blocks mined by selfish miners may be closer together than the normal blocks
Miner revenue per hour	A decrease in a miner's revenue per hour may indicate an attack
Transaction count / Block size	The number of transactions included in a block/size of the block may indicate an attack. Smaller blocks propagate more quickly so they are more likely to be selfish
Receiving address / Coinbase address	The receiving wallet address of the transactions in the block may indicate an attack
Current difficulty	The current difficulty may reveal optimal times to engage in selfish mining
Difficulty delta	A change in the difficulty may suggest that a group of miners started mining selfishly
Hash price	A higher hash price may indicate an attack due to the attackers renting hashing power
Hash price delta	Spikes in the hash price could mean the attacker rented a large amount of hashing power

4.2 Research Objectives

This research consisted of two major phases. The first phase was the analysis and selection of factors and the gathering of data. We evaluated several data sources before settling on the GetBlock API.

For the second phase, we worked on choosing the most appropriate machine-learning algorithm for our problem. After the selection of an algorithm, we fed the newly gathered data into it to find the significant factors. We then verified the classifier against test data that was benign or selfish. At this point, we will identify which factors as the most important for identifying an attack.

CHAPTER 5

METHODOLOGY

As we were unsure what factors would prove significant to detecting a selfish mining attack, we gathered data from known attacks and fed it into a machine learning algorithm and let it choose which factors were significant.

We briefly looked at simulating selfish mining attacks to generate data on real-time attacks [40]. We modified the SimBlock blockchain simulator by adding the ability to model selfish mining attacks. SimBlock is written in Java and is easily extensible due to its open-source nature [32]. Ultimately, using a simulator is not as accurate as using real-world blockchain data, and since we did not have data to verify the simulated attack against, we chose not to go this route.

In the following sections, we will discuss our methodology which contains the data selection process, the transformation of that data into a format usable by our machine learning algorithm, and our selection of a machine learning algorithm.

5.1 Data Source Selection

Gathering selfish mining data is a difficult proposition as there are a limited number of known attacks. Most attacks are poorly documented without any concrete indication of which blocks belong to the attacker and which blocks are benign. For our

purposes, we came up with four criteria for selecting blockchain data and then examined five known selfish mining attacks. We evaluated each cryptocurrency with the following criteria:

1. The cryptocurrency must use a PoW algorithm. Our research concentrates on PoW and other mining algorithms may have different significant factors.
2. Well-documented attacks. We need a verified source of known selfish blocks to train our machine-learning algorithm. We do not want to train it with incorrectly labeled blocks.
3. Public API for gathering block data. We need a way to download the data and transform it for use in our machine-learning algorithm.
4. Hash price history on NiceHash for the same time frame as the attack. NiceHash started publishing hash price history in July 2019. The blockchain needs to contain data from an attack that happened after July 2019.

The first blockchain we examined was Bitcoin Gold. Bitcoin Gold (BTG) was attacked from 05/16/2018-05/19/2018 at block heights 528651-529048 [41]. The attacker blocks were not contiguous but were mixed with ranges of benign blocks. In total, the attacker mined roughly 114 blocks that were included in the final blockchain.

It appears that the attacker mined blocks until the difficulty rose significantly and then released the selfish chain [42]. As a result of the attack, the difficulty would rise for the honest miners who would then have to spend more CPU cycles mining harder blocks. Since the BTG difficulty adjustment happens every 30 blocks, the attacker was able to attack again as soon as the difficulty fell to an acceptable range [43].

BTG looked like a good candidate for harvesting data as it is a PoW blockchain, has a well-documented attack, and has a public API for gathering data. Unfortunately, the documented attack happened a year before NiceHash started publishing the price history and we decided not to use this data. To further limit the usability of the BTG data, the developers changed their PoW algorithm two months later in July 2019 to reduce the risk of attack [44].

The second PoW cryptocurrency we examined was Verge. Instead of using a single PoW algorithm, Verge uses five different algorithms to mine blocks. Attackers replaced around 12,000 blocks during an attack in 2018 and another 560,000 during an attack in 2021 [45], [46]. This attack was unique because it only targeted a single PoW algorithm out of the five that Verge uses. Verge round-robins the PoW algorithm and no two consecutive blocks are allowed to have identical PoW algorithms. The attackers controlled a large amount of Scrypt hashing power and exploited a timestamp bug to fool the network into thinking that attack blocks were mined in the past. By shifting the timestamp on newly mined blocks, the attacker would make the network think these blocks were mined in the past and that the next block was allowed to use the Scrypt PoW algorithm multiple times in a row [45]. Although hash price information is available for the 2021 attack, the unique nature of this attack made it very specific to the Verge cryptocurrency and not generalizable to other blockchains. Ultimately, we decided not to use this data.

ZenCash (now called Horizon) is a privacy-focused cryptocurrency and was attacked in June 2018 with the attacker stealing more than $500,000 [47]. This attack has

very poor documentation and does not meet our requirements, so we did not consider this data for use in our research.

The fourth cryptocurrency we examined was Bitcoin Satoshi's Vision (BSV). BSV is a fork of Bitcoin Cash which, in turn, was a fork of Bitcoin. BSV was attacked in August of 2021 with the attackers reversing at least 100 blocks [48]. While BSV has the desirable characteristics of being the same PoW algorithm as Bitcoin and having an attack that takes place in the same timeframe as our hash price data, it is poorly documented. To accurately train our machine learning algorithm, we need classified data as selfish and benign from the same blockchain. Since we could not find any authoritative sources that categorized the blocks, we ultimately decided not to use BSV.

The last cryptocurrency we examined is Ethereum Classic which is a fork of the Ethereum blockchain. ETC split from Ethereum in 2016 due to a disagreement about how to handle the fallout of a hack that stole funds from a smart contract that was set up to fund the development of Ethereum. ETC uses a variant of the Dagger-Hashimoto PoW algorithm and was the victim of a selfish mining attack once in 2019 and twice in 2020 [8], [14], [15]. The 2019 attack saw 928 selfishly mined blocks added to the blockchain with the 2020 attacks adding another 3,615 in the first and 4,236 in the second. Due to a bug in their software, the company Bitquery was able to capture the contents of the ETC blockchain before and after the attack. Normally the chain reorganization would have deleted the benign blocks, but the bug allowed them to preserve the original contents. This allowed them to positively identify which blocks were the attacker blocks and which ones were benign. ETC satisfies all our requirements for selecting blockchain data to train our machine learning.

1. ETC is a PoW blockchain that used the Dagger-Hashimoto algorithm at the time of the attacks.
2. ETC has a well-documented attack and even preserved the blocks that the attack orphaned.
3. ETC has several public APIs for gathering block data.
4. The 2020 attacks occurred during a period that we have the hash price history for.

5.2 Data Gathering

After selecting a blockchain, the next step was to gather the selfish and benign blocks to use in training and validating our machine learning algorithm. Blockchains store block data in binary format and are often quite large and prohibitive to download. The ETC blockchain that we selected for our analysis is over 14GB and it required custom software to search for blocks. To make blockchain analysis easier, developers host blockchain explorers that allow anyone with a web browser to access blockchain data. The ETC block explorer allowed us to spot-check blocks for initial analysis, but we still required a way to download blocks and format them for use in machine learning.

We evaluated downloading the entire ETC blockchain or using a public API service to download individual blocks. We ended up finding a company called GetBlock that provides a public API that allows users to access nodes connected to various blockchains [49]. We set up an account with GetBlock and obtained an API key which allowed us to make API calls to download individual ETC blocks without needing to download the entire blockchain.

To download this data, we wrote a program called Block Scraper to download blocks and transform them into a JSON-formatted data file. The program is written in C# and allows the user to select a blockchain and define which blocks to download by inputting the block heights. Block Scraper has an internal list of the block heights that were selfishly mined and automatically classifies the blocks as selfish or benign. A screenshot of the program is shown in Figure 5 and the program is available for download on our GitHub page [50].

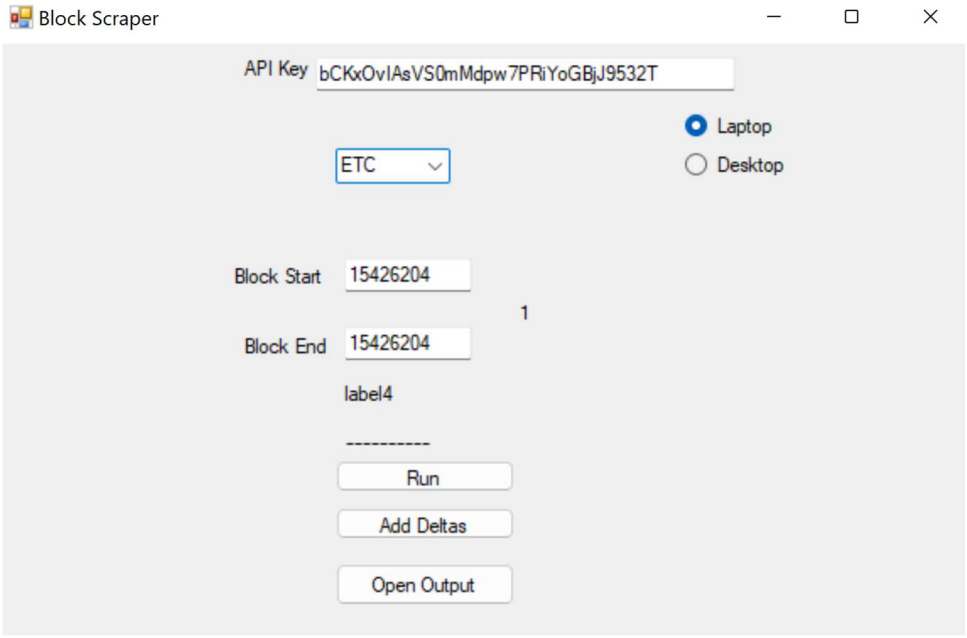

Figure 5. Screenshot of the block scraper program. The user defines block heights to download which are automatically labeled via the internal list of selfish block ranges. Output is a JSON file.

One piece of data that is unavailable from GetBlock is the hash power price history. As previously mentioned, the company NiceHash sells hash power for many different mining algorithms. While NiceHash has been in operation since 2014, it only

started publishing price history in 2019 and hosts a public API that allows a user to download the price history for a selected mining algorithm [51]. The hash price data is a JSON file that has price measurements every 15 minutes in the format of timestamp (Unix formatted), speed (terahashes per second), and price (Satoshi's per hash per second). After we download each block, we match the timestamp of the block to the closest hash price timestamp and use that value to populate the hash price at the time the block was mined. Figures 6 and 7 show sample data from these data sources.


```
[
   1562913900,
   168674245389.6009,
   0.000159474398433901
],
[
   1562914800,
   175022500689.58502,
   0.00015961565659483
],
```


Figure 6. Example hash price data from NiceHash. Each JSON array represents a 15-minute reading of the hash price. The three values in order are, the UNIX timestamp, hash speed, and cost in Satoshi's per hash, per second.

```
{
   "isSelfish": 1,
   "height": 10939858,
   "size": 542,
   "timeDelta": 13,
   "target": "66303180309088",
   "txCount": 0,
   "difficulty": "66303180309088",
   "difficultyDelta": "0",
   "hashPrice": "0.00042832392583834900",
   "hashPriceDelta": "0.00000000000000000000"
},
{
   "isSelfish": 0,
   "height": 10939859,
   "size": 542,
   "timeDelta": 13,
   "target": "66303180309088",
   "txCount": 0,
   "difficulty": "66303180309088",
   "difficultyDelta": "0",
   "hashPrice": "0.00042832392583834900",
   "hashPriceDelta": "0.00000000000000000000"
},
```


Figure 7. Sample GetBlock data. In some instances, values were transformed from hexadecimal before being output to JSON. Hash price was added in from NiceHash data.

After creating our block scraper, we downloaded selfish and benign blocks from the three different ETC attacks. We named our datasets ETC_2019_Attack.json, ETC_2020_A_Attack.json, and ETC_2020_B_Attack.json. The contents of these datasets are summarized in Table 4. We started by gathering blocks from a large time frame around the attacks. These initial datasets were unbalanced and contained more benign rows than selfish rows. We then created three additional, balanced datasets to train and test our model.

Our first dataset is from the 2019 attack for which we downloaded blocks 7247400-7261800 for a total of 14,401 blocks of which 920 were selfish and 13,481 were

benign. We sampled the 2019 dataset to create a balanced dataset that contained 920 selfish rows and 920 benign rows.

For the first 2020 attack, we gathered blocks 10901000-10908682 for a total of 7,683 blocks of which 3,615 were selfish and 4,069 were benign. When then sampled this data to create a balanced dataset of 3,615 selfish rows and 3,615 benign rows for a total of 7,230 rows.

Our last dataset was the second 2020 attack which contained 4 rows of which 23 were selfish and 12 were benign. As we did with the others, we sampled this data to create a balanced dataset of 23 selfish and 12 benign for a total of 12 blocks.

Table 4. Summary of block ranges downloaded for each dataset for use in machine learning.

Label	Block ranges	Selfish blocks	Benign blocks	Total blocks
2019 Attack	7247400-7261800	920	13,481	14,401
2019 Balanced	Sampled subset of 2019	920	920	1,840
2020 Attack A	10901000-10908682	3,615	4,069	7,683
2020 A Balanced	Sampled subset of 2020A	3,615	3,615	7,230
2020 Attack B	10933000-10943000	4,236	5,765	10,001
2020 B Balanced	Sampled subset of 2020B	4,236	4,236	8,472

5.3 Data Figures

Figures 8-16 show the shape of the data we gathered from each period. Due to formatting issues, we keep the blocks in order by block height but display the data using the index of the block in the list. Since we sampled the block in the attack period instead

of taking all the contiguous blocks, it was causing display issues in the graphs. Each figure shows a feature we identified in Chapter 4 plotted to the x-axis which is the block index and the red marks on the line identify the selfish blocks.

5.3.1 2019 Data Figures

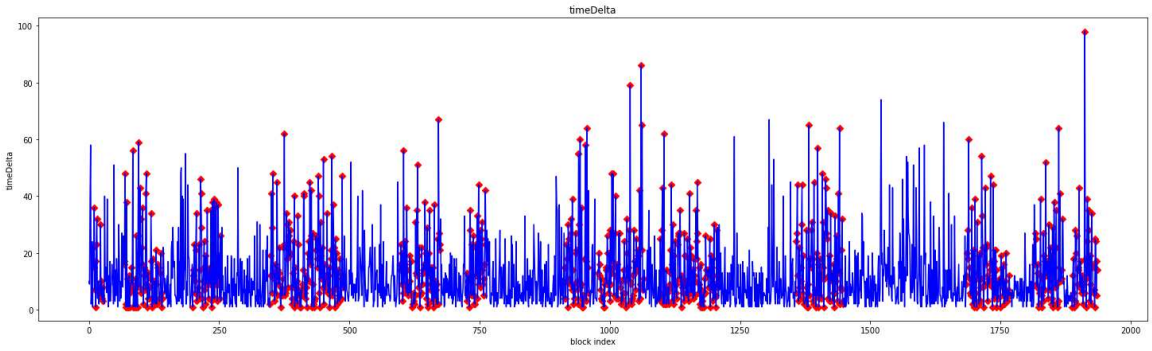

Figure 8. 2019 attack time deltas. This graph shows the time difference in seconds between blocks during the 2019 attack. Selfishly-mined blocks are shown in red. Block heights are non-contiguous but deltas are calculated off the previous contiguous block.

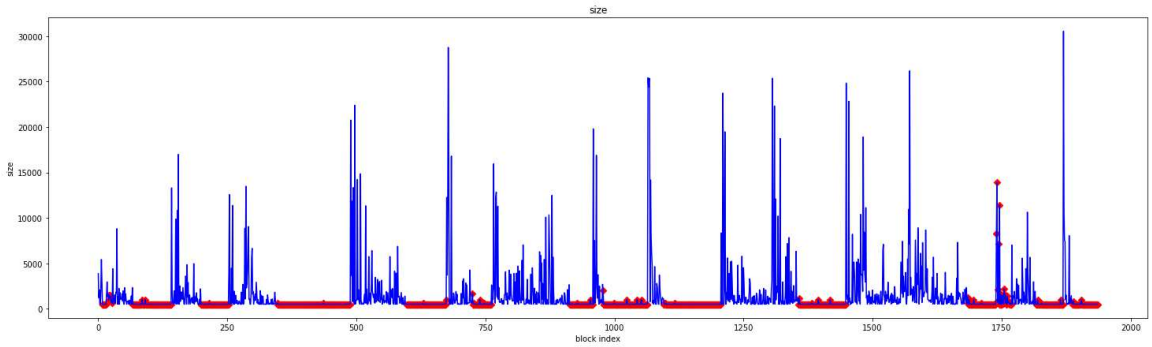

Figure 9. 2019 attack block size. Block size is measured in bytes and red marks indicate an attack block.

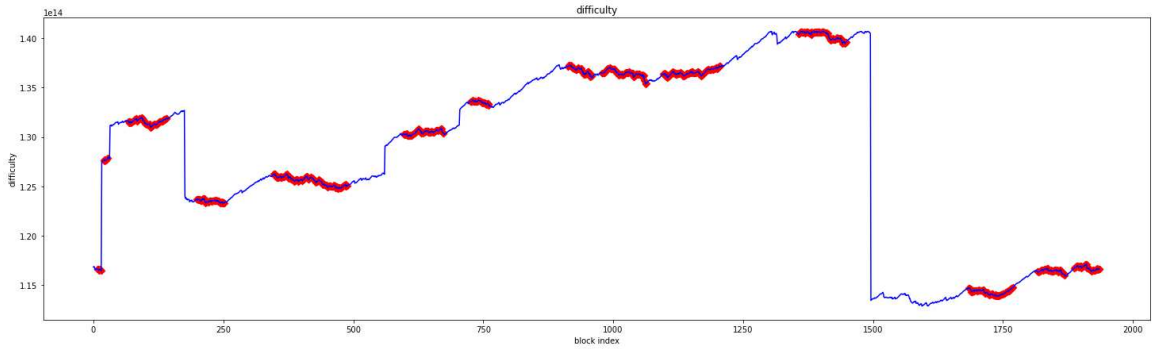

Figure 10. 2019 attack difficulty. Value reduced to decimal for readability. Significant spikes and dips show where the block sampling took place and do not show a sudden dip in difficulty. Red marks indicate attack blocks.

5.3.2 2020A Data Figures

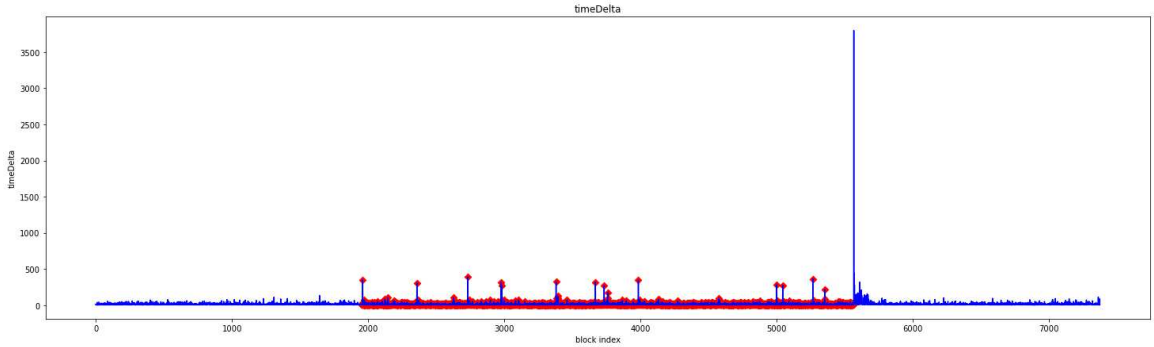

Figure 11. 2020A time delta. This graph shows the time difference in seconds between blocks during the 2020A attack. Selfishly mined blocks are shown in red. Block heights are non-contiguous but deltas are calculated off the previous contiguous block.

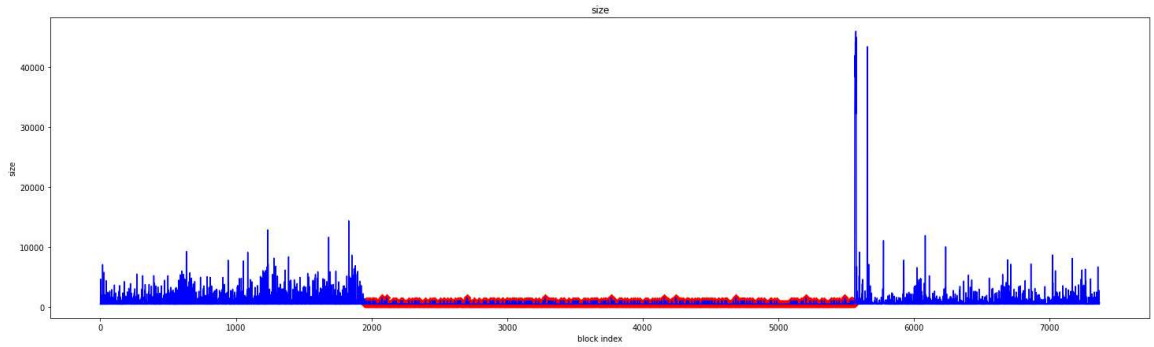

Figure 12. 2020A block size. Block size is measured in bytes and red marks indicate an attack block.

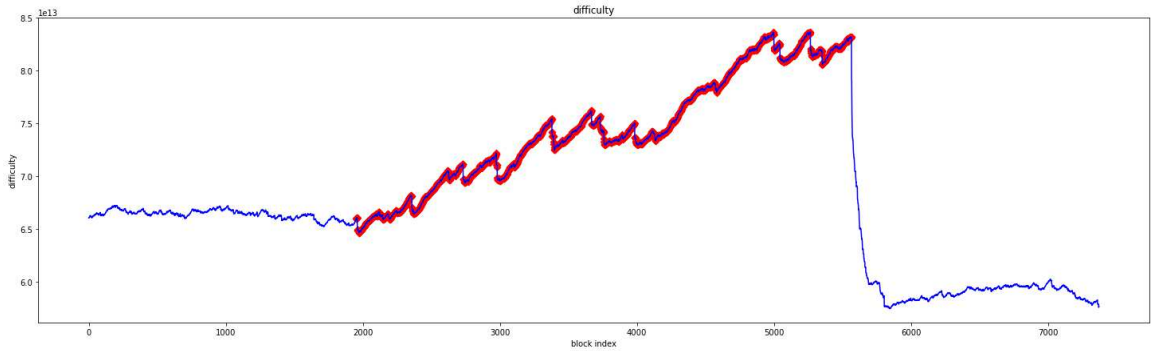

Figure 13. 2020A difficulty. Difficulty reduced to decimal for readability and red marks indicate attack blocks.

5.3.3 2020B Data Figures

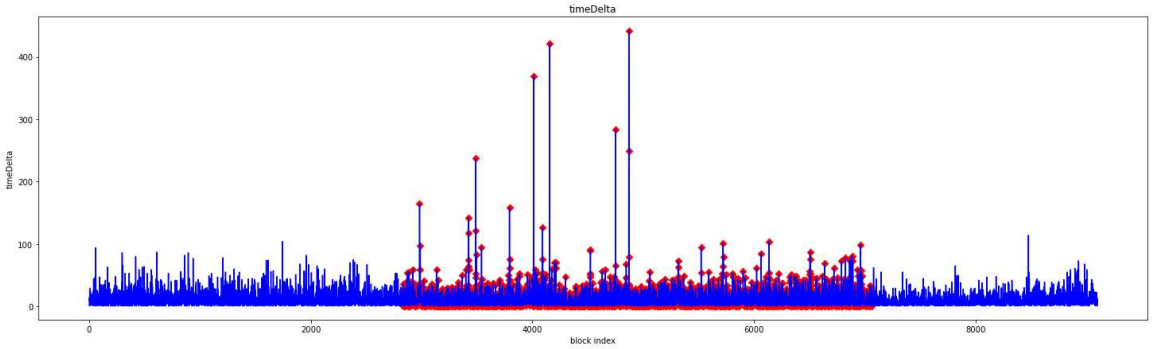

Figure 14. 2020B time delta. Time deltas between blocks are shown in seconds. Red marks indicate an attack block.

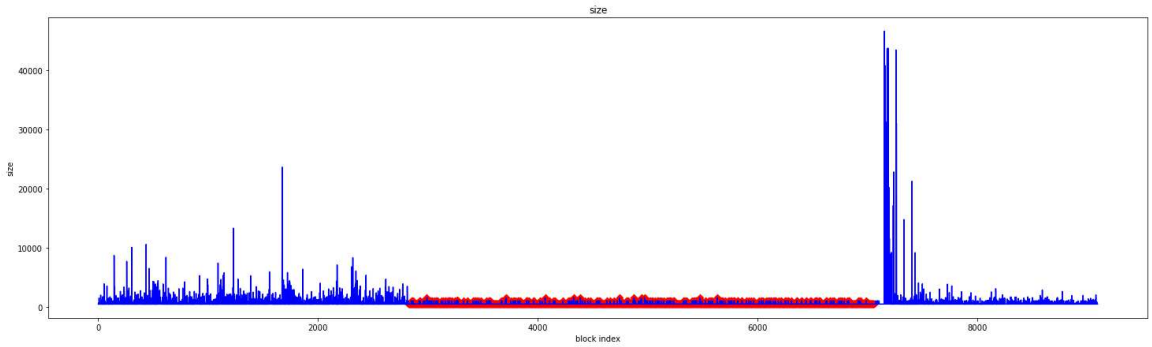

Figure 15. 2020B block size. Block sizes are shown in bytes and red marks indicate an attack block.

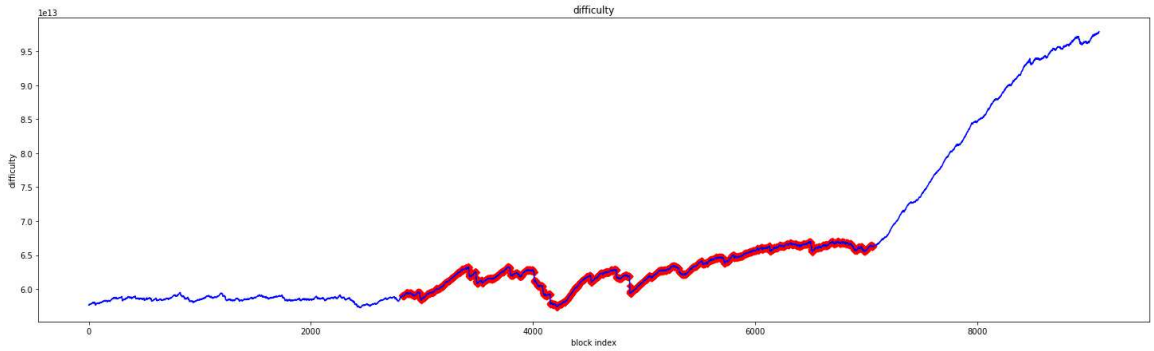

Figure 16. 2020B difficulty. Difficulty reduced to decimal for readability and red marks indicate attack blocks.

5.4 Machine Learning

The problem of finding selfish mining attacks is a classic classification problem and as we are unsure which feature will prove significant, we selected a Random Forest machine learning algorithm. We used Anaconda version 2.3.0 to download and install the RandomForestClassifier package from scikit learn version 1.0.2 [52]. We used Spyder version 5.2.2 to code our machine learning classifier and the default setting of 100 estimators for training our model. We made predictions using the test data and then outputted the accuracy, recall, and precision metrics. The following paragraphs detail the metrics and they are summarized in Table 5.

Table 5. Summary of machine learning metrics.

Metric	Description
Accuracy	The ability of the classifier to correctly label the block as selfish or benign
Precision	How accurate a selfish prediction is
Recall	The ability of the classifier to find all the selfish blocks
F1 Score	How accurate the classifier is at finding selfish blocks and correctly labeling selfish blocks

The accuracy metric is simply the number of correct predictions made by the classifier divided by the total number of predictions. This gives a measure of how accurate the classifier is but requires a balanced test dataset to give accurate results. An unbalanced dataset may correctly classify all the benign samples but miss all the selfish and still have an 80% accuracy rating.

The precision metric assigns a score to the proportion of positive identifications that were correct. A precision score of 0.80 would mean that 80% of the blocks that the classifier predicted as selfish were selfish.

The recall metric assigns a score to the proportion of true positives that were accurately identified. A recall score of 0.75 means that the classifier found 75% of the selfish blocks and missed 25%.

The F1 score combines precision and recall into a single score. This score measures both how many of the selfish blocks the classifier found and how accurate a selfish prediction was.

We measured the importance of features by using the Mean Decrease in Impurity (MDI), also known as the Gini importance. The MDI calculates feature importance by how many times this feature is used in a split in the decision tree. The higher the MDI

score for a feature, the more that feature is used to classify a sample across all the decision trees in the random forest [53]. The black bars on the MDI graphs are error bars that represent the feature's inner-tree variability [54].

CHAPTER 6

TRAINING APPROACHES AND RESULTS

We split our balanced datasets into a 70/30 division of training and test data. Our initial feature set in each dataset was the time delta between blocks, block size, current difficulty, and hash price for the datasets where this feature was available.

6.1 2019 Training and Prediction

We started with training on the 2019 training data and then made predictions on the 2019 test data. The three features used were block size, time delta, and difficulty. Hash price was not used as it was unavailable for the 2019 dataset. The results showed our classifier had an accuracy of 85.6% with the block size being the most important factor in detecting selfish mining attacks. The precision and recall for both benign and selfish data we about the same at roughly 85%. Figure 17 shows the feature importance and Tables 6 and 7 show the classifier metrics.

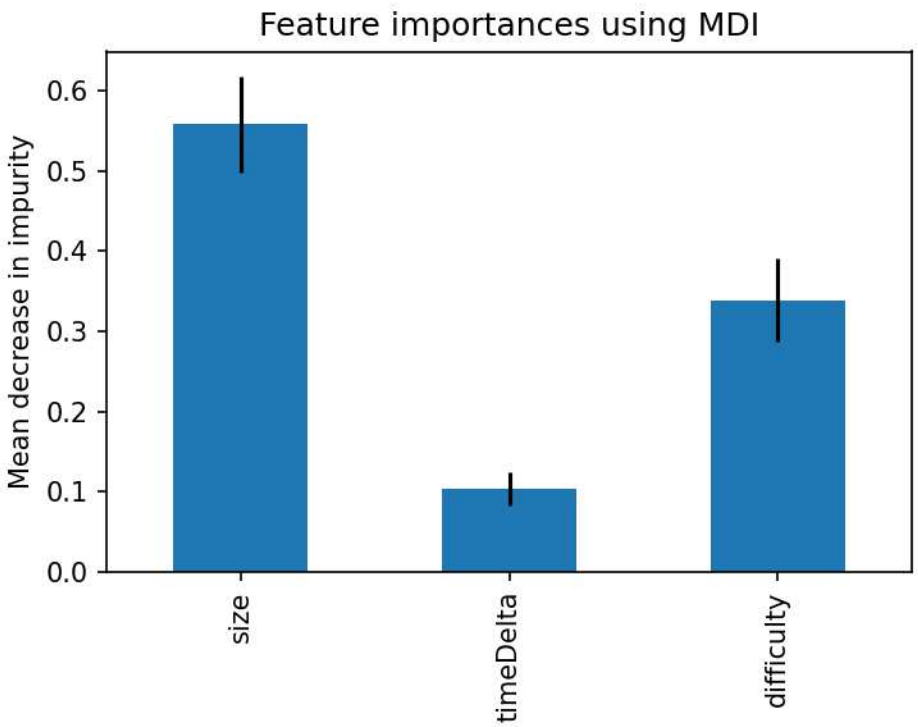

Figure 17. 2019 feature importance measured by mean decrease in impurity.

Table 6. 2019 classifier accuracy, precision, and recall.

Accuracy	Precision	Recall
0.856	0.852	0.878

Table 7. 2019 classifier metrics by block classification.

Label	Precision	Recall	F1
Benign	0.86	0.83	0.85
Selfish	0.85	0.88	0.87

6.2 2020A and 2020B Training and Prediction

We repeated this training on the 2020A and 2020B datasets but included the hash price as it was available for these two datasets. Our features included the block size, difficulty, hash price, and time delta. For each dataset, we split it into 70% training data and 30% test data.

The results were very good, and the accuracy of each prediction hovered around the 99% range. In the 2020A dataset, the hash price was chosen as the most important factor followed by the difficulty and in the 2020B dataset, the difficulty was chosen as the most important factor followed by the hash price. Both results were different from the block size favored by the 2019 dataset. Figure 18 shows the 2020A feature importance by MDI score and Figure 19 shows the 2020B feature importance while Tables 8-11 show the classifier metrics. The precision and recall for both these datasets sat close to 100% which was suspicious. We discuss this further in section 6.5.

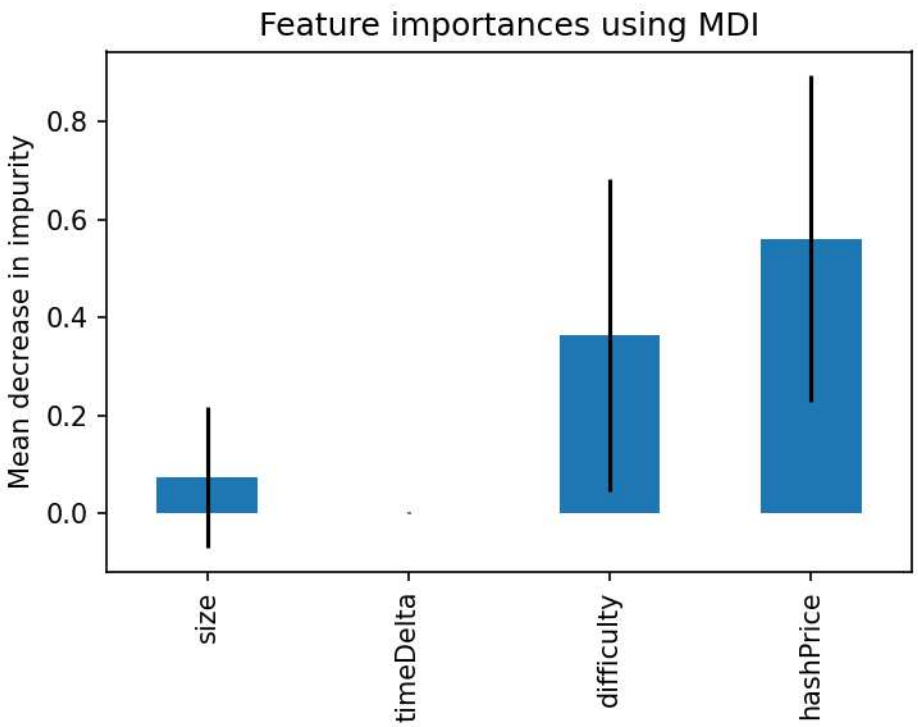

Figure 18. 2020A feature importance measured by mean decrease in impurity.

Table 8. 2020A classifier accuracy, precision, and recall.

Accuracy	Precision	Recall
0.997	1.0	0.995

Table 9. 2020A classifier metrics by block classification.

Label	Precision	Recall	F1
Benign	1.0	1.0	1.0
Selfish	1.0	1.0	1.0

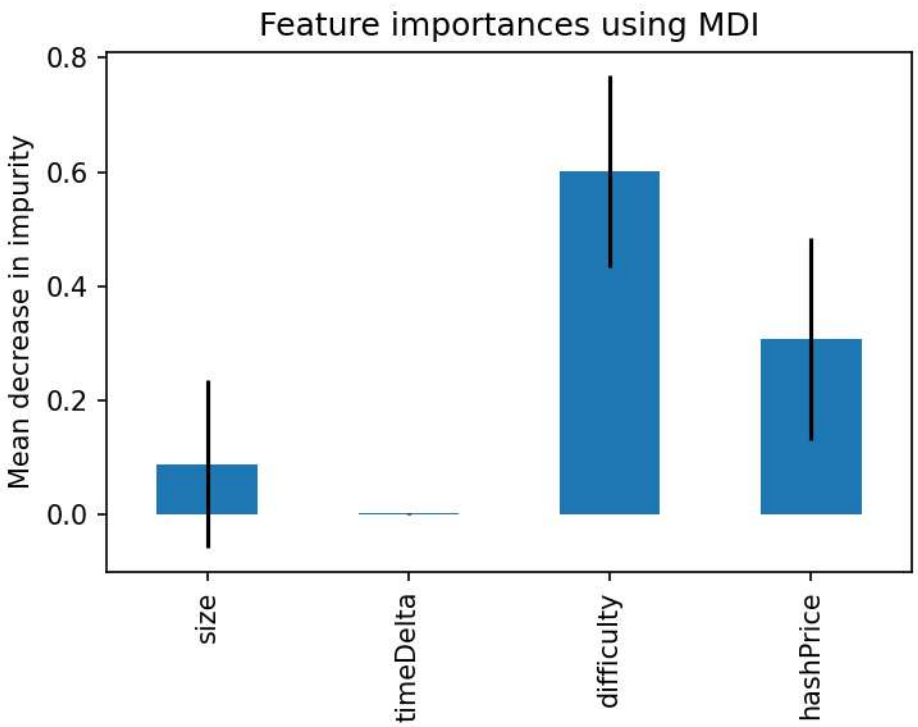

Figure 19. 2020B feature importance measured by mean decrease in impurity.

Table 10. 2020B classifier accuracy, precision, and recall.

Accuracy	Precision	Recall
0.999	1.0	0.999

Table 11. 2020B classifier metrics by block classification.

Label	Precision	Recall	F1
Benign	1.0	1.0	1.0
Selfish	1.0	1.0	1.0

6.3 2019 Training and 2020A and 2020B Predictions

Next, we trained the classifier on the 2019 attack and used it to predict the 2020A and 2020B attacks. Although the 2020A and 2020B attacks contained the hash price, we did not include it since we trained the classifier with the 2019 dataset which did not have it. We used the block size, time delta, and difficulty in our training. Unfortunately, we did not achieve the results we were looking for as the accuracy hovered around 50% for predicting both datasets. The block size was once again the most important factor, but the prediction was no more accurate than a coin flip. The MDI chart for the 2020A prediction and the 2020B prediction is shown in Figure 20. Tables 12 and 13 show the classifier metrics for the 2020A prediction while Tables 14 and 15 show the metrics for 2020B.

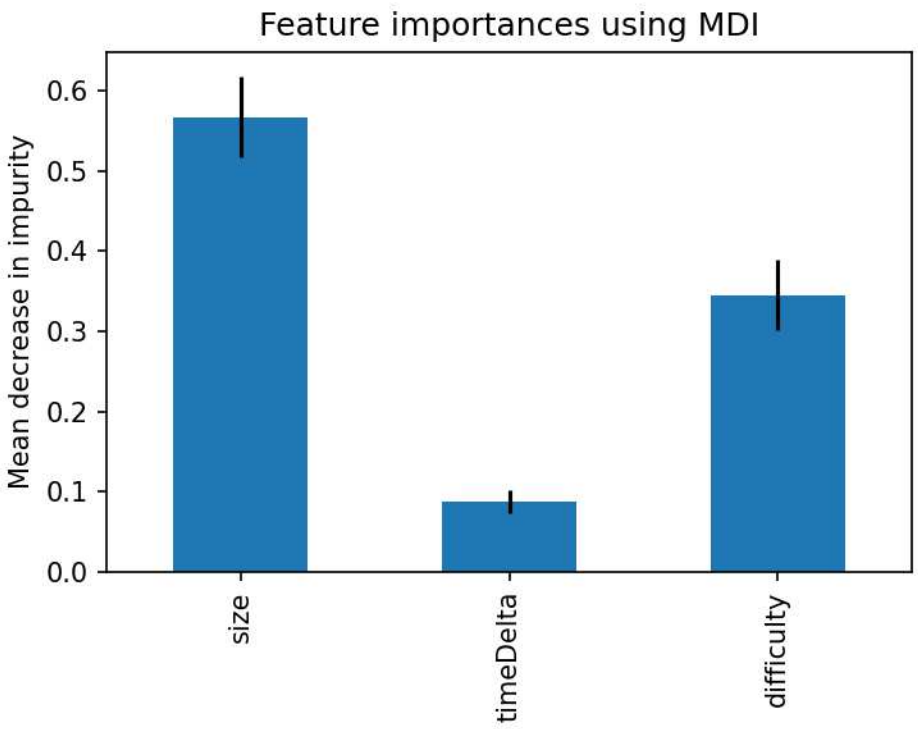

Figure 20. 2020A and 2020B feature importance measured by MDI for original training data. The classifier was trained on the 2019 dataset and the MDI chart was the same for the 2020A prediction and 2020B prediction.

Table 12. 2020A classifier accuracy, precision, and recall for original training data.

Accuracy	Precision	Recall
0.537	0.564	0.330

Table 13. 2020A classifier metrics by block classification for original training data.

Label	Precision	Recall	F1
Benign	0.53	0.74	0.62
Selfish	0.56	0.33	0.42

Table 14. 2020B classifier accuracy, precision, and recall for original training data.

Accuracy	Precision	Recall
0.527	0.542	0.359

Table 15. 2020B classifier metrics by block classification for original training data.

Label	Precision	Recall	F1
Benign	0.52	0.70	0.60
Selfish	0.54	0.36	0.43

6.4 2020A Training, 2020B Prediction

We wanted to see if the hash price was a significant factor for predicting an attack, so we trained the classifier on the 2020A attack and used it to predict the 2020B attack. We included the hash price, difficulty, block size, and time delta. Although the hash price was chosen as the most significant factory, the accuracy ended up being less than 50% so it was immediately apparent that we needed to improve the way we formatted the data. Figure 21 shows the MDI score for the features while Tables 16 and 17 show the classifier metrics.

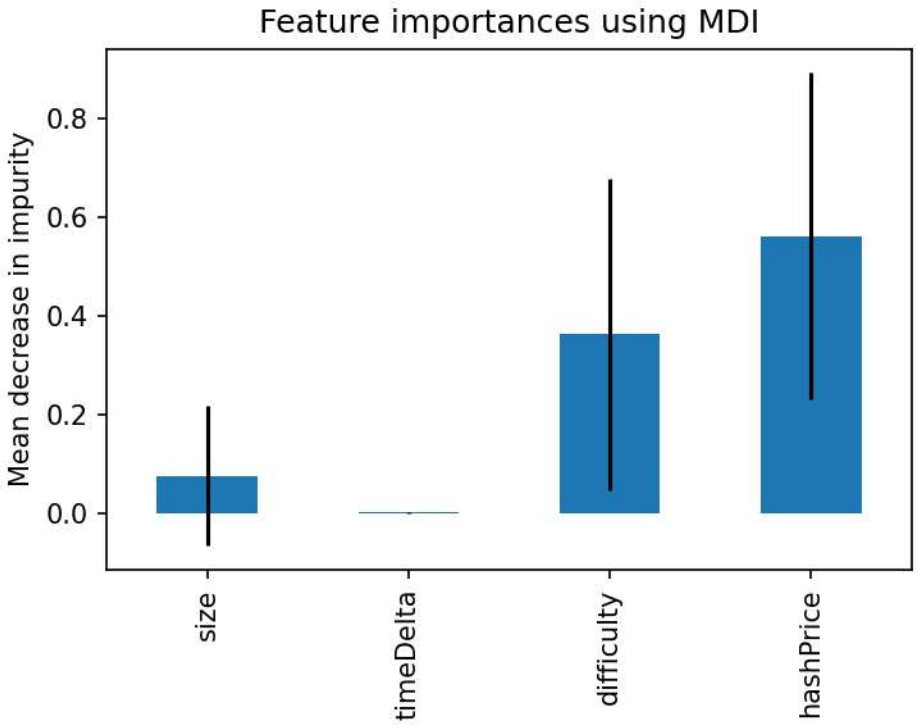

Figure 21. 2020B feature importance measured by MDI for original training data. The classifier was trained on the 2020A dataset and predicted the 2020B.

Table 16. 2020B classifier accuracy, precision, and recall for original training data. Classifier trained on 2020A and predicted 2020B.

Accuracy	Precision	Recall
0.396	0.359	0.266

Table 17. 2020B classifier metrics by block classification for original training data. Classifier trained on 2020A and predicted 2020B.

Label	Precision	Recall	F1
Benign	0.42	0.53	0.47
Selfish	0.36	0.27	0.31

6.5 Discussion

It was apparent that we had a fundamental flaw in our datasets as 50% accuracy was less than desirable. We discovered three things from our initial training.

1. Visual inspection of the data indicated that the hash price showed promise as a significant factor but was not providing accurate results
2. The classifier favored the block size, but block size alone was not sufficient to predict an attack
3. The difficulty was being used incorrectly

One of the problems we discovered is that our difficulty factor was being used incorrectly. The classifier was choosing specific difficulty values as selfish or benign. The difficulty rises and falls naturally as miner join or leave the network, so a specific difficulty value is not an indicator of an attack. What we needed was the change in difficulty as opposed to a specific value.

CHAPTER 7

DIFFICULTY DELTA TRAINING AND RESULTS

To solve the issue of the difficulty, we decided to drop the difficulty and replace it with the delta between two blocks. This allowed us to track how the difficulty changed over time instead of tracking specific difficulty values. Our theory is that significant changes to the difficulty would indicate a selfish miner joined the network and added a large amount of hashing power. We also theorize that this should track with a rise in the rental cost of hash power. Although the hash price is not present for the 2019 attack, the difficulty should rise due to an increase in hash power and provide a latent indicator. The total number of records is the same as each of the original datasets.

7.1 Data Figures

Figures 22-24 plot the new difficulty delta feature for each dataset. A closer examination revealed that significant dips in the difficulty correlate with significant spikes in the time delta. This is to be expected as the difficulty drops when it takes too long to find a block. Red marks indicate a selfishly mined block.

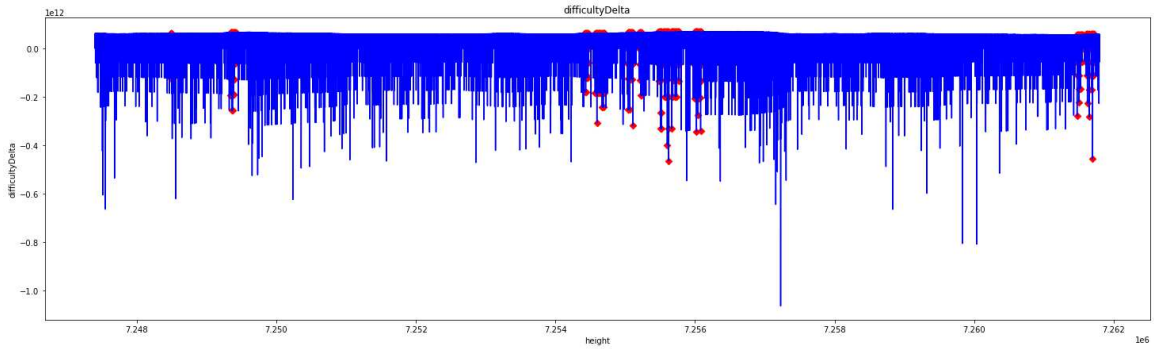

Figure 22. 2019 difficulty delta. Selfishly mined blocks are shown with red marks.

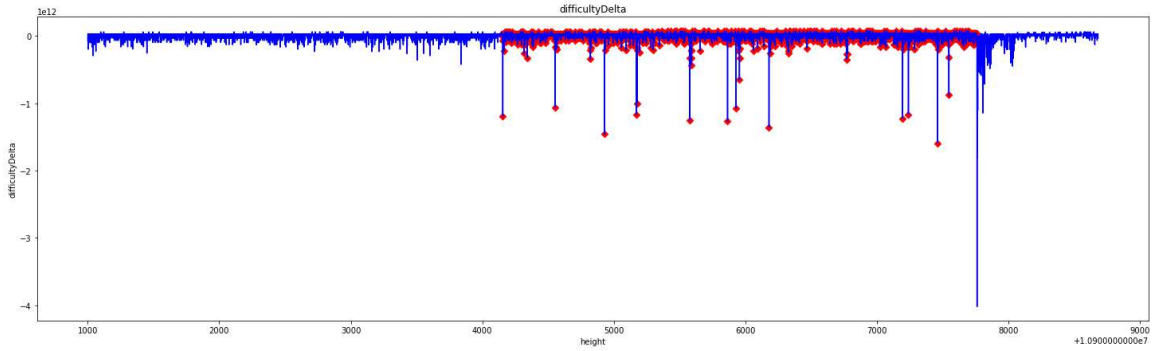

Figure 23. 2020A difficulty delta. Selfishly mined blocks are shown with red marks.

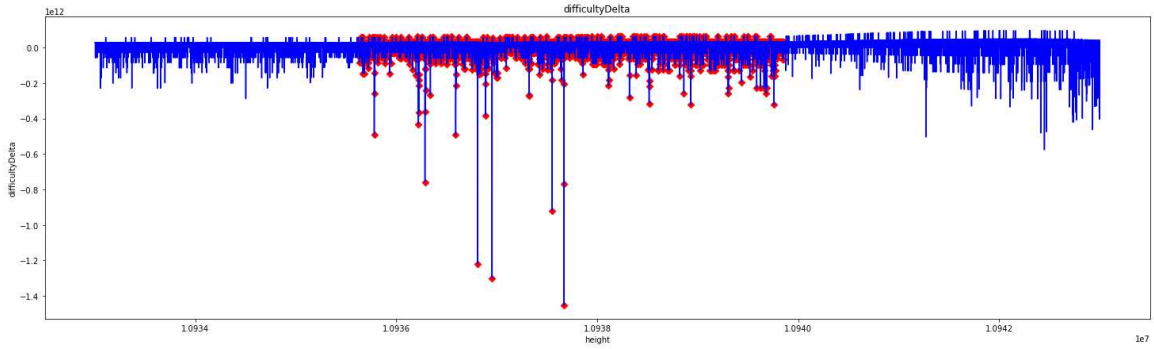

Figure 24. 2020B difficulty delta. Selfishly mined blocks are shown with red marks.

7.2 Training with Difficulty Delta

We swapped the difficulty for the difficulty delta and retrained the classifier on the 2019 dataset and then used it to predict the 2020A and 2020B datasets. Rather than the predicted increase in accuracy, we once again achieved a 50% accuracy rating from the classifier. Figure 25 shows the MDI chart for both the 2020A prediction and the 2020B prediction as they were identical.

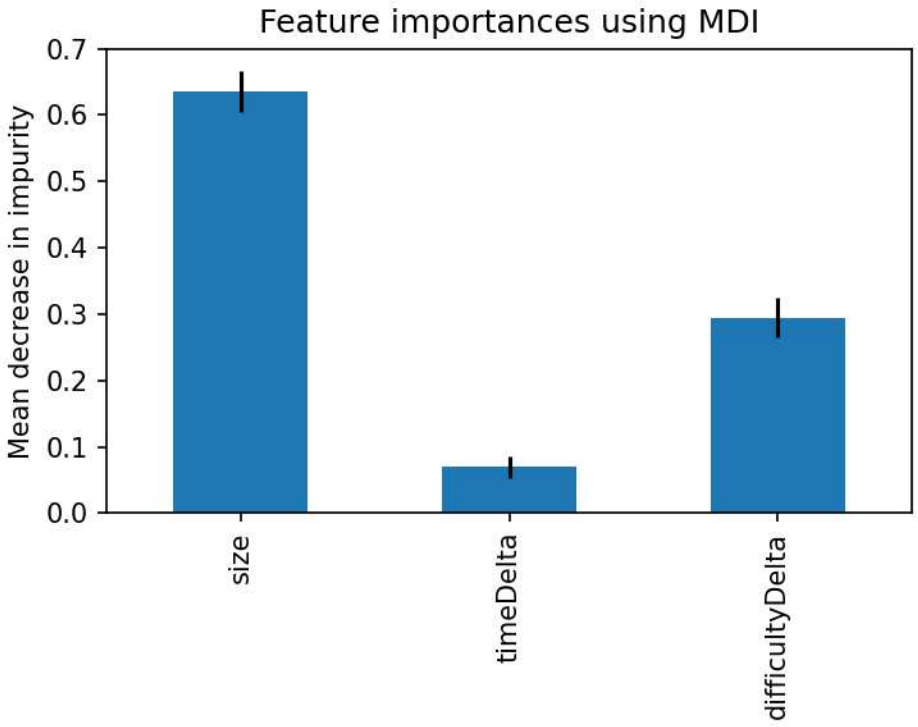

Figure 25. 2020A and 2020B feature importance measured by MDI with difficulty delta. The MDI charts were identical for both predictions.

Table 18.2020A classifier accuracy, precision, and recall for difficulty delta.

Accuracy	Precision	Recall
0.517	0.532	0.290

Table 19. 2020A classifier metrics by block classification for difficulty delta.

Label	Precision	Recall	F1
Benign	0.51	0.74	0.61
Selfish	0.53	0.29	0.38

Table 20. 2020B classifier accuracy, precision, and recall for difficulty delta.

Accuracy	Precision	Recall
0.500	0.501	0.305

Table 21. 2020B classifier metrics by block classification for difficulty delta.

Label	Precision	Recall	F1
Benign	0.50	0.70	0.58
Selfish	0.50	0.31	0.38

7.3 Training with Difficulty Delta and Hash Price

After our experiments with training on the 2019 dataset, we tried training the classifier on the 2020A dataset and predicting the 2020B. Hash price was a very highly rated feature in the previous training, so we wanted to see if the combination of difficulty delta and hash price yielded better results. The hash price once again proved to be an important feature by MDI score and the accuracy did improve, but the overall accuracy was still only 53%. Figure 26 shows the feature importance by MDI score while Tables 22 and 23 show the classifier metrics.

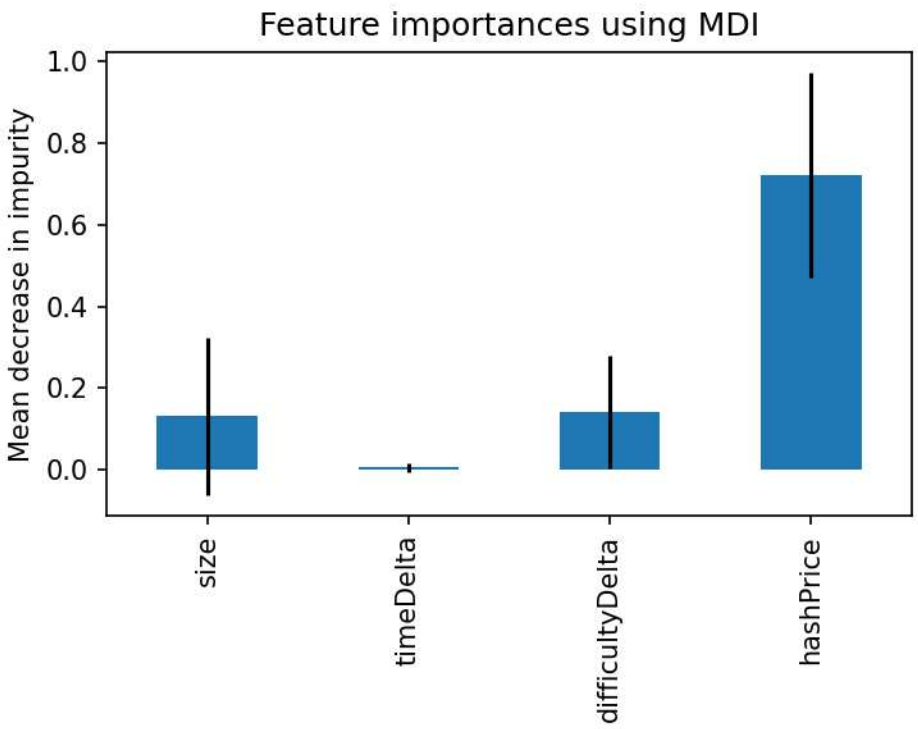

Figure 26. 2020B feature importance measured by MDI with difficulty delta and hash price. Trained on 2020A dataset.

Table 22. 2020B classifier accuracy, precision, and recall for difficulty delta and hash price. Classifier was trained on 2020A dataset.

Accuracy	Precision	Recall
0.535	0.536	0.509

Table 23. 2020B classifier metrics by block classification for difficulty delta and hash price. Classifier was trained on 2020A dataset.

Label	Precision	Recall	F1
Benign	0.53	0.56	0.55
Selfish	0.54	0.51	0.52

7.4 Discussion

Contrary to our predictions, the difficulty delta did little to improve the classifier accuracy. There could be two reasons for this. Either the difficulty does not have any indication of an attack, or we are measuring in too granular of a timeframe. The difficulty adjustment happens in Ethereum Classic on every block so the difficulty may not show significant trends. For the same reasons stated in the previous discussion, we still believe that the difficulty could detect an attack, but we needed to bin the data in a way that made the difficulty adjustment more apparent.

Once again, we saw the hash price selected as a significant factor without adding anything to the accuracy. We predicted that this has to do with the granularity of our hash price measurements. The data from NiceHash only changes every 15 minutes so the hash price associated with a block will not change that often. We could see from our graphs that the hash price does rise during an attack and levels off after the attack.

CHAPTER 8

BINNED DATA TRAINING AND RESULTS

We wanted to capture both the hash price delta and the meaningful changes in the difficulty, so we decided to create new datasets for each attack period. We generated these datasets from the data files that contained the entire attack data for all three instances as opposed to the balanced datasets.

The first datasets captured the hash price deltas and were binned every 15 minutes. The creation of the 2020A and 2020B datasets was relatively easy as we already had the hash price associated with each block. We compressed our other features into 15-minute aggregates to make them match the hash price change interval. The difficulty became the average difficulty during the interval and the difficulty delta became the delta between the 15-minute averages. The block size became the average number of transactions in the interval and the time delta became the average time delta. Since there was a possibility of mixing selfish blocks with benign blocks, we classified a binned block as selfish if there was even one block in the interval that was selfish.

Creating the binned dataset for the 2019 attack was a little different as we did not have the hash price data. In the interest of keeping the data in the same format as the 2020 attacks, we binned the data every 69 blocks. Since the block interval for ETC is 10-

15 seconds, 69-block aggregations are equivalent to 15-minutes. We followed the same heuristics for creating the binned blocks as we did with the 2020 attacks.

The new, binned datasets had fewer records in them than our balanced training data and were not perfectly balanced. We ended up removing a few records from the 2019 dataset as it was severely unbalanced. To ensure accurate prediction results, we balanced the 2020 datasets by sampling them. The newly created datasets are summarized in Table 24.

Table 24. Summary of block ranges after binning for each dataset. 2020A and 2020B datasets were manually balanced after binning.

Label	Selfish blocks	Benign blocks	Total blocks
2019 Binned	23	71	94
2020 A Binned	51	51	102
2020 B Binned	62	62	124

8.1 2019 Data Figures

Binning the data into 15-minute blocks slightly changed the format of the blocks. Figures 27-29 show the shape of the data for the binned training data. Instead of charting on the block height, we switched to charting on the index to make the charts more readable. The following data figures show the binned data before sampling. Red marks indicated a selfishly mined block.

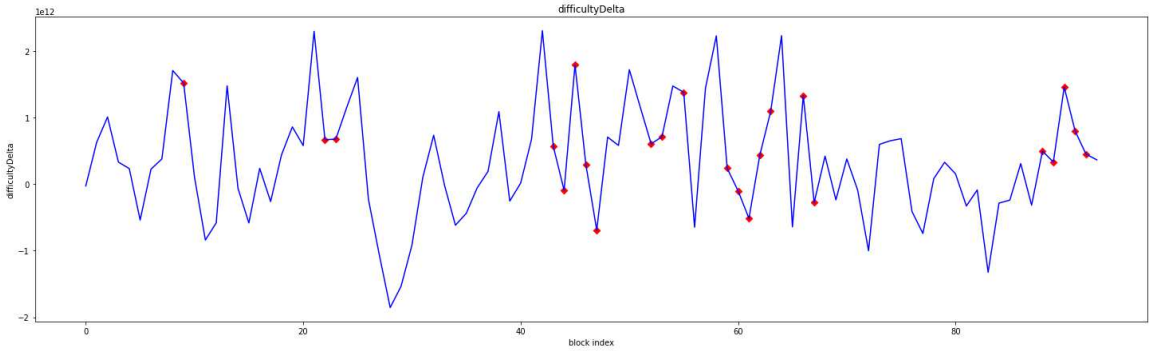

Figure 27. 2019 difficulty deltas for 69-block bins. This is calculated as the average difficulty of the current 69-block range minus the average difficulty of the previous 69-block range.

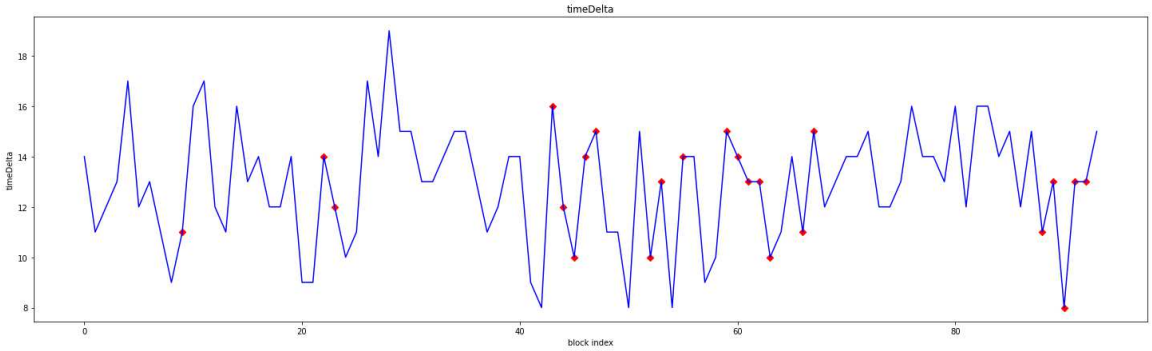

Figure 28. 2019 time deltas for 69-block bins. This is calculated as the average time delta of the current 69-block range minus the average time delta of the previous 69-block range.

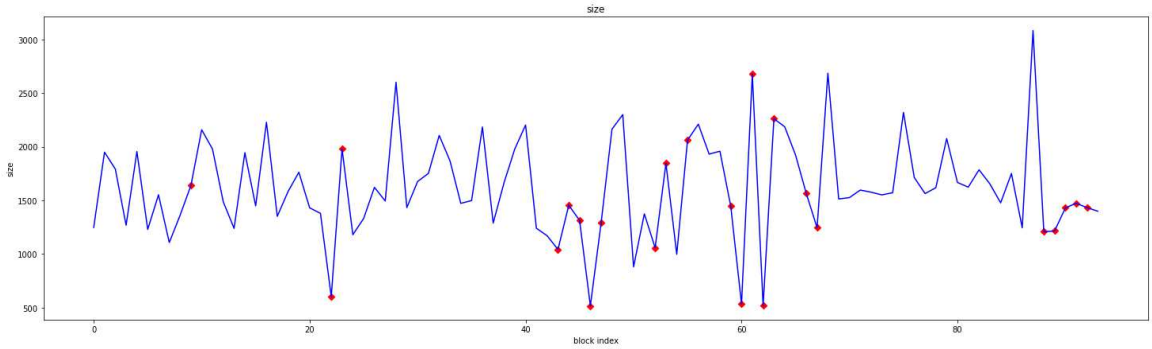

Figure 29. 2019 block sizes for 69-block bins. This is calculated as the average block size of the current 69-block range minus the average block size of the previous 69-block range.

8.2 2020A and 2020B Data Figures

For the 2020A and 2020B datasets, the 15-minute bin was easy to calculate as we could just bin the data every time the hash price value changed. Figures 30-37 show the data for the 2020 datasets in 15-minute bins. Red marks indicate selfishly mined blocks.

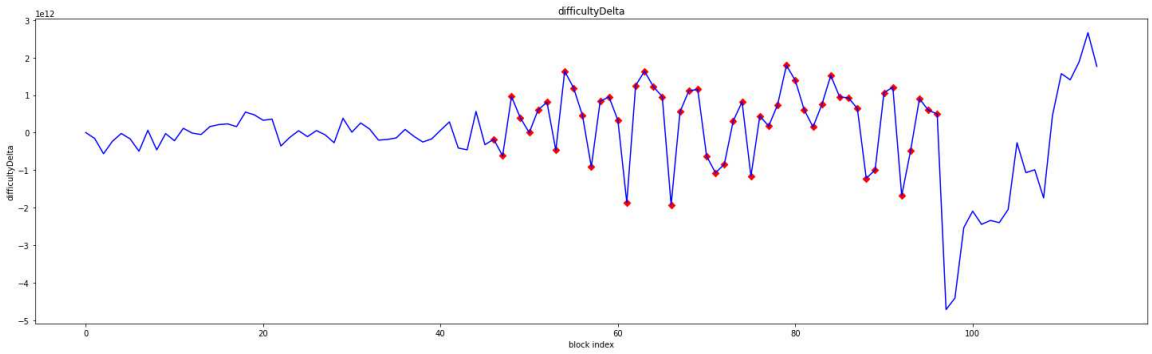

Figure 30. 2020A difficulty deltas for 15-minute bins. This is calculated as the average difficulty of the current 15-minute range minus the average difficulty of the previous 15-minute range.

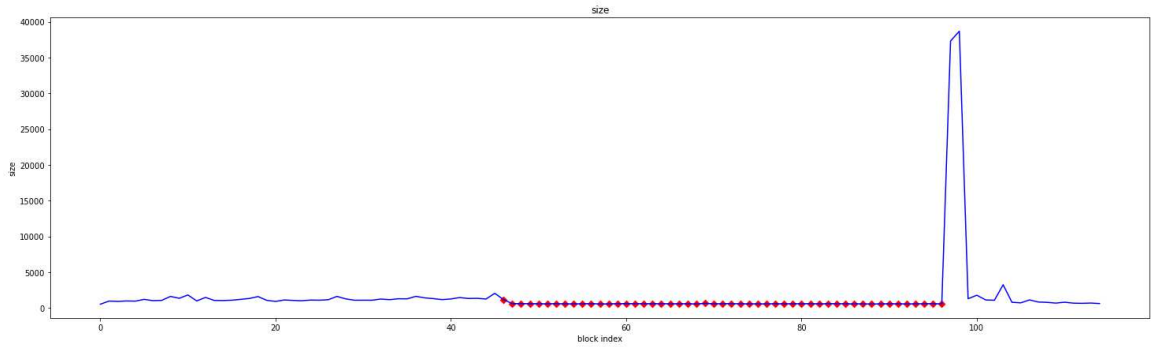

Figure 31. 2020A block sizes for 15-minute bins. This is calculated as the average block size of the current 15-minute range minus the average block size of the previous 15-minute range.

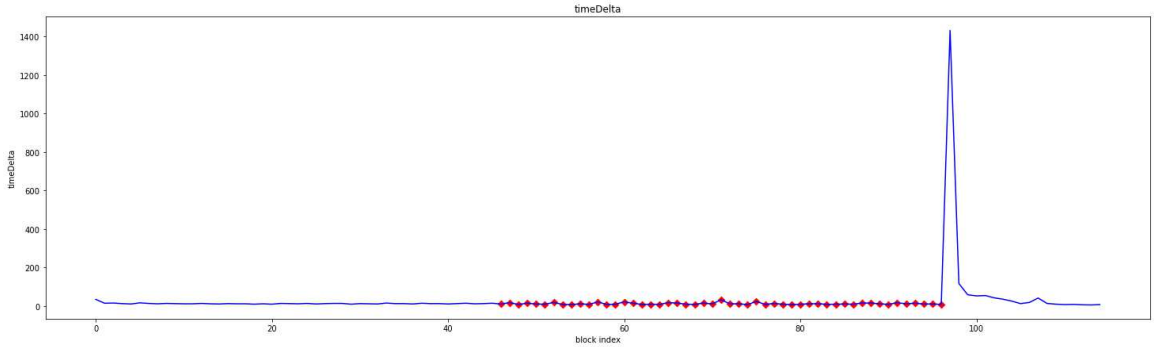

Figure 32. 2020A time deltas for 15-minute bins. This is calculated as the average time delta of the current 15-minute range minus the average time delta of the previous 15-minute range.

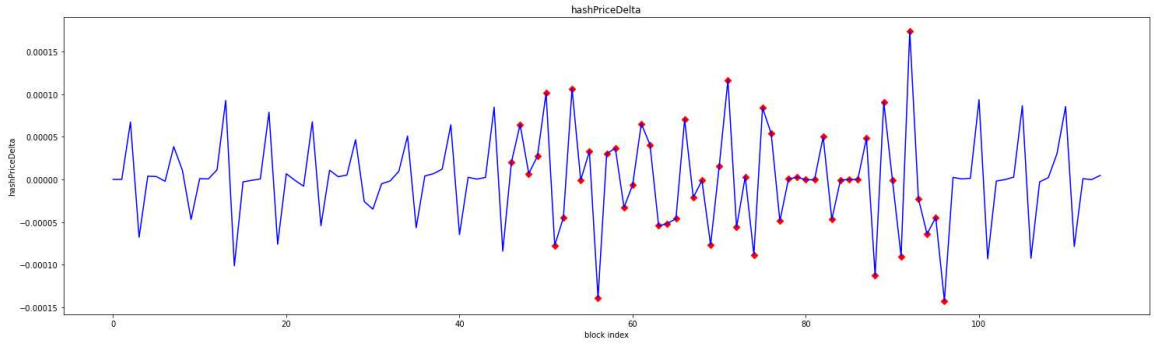

Figure 33. 2020A hash prices for 15-minute bins. The feature changes every 15 minutes so no binning was required.

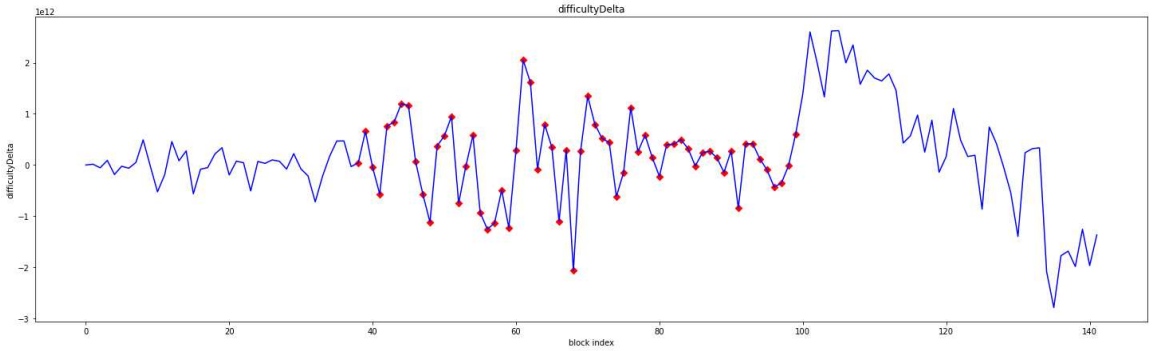

Figure 34. 2020B difficulty deltas for 15-minute bins. This is calculated as the average difficulty of the current 15-minute range minus the average difficulty of the previous 15-minute range.

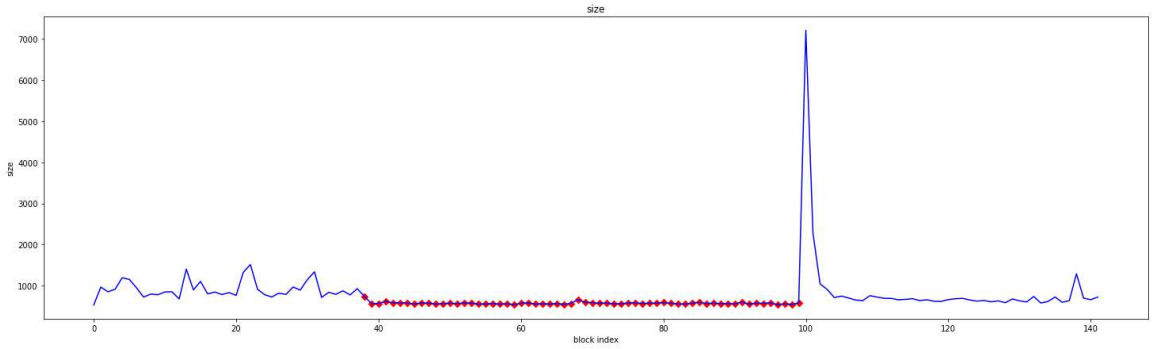

Figure 35. 2020B block sizes for 15-minute bins. This is calculated as the average block size of the current 15-minute range minus the average block size of the previous 15-minute range.

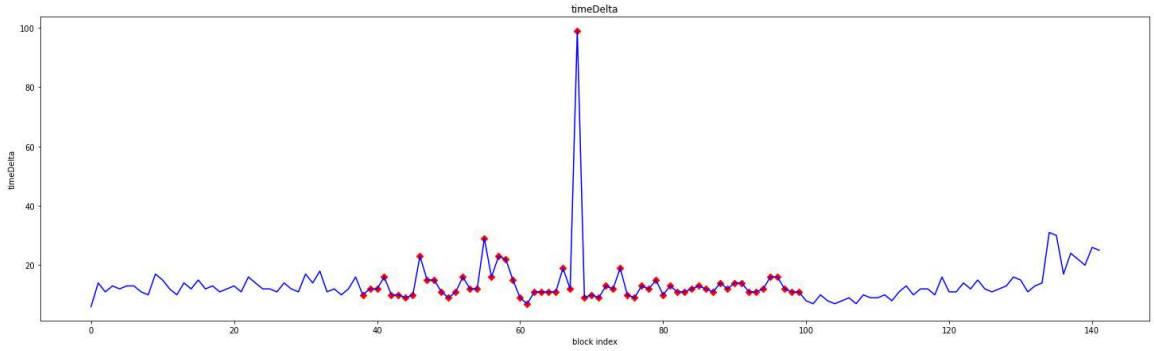

Figure 36. 2020B time deltas for 15-minute bins. This is calculated as the average time delta of the current 15-minute range minus the average time delta of the previous 15-minute range.

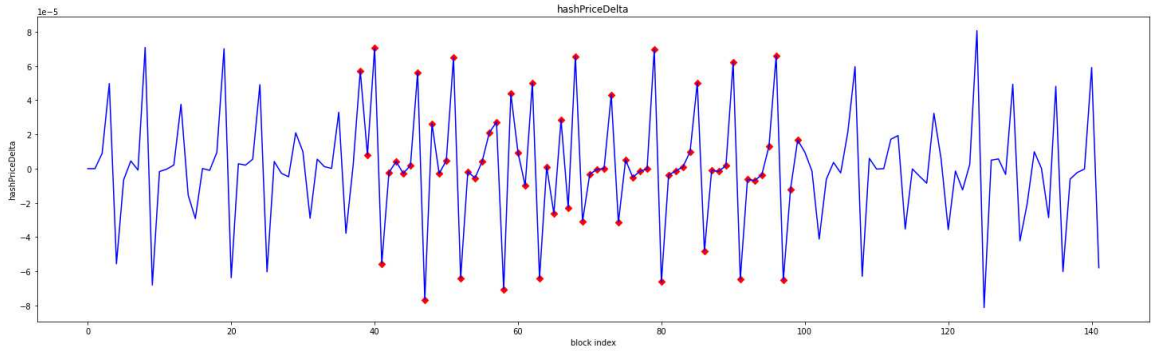

Figure 37. 2020B hash prices for 15-minute bins. The feature changes every 15 minutes so no binning was required.

8.3 2019 Training and 2020A and 2020B Predictions

Once we had the binned datasets, we repeated the same training and prediction that we did on the original datasets. We used the entire 2019 dataset to train the classifier and then predicted the 2020A and 2020B datasets. The binned datasets performed much better than the original dataset and offered a 10%-20% increase in accuracy. This was a significant increase as the classifier now had the potential to detect attacks without being

specifically trained on the dataset. Figure 38 shows the MDI scores for the features of both datasets. Tables 25-28 show the classifier metrics.

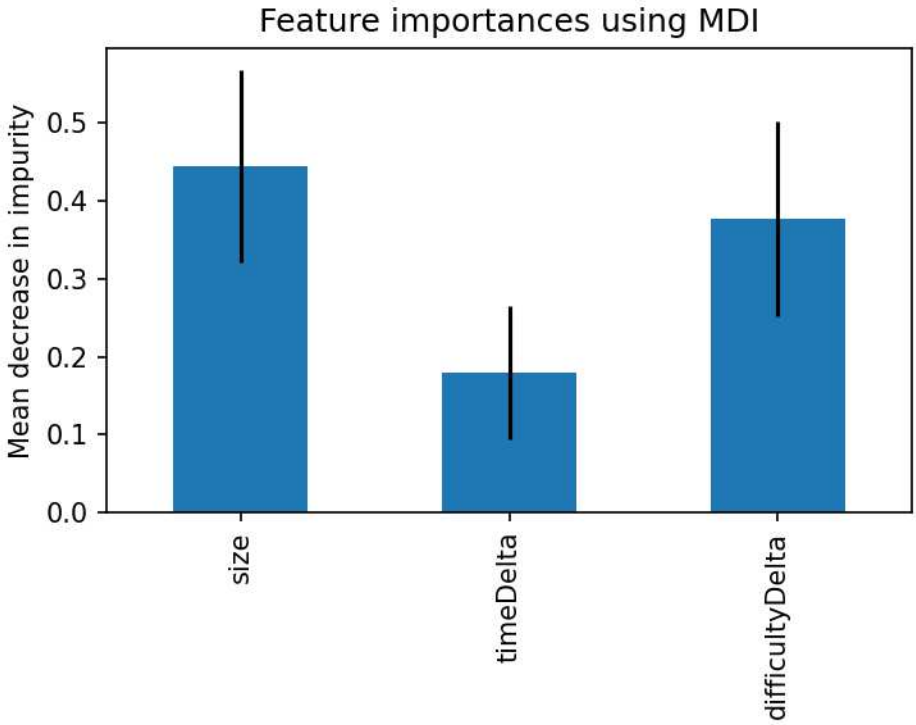

Figure 38. 2020A and 2020B feature importance measured by MDI for binned data. Classifier was trained on 2019 dataset.

Table 25. 2020A classifier accuracy, precision, and recall for binned data.

Accuracy	Precision	Recall
0.730	0.672	0.764

Table 26. 2020A classifier metrics by block classification for binned data.

Label	Precision	Recall	F1
Benign	0.79	0.70	0.74
Selfish	0.67	0.76	0.72

Table 27. 2020B classifier accuracy, precision, and recall for binned data.

Accuracy	Precision	Recall
0.612	0.532	0.919

Table 28. 2020B classifier metrics by block classification for binned data.

Label	Precision	Recall	F1
Benign	0.86	0.38	0.52
Selfish	0.53	0.92	0.67

8.4 2020A Training, 2020B Prediction

After training the classifier on the 2019 data, we wanted to see what accuracy gain we could get out of the hash price delta, so we trained on the 2020A dataset and then predicted the 2020B dataset. Our feature set included the block size, time delta, difficulty delta, and hash price delta. While the accuracy did increase (up 20% from the 2019 dataset) when the classifier was trained on the 2020A dataset, the hash price delta did not prove to be a significant factor. Figure 39 shows the MDI scores for the features while Tables 29 and 30 show the classifier metrics.

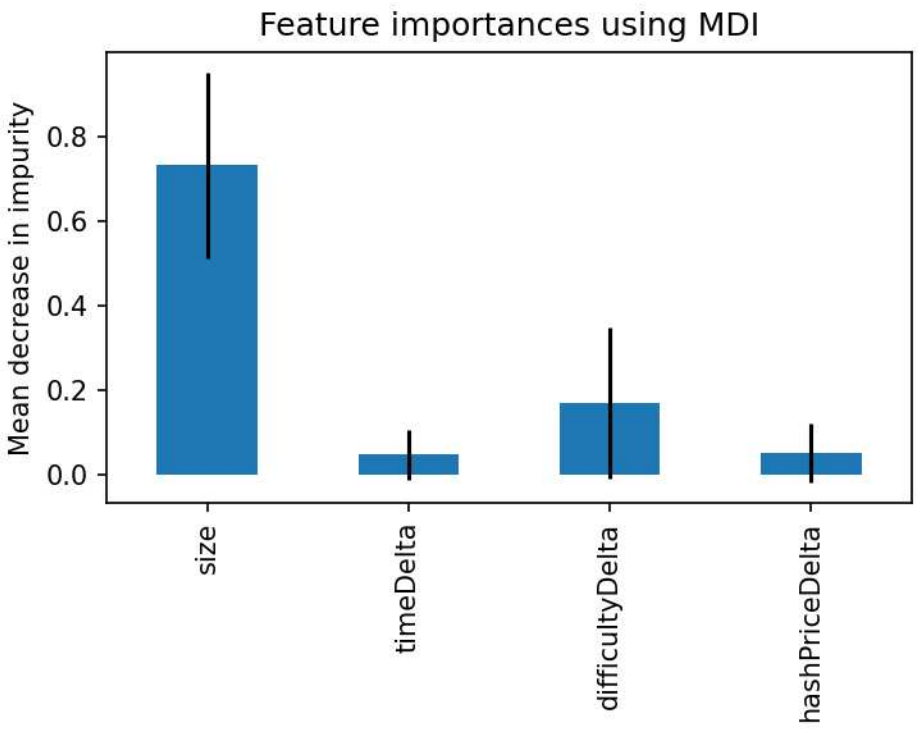

Figure 39. 2020B feature importance measured by MDI for binned data with hash price.

Table 29. 2020B classifier accuracy, precision, and recall for binned data with hash price.

Accuracy	Precision	Recall
0.873	0.789	0.967

Table 30. 2020B classifier metrics by block classification binned data with hash price.

Label	Precision	Recall	F1
Benign	0.97	0.80	0.88
Selfish	0.79	0.97	0.87

8.5 Discussion

While binning the data into 15-minute blocks did increase the accuracy, it had the limitation of mixing selfish data with benign data. In the 2019 attack, the attacks were

much smaller so the binned blocks could contain mostly benign data. In one instance, the attacker only published eight blocks so the worst-case is where 61 out of 69 blocks were benign, but the block is still classified as selfish.

Binning blocks in less than 69 bocks has an inherent problem with the hash price delta. It is only measuring every 15 minutes so binning in a more granular way could cause this factor to become irrelevant as multiple blocks record a hash price delta of zero.

CHAPTER 9

FIVE-MINUTE BIN-TIME TRAINING AND RESULTS

The last data transformation we tried on our datasets was to bin the data every five blocks. Since the average block time in ETC is 13 seconds, this works out to binning the blocks in one-minute increments. As noted in the previous chapter, we still have a limitation on how we classify blocks that contain benign and selfish data. Since the bin size is only five blocks, we settled on a simple majority vote to decide if the block was selfish or benign. If the binned block contained three or more selfish blocks, we classified it as selfish and classified all other blocks as benign.

An additional problem we faced with binning the data every five minutes was a lowered accuracy from the hash price delta. Since the hash price delta only changes every 15 minutes, the binned blocks would only provide a non-zero number for the changes to the hash price once every 15 blocks. This essentially made the hash price delta irrelevant as a factor. To get around this issue, we repeated the non-zero hash delta for each binned block until we detected another change to the hash price.

Table 31. Summary of datasets after binning every five blocks. 2020A and 2020B datasets were manually balanced after binning.

Label	Selfish blocks	Benign blocks	Total blocks
2019 Binned	184	298	482
2020 A Binned	723	723	1446
2020 B Binned	848	848	1696

9.1 2019 Data Figures

Figures 40-50 show the shape of the data after binning the original datasets into five-block bins. Blocks were labeled as selfish if three out of the five were selfish blocks and red marks indicate selfish blocks.

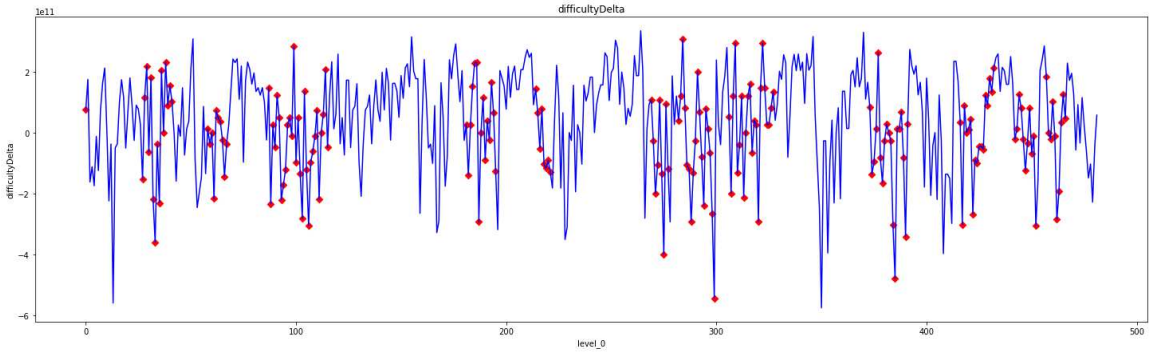

Figure 40. 2019 difficulty delta for five-block bins. This is calculated as the average difficulty of the current five-block range minus the average difficulty of the previous five-block range.

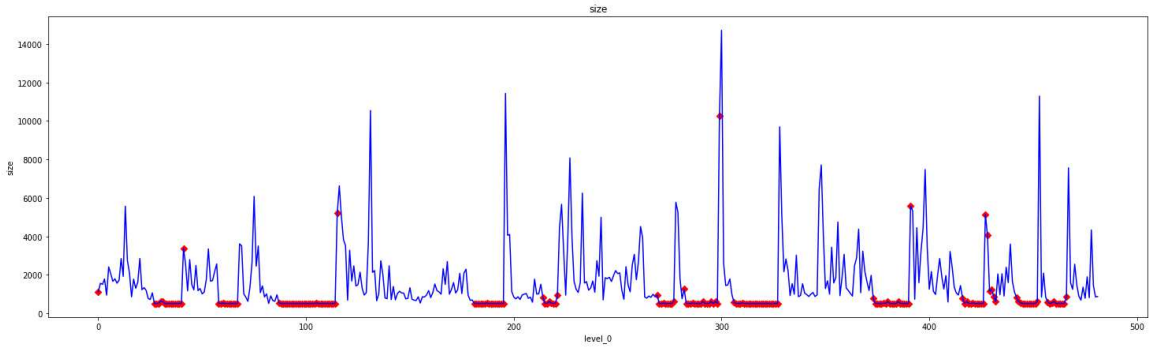

Figure 41. 2019 block size for five-block bins. This is calculated as the average block size of the current five-block range minus the average block size of the previous five-block range.

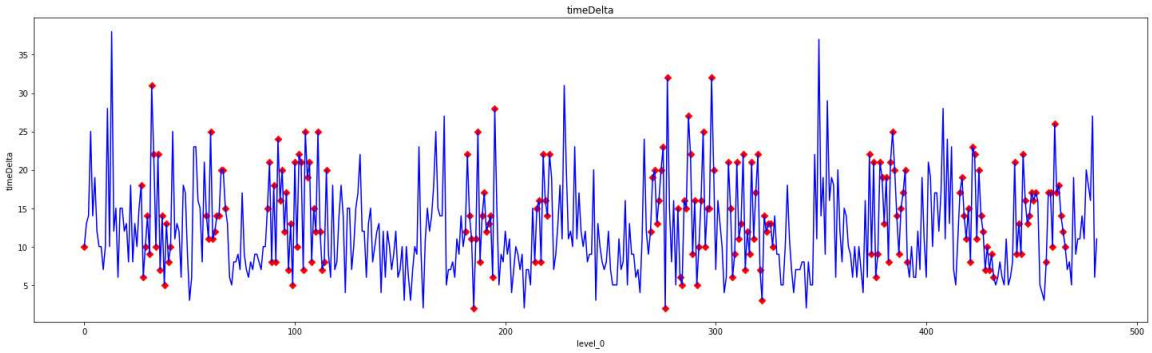

Figure 42. 2019 time delta for five-block bins. This is calculated as the average time delta of the current five-block range minus the average time delta of the previous five-block range.

9.2 2020A and 2020B Data Figures

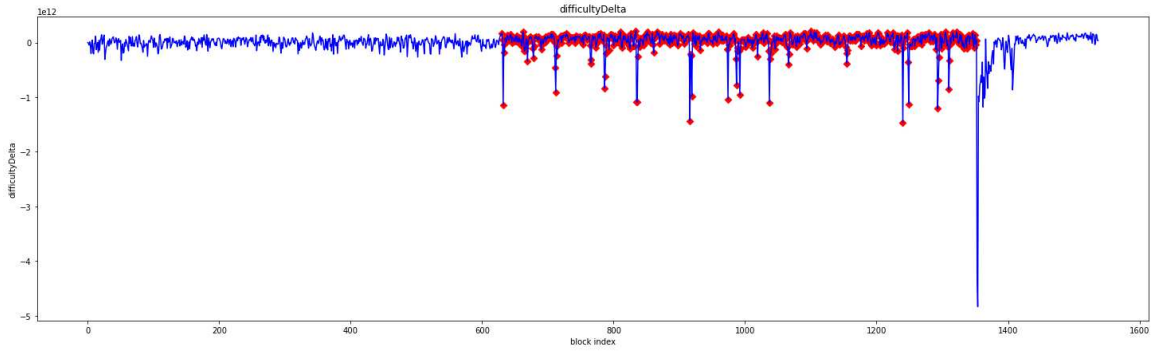

Figure 43. 2020A difficulty delta for five-block bins. This is calculated as the average difficulty of the current five-block range minus the average difficulty of the previous five-block range.

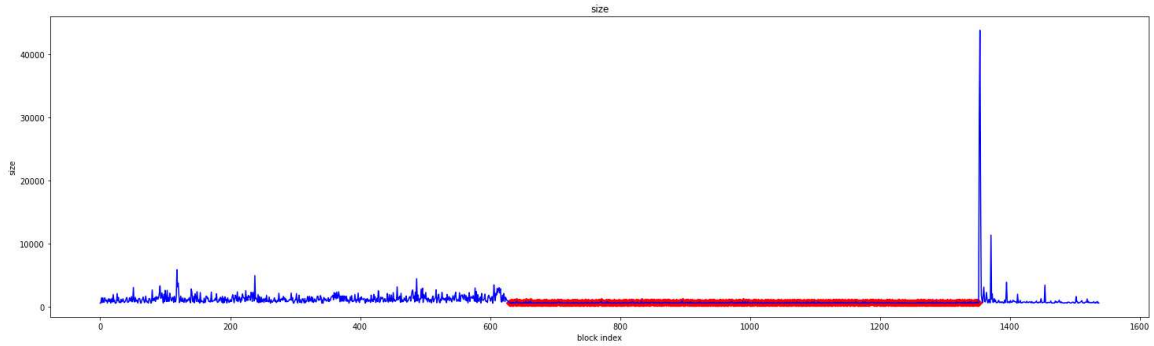

Figure 44. 2020A block size for five-block bins. This is calculated as the average block size of the current five-block range minus the average block size of the previous five-block range.

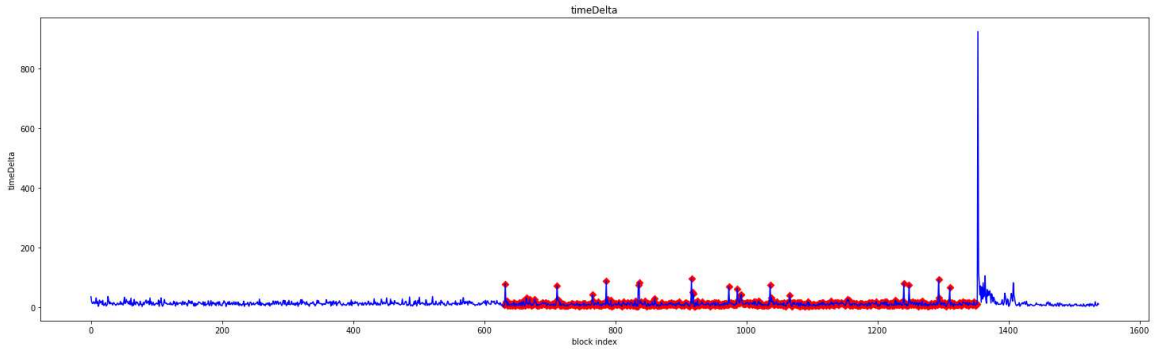

Figure 45. 2020A time delta for five-block bins. This is calculated as the average time delta of the current five-block range minus the average time delta of the previous five-block range.

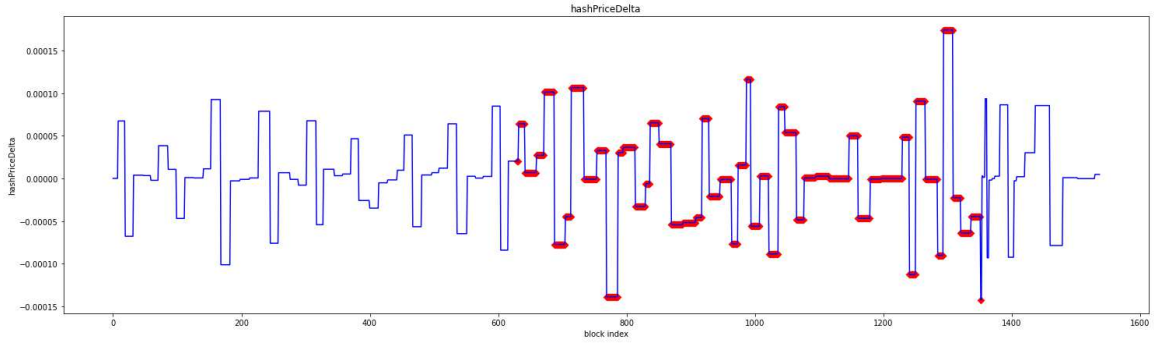

Figure 46. 2020A hash price five-block bins. This is calculated as the current hash price minus the previous hash price. If the hash price has not changed, the previous delta is repeated.

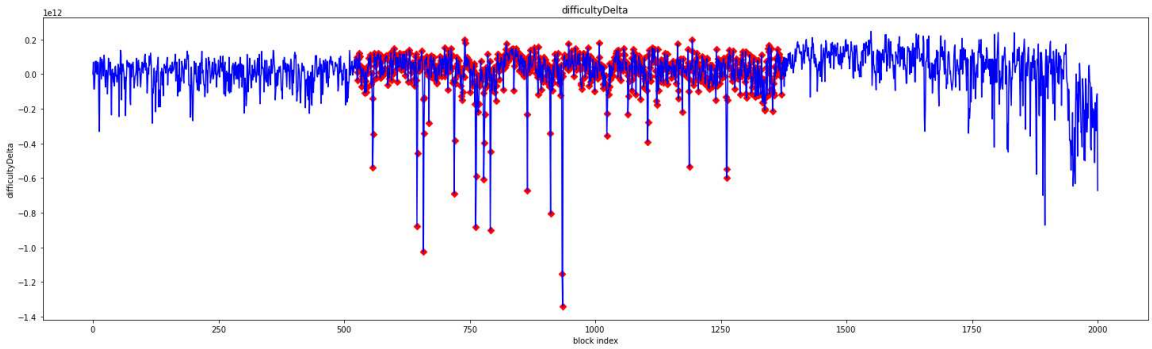

Figure 47. 2020B difficulty delta for five-block bins. This is calculated as the average difficulty of the current five-block range minus the average difficulty of the previous five-block range.

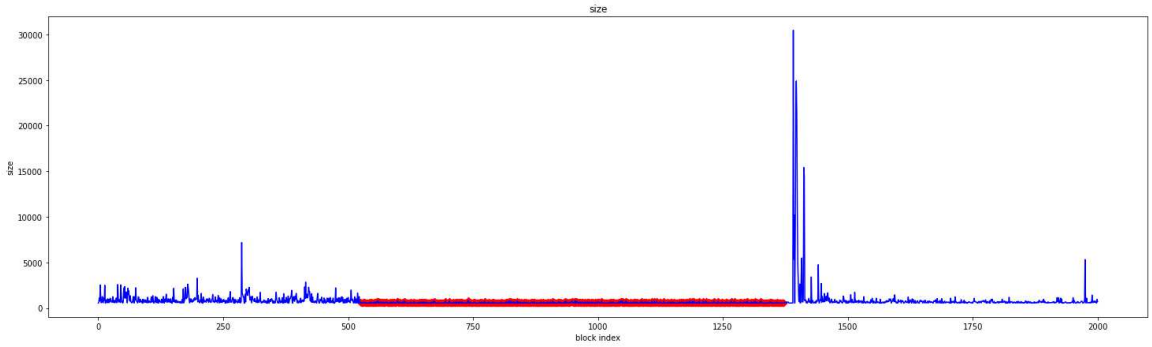

Figure 48. 2020B block size for five-block bins. This is calculated as the average block size of the current five-block range minus the average block size of the previous five-block range.

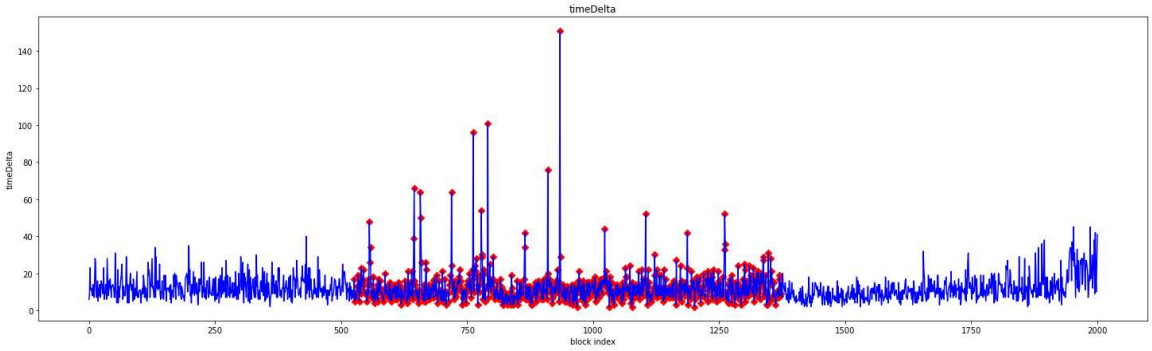

Figure 49. 2020B time delta for five-block bins. This is calculated as the average time delta of the current five-block range minus the average time delta of the previous five-block range.

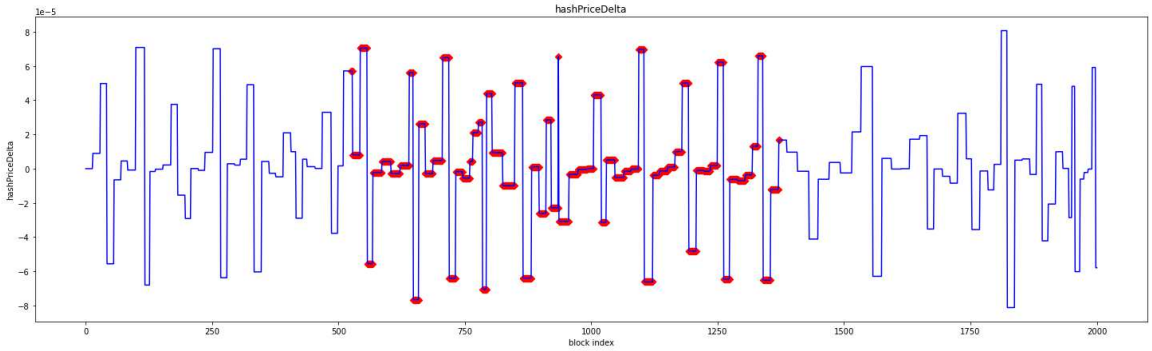

Figure 50. 2020B hash price for five-block bins. This is calculated as the current hash price minus the previous hash price. If the hash price has not changed, the previous delta is repeated.

9.3 2019 Training and 2020A and 2020B Predictions

Following the pattern defined in the previous chapters, we trained the classifier on the 2019 dataset and then used it to predict the 2020A and 2020B datasets. We used the difficulty delta, time delta, and block size features. This dataset offered us the best results of the four we tried. Figure 51 shows the MDI scores for both the 2020A prediction and the 2020B prediction.

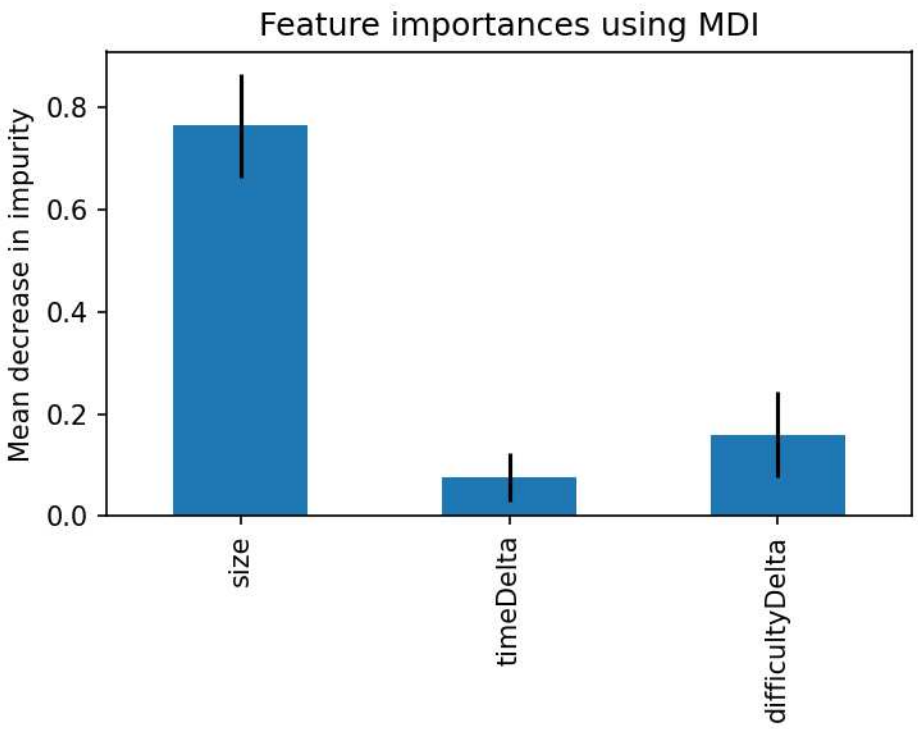

Figure 51. 2020A and 2020B feature importance measured by MDI for five-block bins.

Table 32. 2020A classifier accuracy, precision, and recall for five-block bins.

Accuracy	Precision	Recall
0.792	0.786	0.767

Table 33. 2020A classifier metrics by block classification for five-block bins.

Label	Precision	Recall	F1
Benign	0.80	0.81	0.81
Selfish	0.79	0.77	0.78

Table 34. 2020B classifier accuracy, precision, and recall for five-block bins.

Accuracy	Precision	Recall
0. 673	0.580	0.825

Table 35. 2020B classifier metrics by block classification for five-block bins.

Label	Precision	Recall	F1
Benign	0.81	0.56	0.66
Selfish	0.58	0.83	0.68

9.4 2020A Training and 2020B Prediction

Training with the 2020A dataset and predicting the 2020B dataset followed the same pattern but added the hash price delta as it was available. The feature set included the difficulty delta, time delta, block size, and hash price delta. Figure 51 shows the MDI score for predicting the 2020B dataset while Tables 36 and 37 show the classifier metrics.

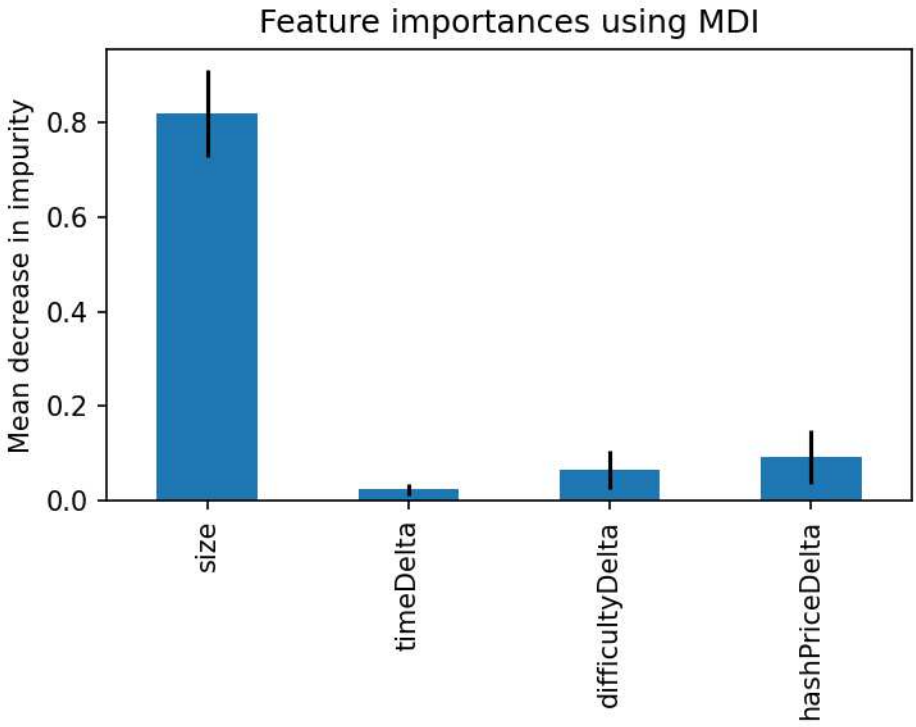

Figure 52. 2020B feature importance measured by MDI for five-block bins with hash price. Trained on 2020A dataset.

Table 36. 2020B classifier accuracy, precision, and recall for five-block bins with hash price. Trained on 2020A dataset.

Accuracy	Precision	Recall
0.946	0.895	0.988

Table 37. 2020B classifier metrics by block classification for five-block bins with hash price. Trained on 2020A dataset.

Label	Precision	Recall	F1
Benign	0.99	0.92	0.95
Selfish	0.90	0.99	0.94

9.5 Discussion

Binning the data every five blocks provided the best classifier results. This more granular binning of data allowed the classifier to discover the more nuanced differences in the data while providing a bigger picture of the data. The approach also had the benefit of evaluating blocks in a time frame that is usable by a real-time detection system.

Another benefit of binning the data every five blocks is the increased precision. The shorter error bars in the MDI graph show this increase in precision. The five-block bins made the classifier more accurate when it said that a block was selfish. Put another way, the classifier is more trustworthy when it says a block is selfish. The results from training on the 2020A dataset and predicting the 2020B dataset had very high precision and recall.

CHAPTER 10

DISCUSSION

Our classifier showed promise in being able to identify selfish mining attacks. Although the accuracy started quite low, each change we made to the dataset increased the accuracy, precision, and recall. In all cases, training on the 2019 dataset and predicting the 2020B dataset had the worst results. Similarly, training on the 2020A dataset and predicting the 2020B dataset always had the best results. There are a few possibilities as to why this. First, it is probable that the 2019 and 2020 attacks were launched by different people. The reasons behind the attacks or implementation of the attacks may be different. Since we do not have the hash price for the 2019 attack, it is possible that the attack was not implemented with rented hash power. Despite this, our classifier was still accurate at detecting the attacks. Due to the timeframe and similarity of attack lengths, it is probable that the 2020 attacks were launched by the same person or group of people. This would explain why the 2020A attack was such a good predictor of the 2020B attack. A summary of the accuracy, precision, and recall for each variation of the training data is shown in Table 38.

Table 38. Summary of classifier metrics for each variation of the training data.

Label	Training dataset	Prediction dataset	Accuracy	Precision	Recall
Original data	2019	2020A	0.537	0.564	0.330
	2019	2020B	0.527	0.542	0.359
	2020A	2020B	0.396	0.359	0.266
Original data with difficulty delta	2019	2020A	0.517	0.532	0.290
	2019	2020B	0.500	0.501	0.305
	2020A	2020B	0.535	0.536	0.509
Binned 15 minutes	2019	2020A	0.730	0.672	0.764
	2019	2020B	0.612	0.532	0.919
	2020A	2020B	0.873	0.789	0.967
Binned five minutes	2019	2020A	0.792	0.786	0.767
	2019	2020B	0.673	0.580	0.825
	2020A	2020B	0.946	0.895	0.988

10.1 Differences in 2019 and 2020 Attacks

Our next question was how are the 2019 and 2020 attacks different? Our first observation is that during the 2019 attack ranges, the network difficulty trended downward. The average delta across all 2019 attack ranges was -4378453935. A decrease in mining difficulty means that it is too hard to mine a block, so the algorithm decreases the difficulty. Usually, this means that the hash power of the network decreased. In the context of the 2019 attack, is it possible that existing network hashing power was used to launch the attack. The attacker was likely behaving as an honest network participant for a long period before the attack. The fact that they were able to outpace the rest of the network for several extended periods implies that this attacker controlled a significant amount of hash power.

The average difficulty delta across the 2020A and 2020B attacks were 4771128476 and 1708641515 respectively. A rise in difficulty such as this implies that mining power was added to the network and so the PoW algorithm increased the difficulty of mining a block. This correlates with the spike in hash price from our NiceHash data and implies that the attack rented hash power instead of using existing honest hashing power.

The other way that the attacks were different was the length of the attacks. The 2019 attack was much shorter than the 2020 attacks. In total, the 2019 attack had 920, non-contiguous blocks whereas the 2020 attacks had contiguous selfish ranges of 3,615 and 4,236 blocks.

10.2 Feature Importance

The initial set of features we set out to examine was the fork height, fork rate, timings between blocks, miner revenue-per-hour, transaction count/block size, miner wallet address/coinbase wallet address, current difficulty, difficulty delta, hash price, and hash price delta. Of these features, we measured the timing between blocks, block size, current difficulty, hash price, and hash price delta. While we initially thought that certain difficulty ranges could be more susceptible to attacks, the network difficulty fluctuated too much to provide any real insight and so we dropped this feature. The same was true for the hash price which we also dropped. We replaced the difficulty and hash price with the delta which measured the changes in these features.

We found that the feature of block size ended up being the most reliable predictor of an attack. We were even to detect attacks using this feature alone. This makes sense as

honest miners would wish to maximize revenue by including as many transactions in a block as possible. Due to the possible conflicts, selfish miners have far fewer incentives to include transactions. Smaller block sizes can help selfish miners propagate their blocks faster which further incentivizes them to keep the blocks small. On less-used blockchains, it is probable that honest miners will mine zero-transaction blocks as it depends on users submitting transactions. Despite this, multiple zero-transaction blocks still tend to be rare. Ultimately, the detection accuracy can be increased by adding other features. To the best of our knowledge, this is the only research that has examined how the block size can indicate a selfish mining attack.

When available, the hash price delta was the second most important feature. Initially, we thought that this would be the most important factor. A casual observation of the hash price history shows a spike in the price at the same time as the attacks. While the hash price was not available for the 2019 attack, it is possible that the attack came from existing hash power, and no power was rented. In this case, the hash price history would not add any accuracy. A more interesting application of the hash price history is for the detection of unlaunched attacks. In an anomaly detection context, a spike outside of normal operating ranges could help find an attack before it happens.

The difficulty delta added the second most accuracy when the hash price delta was not present. This is probably due to extra hash power entering the system and causing a difficulty adjustment. It is interesting that the difficult delta added accuracy when the classifier trained on the 2019 dataset as the difficulty delta seemed different for the 2020 attacks. We would have expected the difficulty not to add any accuracy since the difficulty adjustment appeared to behave differently during the 2019 and 2020

attacks. We also would have expected the difficulty adjustment to be a significant factor in the 2020 attacks as a lot of hashing power was added to the system.

The third most important feature was the time delta. The time delta did not add very much accuracy and was consistently the most insignificant feature. It is worth noting that it was still used in some of the decision trees. This is expected since the time delta is related to the difficulty delta. When the time delta is larger than 19 seconds, the difficulty is lower and increased when the time delta is less than 10 seconds.

CHAPTER 11

LIMITATIONS

Reaching selfish mining is a difficult proposition due to the lack of data. While selfish mining attacks do happen and have devastating effects, they are somewhat uncommon. This is to be expected as financial systems that fall victim to repeated attacks do not last long. This lack of data makes it difficult to build the large datasets needed for rigorous analysis. The available data is dependent on blog posts and developer comments to correctly label the datasets. Our main limitation is how well these articles identify selfishly-mined blocks. We hope that this classifier can solve this problem by finding selfishly mined blocks in existing blockchains or at least provide a secondary verification source for other researchers.

Another data-related limitation of this study is the lack of real-time data. While this research does explore the factors that identify historical and launched selfish mining attacks, it does not detect an unlaunched attack. Simulation of data for studying unlaunched attacks suffers from built-in assumptions on how selfish mining attacks operate. Capturing a real-time attack and being able to replay it is necessary for analysis. It is possible that this research can be used to verify the results of a simulated selfish mining attack and we will explore this possibility in Chapter 12.

While the purpose of this research was to build a generalizable solution for detecting selfish mining, it ended up being a very ETC-specific classifier. At its best, this research can be used for any cryptocurrency that uses the Dagger-Hashimoto algorithm, and at its worst, it can only be used for ETC. To use this research with other blockchains, the block size, difficulty, block times, and hash price would have to be standardized.

Many times, a cryptocurrency that is successfully attacked will switch its PoW algorithm to foil the attackers. Smaller cryptocurrencies that share an algorithm with large ones tend to be attacked more and cryptocurrencies that share a PoW algorithm with Bitcoin are more likely to be attacked. A portion of the hashing power that may not be much to the Bitcoin network could be more than 51% of the hash power used by the smaller currency. The constant switching of PoW algorithms to avoid selfish mining attacks makes it difficult to gather blockchain data for extended periods.

CHAPTER 12

FUTURE WORK

While this research uncovered some significant factors that identify selfish mining attacks, there is much work still to be done. There are several features that we did not analyze such as the miner-revenue-per-hour and the fork height and rate. In Bitcoin-based blockchains, the fork height is partially accessible through the getchaintips command. This command keeps a history of all the forks that the network node encounters. We even found a data source for this, but we ultimately did not use it since there are no known selfish mining attacks against Bitcoin. In Ethereum-based blockchains, data about orphaned blocks can be included in the blocks to increase miner rewards. Each time an orphan block is included in a valid block, there is an implied network fork. Although it is unclear if selfish miners include orphans in their blocks, this is an opportunity for research.

The most beneficial area for future work is in the detection of unlaunched attacks. Detecting in-progress and historical attacks is a step in the right direction but the true value lies in stopping an attack before it harms the system. We made modifications to a network simulator to simulate selfish mining attacks, but it needed verification to correctly replay an attack. Our research can be used to verify the output of a simulator so

that no bias is built into it. With the verified simulator, one could generate enough data to analyze a real-time attack or further study the factors that identify an attack.

Another area for future work is changing the problem from one of classification to one of anomaly detection. We used a random forest to classify the data as selfish or benign, but the spikes seen in the hash price data and difficulty adjustment lend themselves more to anomaly detection. By researching this problem from an anomaly detection standpoint, one could detect an attack before it is launched. In a related context, one could examine different classification algorithms and see if they are any better at detecting attacks. Algorithms such as k-nearest neighbor or support vector machines could be used, and the results compared against the random forest.

Finding and adding additional factors supplies another area for future research. While this research does identify a few factors, uncovering additional external data sources could add more factors to analyze. One such data source is the transaction pool. When users issue transactions, they are broadcast to the network and added to each miner's transaction pool. After a selfish mining attack reorganizes the blockchain and removes all the work done by the honest miners, the network users may resubmit their transactions and cause a spike in the number of transactions in the transaction pool. Additional factors such as this that are available from external data sources could increase the accuracy of the classifier.

As stated in Chapter 11, the classifier is not generalizable to multiple blockchains. We know of at least six cryptocurrencies that have fallen victim to selfish mining attacks but only evaluated one. If someone came up with an algorithm for generalizing the data from the different blockchains, they could create a larger dataset and further investigate

the classifier accuracy. The creation of this general-purpose classifier for selfish mining attacks would be a valuable tool for the industry.

CHAPTER 13

CONCLUSIONS

Detection of selfish mining is a difficult problem to solve. Although selfish mining was discovered in 2013 and the first selfish mining attack was recorded in 2018, this is the first paper we know of that looks at a large number of detection factors. The infrequency of these attacks and the lack of labeled data make it difficult to analyze them.

Despite these difficulties, we were able to successfully classify known examples of selfish mining. We found documented examples of selfish mining attacks from the Ethereum Classic blockchain and created a program to download these blocks from a public API and transform them into a format that was useable by a random forest classifier. We identified a data source for the hash rental power and used it to augment the existing factors in our dataset. We then went through several rounds of training and feature engineering until we found the most successful combination of features and binning timeframes. We found that binning the data every five blocks not only gave the best accuracy but also formatted the data in a way that made it conducive to a real-time detection system. Binning the data every 15 minutes was too short for real-time detection but could be useful for forensic tools. We would describe our classifier as more useful for forensic analysis, but it has the potential for real-time detection. Ultimately, this research lays the groundwork for building a successful selfish mining classifier.

The success of this classifier does not completely solve the detection problem and there is room for improvement. Much work is needed before this classifier can be used in a real-time environment. This classifier moves the body of work forward and shows that it is possible to classify these attacks in historical data while identifying important indicators. The ability to detect historical selfish mining should help blockchain developers verify the efficacy of patches intended to mitigate selfish mining attacks.

An additional benefit of this research is that researchers now have access to a labeled dataset of selfish mining. To the best of our knowledge, labeled selfish mining data did not exist before this research.

REFERENCES

[1] S. Nakamoto, "Bitcoin P2P e-cash paper," Oct. 31, 2008. Accessed: Jul. 26, 2020. [Online]. Available: https://www.metzdowd.com/pipermail/cryptography/2008-October/014810.html

[2] "Cryptocurrency Market Capitalizations," *CoinMarketCap*. https://coinmarketcap.com/ (accessed Mar. 10, 2019).

[3] "List of countries by GDP (nominal)," *Wikipedia*. Nov. 25, 2019. Accessed: Dec. 02, 2019. [Online]. Available: https://en.wikipedia.org/w/index.php?title=List_of_countries_by_GDP_(nominal)&oldid=927945586

[4] "The History of the Mt Gox Hack: Bitcoin's Biggest Heist," *Blockonomi*, Mar. 31, 2020. https://blockonomi.com/mt-gox-hack/ (accessed Sep. 21, 2020).

[5] "Hack Flashback: The Mt.Gox Hack - The Most Iconic Exchange Hack," *Ledger*, Feb. 18, 2019. https://www.ledger.com/hack-flasback-the-mt-gox-hack-the-most-iconic-exchange-hack (accessed Sep. 21, 2020).

[6] "A History of 51 Percent Attacks on Blockchains and Cryptocurrencies," *Blocks Decoded*, Mar. 08, 2019. https://blocksdecoded.com/history-51-percent-attacks/ (accessed Jun. 26, 2019).

[7] A. Hertig, “Blockchain’s Once-Feared 51% Attack Is Now Becoming Regular,” *CoinDesk*, Jun. 08, 2018. https://www.coindesk.com/blockchains-feared-51-attack-now-becoming-regular (accessed Feb. 24, 2019).

[8] M. Nesbitt, “Ethereum Classic (ETC) is currently being 51% attacked,” *The Coinbase Blog*, Jan. 07, 2019. https://blog.coinbase.com/ethereum-classic-etc-is-currently-being-51-attacked-33be13ce32de (accessed Feb. 24, 2019).

[9] I. Eyal and E. G. Sirer, “Bitcoin Mining Is Vulnerable,” *Communications of the ACM*, vol. 61, no. 7, p. 8, Jul. 2018.

[10] V. Chicarino, C. Albuquerque, E. Jesus, and A. Rocha, “On the detection of selfish mining and stalker attacks in blockchain networks,” *Ann. Telecommun.*, vol. 75, no. 3–4, pp. 143–152, Apr. 2020, doi: 10.1007/s12243-019-00746-2.

[11] J. Göbel, H. P. Keeler, A. E. Krzesinski, and P. G. Taylor, “Bitcoin blockchain dynamics: The selfish-mine strategy in the presence of propagation delay,” *Performance Evaluation*, vol. 104, pp. 23–41, Oct. 2016, doi: 10.1016/j.peva.2016.07.001.

[12] S. Bragagnolo, M. Marra, G. Polito, and E. Gonzalez Boix, “Towards Scalable Blockchain Analysis,” in *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, Montreal, QC, Canada, May 2019, pp. 1–7. doi: 10.1109/WETSEB.2019.00007.

[13] H. Kalodner, S. Goldfeder, A. Chator, M. Möser, and A. Narayanan, “BlockSci: Design and applications of a blockchain analysis platform,” *arXiv:1709.02489 [cs]*, Sep. 2017, Accessed: Feb. 27, 2020. [Online]. Available: http://arxiv.org/abs/1709.02489

[14] A. Studnev, “Ethereum Classic Attack, 8 August: Catch me if you can,” *Bitquery*, Aug. 07, 2020. https://bitquery.io/blog/ethereum-classic-attack-8-august-catch-me-if-you-can (accessed Sep. 14, 2022).

[15] A. Studnev, “Attacker Stole 807K ETC in Ethereum Classic 51% Attack,” *Bitquery*, Aug. 05, 2020. https://bitquery.io/blog/attacker-stole-807k-etc-in-ethereum-classic-51-attack (accessed Sep. 14, 2022).

[16] “Block Chain — Bitcoin.” https://developer.bitcoin.org/reference/block_chain.html (accessed Nov. 16, 2020).

[17] K. Vaidya, “Bitcoin’s implementation of Blockchain,” *Medium*, Jun. 01, 2018. https://medium.com/all-things-ledger/bitcoins-implementation-of-blockchain-2be713f662c2 (accessed Nov. 12, 2020).

[18] L. Lamport, R. Shostak, and M. Pease, “The Byzantine Generals Problem,” *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, p. 20.

[19] A. Back, “Hashcash - A Denial of Service Counter-Measure,” p. 10, Aug. 2002.

[20] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” p. 9, 2008.

[21] H. Hasanova, U. Baek, M. Shin, K. Cho, and M.-S. Kim, “A survey on blockchain cybersecurity vulnerabilities and possible countermeasures,” *International Journal of Network Management*, vol. 29, no. 2, p. e2060, 2019, doi: 10.1002/nem.2060.

[22] G. O. Karame, E. Androulaki, and S. Capkun, “Two Bitcoins at the Price of One? Double-Spending Attacks on Fast Payments in Bitcoin,” p. 17.

[23] T. Ruffing, A. Kate, and D. Schröder, “Liar, Liar, Coins on Fire!: Penalizing Equivocation By Loss of Bitcoins,” in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security - CCS ’15*, Denver, Colorado, USA, 2015, pp. 219–230. doi: 10.1145/2810103.2813686.

[24] X. Yu, M. T. Shiwen, Y. Li, and R. Deng Huijie, “Fair deposits against double-spending for Bitcoin transactions,” in *2017 IEEE Conference on Dependable and Secure Computing*, Aug. 2017, pp. 44–51. doi: 10.1109/DESEC.2017.8073796.

[25] M. Rosenfeld, “Analysis of Hashrate-Based Double Spending,” *arXiv:1402.2009 [cs]*, Feb. 2014, Accessed: Nov. 13, 2019. [Online]. Available: http://arxiv.org/abs/1402.2009

[26] “Blockchain Explorer - Search the Blockchain | BTC | ETH | BCH.” https://www.blockchain.com/explorer (accessed Feb. 20, 2019).

[27] A. Sapirshtein, Y. Sompolinsky, and A. Zohar, “Optimal Selfish Mining Strategies in Bitcoin,” *arXiv:1507.06183 [cs]*, Jul. 2015, Accessed: Apr. 13, 2019. [Online]. Available: http://arxiv.org/abs/1507.06183

[28] A. Miller and R. Jansen, “Shadow-Bitcoin: Scalable Simulation via Direct Execution of Multi-Threaded Applications,” in *Proceedings of the 8th USENIX Conference on Cyber Security Experimentation and Test*, Aug. 2015, p. 7.

[29] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, “On the Security and Performance of Proof of Work Blockchains,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS’16*, Vienna, Austria, 2016, pp. 3–16. doi: 10.1145/2976749.2978341.

[30] “Bitnodes live map.” https://bitnodes.io/nodes/live-map/ (accessed Mar. 04, 2021).

[31] C. Faria and M. Correia, “BlockSim: Blockchain Simulator,” in *2019 IEEE International Conference on Blockchain (Blockchain)*, Atlanta, GA, USA, Jul. 2019, pp. 439–446. doi: 10.1109/Blockchain.2019.00067.

[32] Y. Aoki, K. Otsuki, T. Kaneko, R. Banno, and K. Shudo, “SimBlock: A Blockchain Network Simulator,” in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Apr. 2019, pp. 325–329. doi: 10.1109/INFCOMW.2019.8845253.

[33] C. Decker and R. Wattenhofer, “Information propagation in the Bitcoin network,” in *IEEE P2P 2013 Proceedings*, Trento, Italy, Sep. 2013, pp. 1–10. doi: 10.1109/P2P.2013.6688704.

[34] A. Miller *et al.*, “Discovering Bitcoin’s Public Topology and Influential Nodes,” p. 17.

[35] M. Apostolaki, A. Zohar, and L. Vanbever, “Hijacking Bitcoin: Routing Attacks on Cryptocurrencies,” in *2017 IEEE Symposium on Security and Privacy (SP)*, May 2017, pp. 375–392. doi: 10.1109/SP.2017.29.

[36] “hash-rate,” *Blockchain.com*. https://www.blockchain.com/charts/hash-rate (accessed Feb. 05, 2021).

[37] T. Blummer, “BITCOIN MINING CALCULATIONS,” *Medium*, Feb. 07, 2018. https://medium.com/chainalysis/bitcoin-mining-calculations-ff90dc958dba (accessed Feb. 05, 2021).

[38] “Bitcoin Blocks solved On 2018-05-19 | Insight.” https://explorer.bitcoingold.org/insight/blocks-date/2018-05-19 (accessed Apr. 19, 2020).

[39] “Leading Cryptocurrency Platform for Mining and Trading | NiceHash.” https://www.nicehash.com/ (accessed Mar. 06, 2020).

[40] M. Peterson, T. Andel, and R. Benton, “Towards Detection of Selfish Mining Using Machine Learning,” in *ICCWS 2022 17th International Conference on Cyber Warfare and Security*, State University of New York at Albany Albany, New York, USA: Academic Conferences and Publishing Limited, 2022, pp. 237–243.

[41] “Double Spend Attacks on Exchanges,” *The BTG Community Forum*, May 18, 2018. https://forum.bitcoingold.org/t/double-spend-attacks-on-exchanges/1362 (accessed Apr. 19, 2020).

[42] “Double Spend Attacks on Exchanges - Difficulty adjustment,” *The BTG Community Forum*, May 21, 2018. https://forum.bitcoingold.org/t/double-spend-attacks-on-exchanges/1362/23 (accessed Sep. 15, 2022).

[43] zawy, “Answer to ‘How does Bitcoin Gold difficulty adjustment work?,’” *Bitcoin Stack Exchange*, Nov. 17, 2017. https://bitcoin.stackexchange.com/a/62873 (accessed Sep. 15, 2022).

[44] E. Iskra, “Equihash-BTG: Our New PoW Algorithm | Bitcoin Gold.” https://bitcoingold.org/equihash-btg/ (accessed Sep. 20, 2022).

[45] “Network Attack on XVG / VERGE 2018.” https://bitcointalk.org/index.php?topic=3256693.0 (accessed Sep. 20, 2022).

[46] "Network Attack on XVG / VERGE 2021." https://bitcointalk.org/index.php?topic=3256693.msg56366335#msg56366335 (accessed Sep. 20, 2022).

[47] "ZenCash Statement on Double Spend Attack," *Horizen*, Jun. 03, 2018. https://blog.horizen.io/zencash-statement-on-double-spend-attack/ (accessed Mar. 09, 2022).

[48] E. Gkritsi, "BSV Suffers 51% Attack: Report," Aug. 04, 2021. https://www.coindesk.com/markets/2021/08/04/bsv-suffers-51-attack-report/ (accessed Sep. 20, 2022).

[49] "Blockchain nodes provider — crypto node as a service," *GetBlock.io*. https://getblock.io (accessed Sep. 22, 2022).

[50] "themattman18/PrivateBlockScraper," *GitHub*. https://github.com/themattman18/PrivateBlockScraper (accessed Sep. 22, 2022).

[51] "NiceHash - API Docs." https://www.nicehash.com/docs/rest (accessed Sep. 27, 2022).

[52] "Anaconda | The World's Most Popular Data Science Platform." https://www.anaconda.com/ (accessed Apr. 25, 2020).

[53] C. Lee, "Feature Importance Measures for Tree Models — Part I," *Veritable*, Sep. 08, 2020. https://medium.com/the-artificial-impostor/feature-importance-measures-for-tree-models-part-i-47f187c1a2c3 (accessed Oct. 04, 2022).

[54] "Feature importances with a forest of trees," *scikit-learn*. https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html (accessed Oct. 04, 2022).

BIOGRAPHICAL SKETCH

Name of Author: Matthew Alan Peterson

Graduate and Undergraduate Schools Attended:

Pensacola Christian College, Pensacola, Florida

University of South Alabama, Mobile, Alabama

Degrees Awarded:

Batchelor of Science in Software Engineering, May 2013, Pensacola, Florida

Master of Science in Computer and Information Sciences, May 2017, Mobile, Alabama

Awards and Honors:

Magna Cum Laude, Cumulative 3.8 GPA, Pensacola Christian College, May 2013

Certification of Academic Excellence, Cumulative 4.0 GPA, University of South Alabama, May 2017

Publications:

"Towards Detection of Selfish Mining Using Machine Learning," in *ICCWS 2022 17th International Conference on Cyber Warfare and Security*, State University

of New York at Albany Albany, New York, USA: Academic Conferences and Publishing Limited, 2022, pp. 237–2