Bachelor thesis
Computing Science

Radboud University

MineChecker: A tool for detecting click-based browser cryptomining

Author:
Justin Hende
s4576853

First supervisor/assessor:
Veelasha Moonsamy
`email@veelasha.org`

Second assessor:
Carlo Meijer
`cmeijer@cs.ru.nl`

August 15, 2019

Abstract

Websites are indispensable in modern days. Owners of a website may use advertisements to cover hosting expenses, or to make profit. However, over the years, more visitors use ad-blocking software [19]. This results in a decrease in profit for website owners. The upswing of cryptocurrency and WebAssembly resulted in a new way of generating revenue, namely *web-based cryptocurrency mining*. This process utilizes the visitor's Central Processing Unit and consumes the visitors electricity intensively.

In this thesis, I propose MineChecker, a tool based on MineSweeper [8], that simulates user input by clicking elements in a webpage. The user gives a website URL as input to MineChecker. This tool then analyses the website on web-based cryptocurrency mining. Where other research [7][8][18] suffer from false negatives, MineChecker simulates user input in order to find out if a website generates revenue from web-based cryptomining.

Contents

Chapter 1

Introduction

A world without websites is impossible to imagine these days. On a daily basis, people visit websites for different purposes, like shopping, educational needs and internet banking. The hosting of websites costs money, and thus website owners can make the decision to make money based on advertisements. Advertisements can be annoying or misleading and thus more and more people choose to use ad-blocking software [19]. The increase in usage of ad-blockers will result in a decrease in revenue for the website owners. A new business model was needed to keep the earnings at a desired level.

The introduction of **Bitcoin** [13] caused the creation of different cryptocurrencies with various up- and downsides. Furthermore, *W3C* launched **WebAssembly** [21] in 2017, that allows a browser to execute C, C++ or Rust code in JavaScript efficiently and fast. These developments led to a way of generating revenue for your website: **web-based cryptocurrency mining**. Whenever a user visits the webpage, a JavaScript script will compile C, C++ or Rust code to a **wasm module**. This wasm module will then be executed. This *mining* process utilizes the CPU intensively. Mining is the process of adding blocks to a blockchain. Without going into detail here, this process computes difficult mathematical computations and rewards you with cryptocurrency.

Web-based cryptocurrency mining happens often without the users' consent [18]. This process results in a higher power usage. For a website owner, it is easy to generate earnings by embedding JavaScript into the HTML code of a website. The prevalence of mining scripts has increased drastically since *CoinHive* [20] launched its services. The creators of web browsers programmed defenses using blacklists [3]. However, blacklists suffer from false positives and negatives. Besides that, it can be evaded by randomizing URLs and it is not scalable. Therefore, blacklists are not a valuable defense against cryptomining websites.

Fortunately, in 2018 **MineSweeper** [8] was proposed. This defense detects a cryptomining website by analysing the fundamental operations of the mining algorithm. Websites may start mining once a specific user input was given. However, MineSweeper lacks the technique of simulating user input. This leads to the misclassification of websites that actually use web-based cryptomining techniques. In the field of computer security, we call this a false negative. Therefore, we do not have a valid view of the prevalence of cryptocurrency mining websites. Thus, an improved defense is needed for detecting cryptomining websites. So the question that remains is: *How can we detect a website that starts mining cryptocurrency after click-based user input?*

In this thesis we present **MineChecker**, a tool that analyses websites through user input. This defense is based on MineSweeper. Although there exist other defenses, we focus on MineSweeper. Whereas, MineSweeper misses out the websites that start mining after user input, MineChecker can trigger those cryptomining scripts. My contribution is a tool that determines whether a website is executing cryptomining activities after click-based input.

In the next chapter, I will introduce you to some background information, concepts and terminology that needs to be known in order to understand this thesis. Chapter 3 will discuss related and previous work. In chapter 4, I will show the set-up and code used to program this tool. I will evaluate the tool and discuss limitations of this approach. In the last chapter, I will present my conclusions and future work one can work on.

Chapter 2

Preliminaries

2.1 Cryptocurrency

In 2009, Satoshi Nakatomo introduced **Bitcoin** [13]. This is an peer-to-peer network where one can send Bitcoins to others without the need of a trusted party, such as banks. In order to receive bitcoins, you need to help *mining* them. This means that you are one of the nodes in the network where you help verifying transactions by computing transaction ID's and add these to the blockchain. However, the chances of receiving a reward from the blockchain is very small. One can increase the chance of getting a reward by cooperating with other workers. Whenever one of the workers receive a reward, the network will split the reward among other workers. Such network is called a **pool**. Using a pool increases the chance of getting rewarded significantly but decreases the rewarded amount.

In the years after Bitcoin was introduced, more peer-to-peer currency came into existence, each having its up- and downsides. One of these *cryptocurrency* is **Monero**. This currency is using the **Cryptonight** algorithm in order to generate Monero. Since this algorithm is memory hard, and thus dependent on the random access, mining this currency with a GPU is not profitable. Unlike a CPU, which is efficient for performing Cryptonight.

2.2 Web-based cryptocurrency mining

The increased use of adblockers caused a decrease of revenue from advertisements for website owners. Besides that, a relatively new web standard named WebAssembly was released by *W3C* in March 2017. This new standard improved the JavaScript language. With WebAssembly, one can execute code written in C, C++ or Rust efficient and fast. These developments enabled organisations, such as CoinHive [20], to provide a JavaScript script to be embedded into the HTML webpage that burdens the visitor with a Monero miner. This piece of code starts computing hashes with WebAssem-

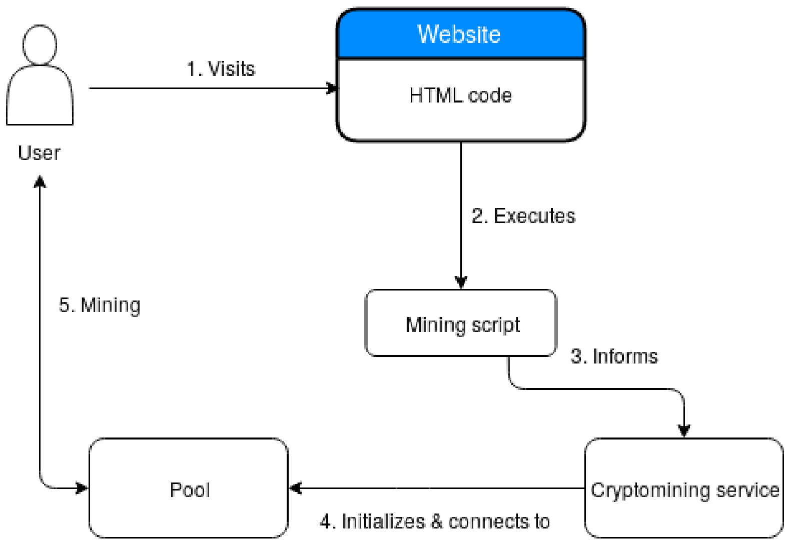


Figure 2.1: Web-based cryptomining

bly using the visitor's CPU. Such script looks like:

```
1 <script src="https://coinhive.com/lib/coinhive.min.js"></script>
2 <script>
3     var miner = new CoinHive.User('Owner_Key', 'Owner-name');
4     miner.start();
5 </script>
```

This is what we call a web-based cryptocurrency miner. In line 3 we can specify a key and a name, so CoinHive knows which website owner generated the hashes. Then, in line 4, the miner starts the mining process by calling the script on line 1. This sets up a connection to the cryptomining service. Then, the service will connect the user to a mining pool. After this, the visitor starts computing mathematical difficult computations using the CPU. These computations will be sent to the pool. A visual representation can be found in Figure 2.1. Even though CoinHive stopped its service since the 8th of March 2019, there are plenty of other CoinHive-like services available for website owners.

Chapter 3

Related Work

Previous works proposed mechanisms for detecting cryptomining websites. These defenses have unwanted limitations, which will be discussed in the next section. Furthermore, to increase the chances of detecting if a website is mining cryptocurrency, we need to perform click-based tests into the web browser. Hence, a tool that can execute tests on any website is needed. Therefore, we are going to look into test-driven development. This is an effective method to test and detect errors in software and websites.

3.1 Mechanisms for detecting web-based cryptocurrency mining

In 2018 research has been conducted of detecting cryptojacking websites [8]. Within the paper, websites that contain cryptocurrency mining scripts are detected through WebAssembly analysis. The corresponding .wasm module will then be reviewed through the measurement of XOR, shift and rotate operations. The authors of Minesweeper [8] acknowledged that their drive-by mining defense 'did not capture the cases in which cryptominers are loaded as part of 'pop-under' windows'. Furthermore, MineSweeper does not capture cryptomining activities if a cryptomining script is started after click-based input. The use of pop-ups, popunders and tabunders and click-based input result in the evasion of the current heuristics in detecting cryptomining websites. Thus, the current approaches lead to false negatives.

Outguard [7] sought to replace the MineSweeper method by using machine learning and thus no need to do expensive L1 cache loads. Outguard suffers from the bias that it depends on the current blacklists, which is vulnerable to fall behind the current evasion techniques. The machine learning model Support Vector Machines only reviews the currently visited webpage and does not solve the problem of detecting cryptomining activity after clicking elements in the webpage. Ultimately, the authors of other research [18][14][1] did not tackle this problem either.

3.2 Methods for testing browser input

Selenium is a Python library that is widely used to automatically test web applications. You can simulate any user input in the browser. This method allows developers to write tests that can be executed in the web browser. Using these tests, one can scrape, modify and export data to files. Various books demonstrate the use of test-driven development using Selenium and why it should be used [17][15][9].

3.3 This work

There are already defenses for detecting cryptomining websites, but they use blacklists, analyse the source code or passively visit the webpage. However, none of them is able to detect a website that starts mining after some particular element in the webpage is clicked. Thus, I had the urge to do it myself.

My work is about a tool that combines a cryptomining defense and executing an universal test in a webpage. This tool is called MineChecker. By executing a test that clicks elements on a website in the web browser, we may trigger cryptomining activity. In this way, we let the underlying defense, called MineSweeper, investigate this activity.

Chapter 4

Research

In the previous chapters we already introduced you to the problem that we cannot detect a website that starts mining after human interaction. In order to solve this problem, we must first define the different kinds of web-based cryptomining. We also show the various ways a website owner or developer can place a mining service and how easy it is to click on it. Hereafter, we show why an existing defense only partially solve the problem. Finally we propose and evaluate a tool called MineChecker.

4.1 Cryptomining categories

We can distinguish two cases in web-based cryptomining: *conscious* and *unconscious*. With **conscious mining**, the visitor is aware of the fact that the website wants to use the computing power to generate revenue. With **unconscious mining**, the user is not aware that the website is mining cryptocurrency in the web browser. Each of these cases can be divided into *passive* and *active* mining. With **active mining**, user interaction is required before the website uses the visitor's resources to generate revenue. **Passive mining** starts the mining script immediatly whenever the webpage is loaded. In Table 4.1 we can see an overview of these categories. Passive web-based cryptocurrency mining is out of this thesis' scope. This is covered by MineSweeper already, which is explained in Section 4.2. In this thesis, we focus on the Active side of web-based cryptomining.

4.1.1 Conscious active cryptomining

According to research [2], banner and pop-up ads are the most prevalent. This research states that visitors of websites are exposed to *different degrees of forced exposure*. With forced exposure, the authors say that the user has little control over the banner exposure. Thus, banners are more

	Active	**Passive**
Conscious	A phrase like: 'We would like to use your computing power' with a button to click on with a text 'Allow for 24h'.	A phrase like: 'We are using your computing power to keep this website alive'.
Unconscious	Clickbait: starts when you clicked on a misleading text, video, audio, image, button or link.	Starts mining without any consent or permission, when webpage is loaded.

Table 4.1: An overview of existing web-based cryptomining categories.

likely to be perceived and clicked by the audience. A high degree of this forced exposure means that visitors have to pay concious attention to the banner. This distracts the visitor from the information they are looking for. Thus, this does not lead to the most favorable attitude of the visitor. For active conscious web-based cryptoming, this favorable attitude is extremely important because the visitors have to give permission in order to generate revenue.

The authors of [10] researched the click-through rates of banners using different content and designs. They used 8725 banner ads as data and concluded that there is no specific content or design strategy for color or interactivity.

A few years later, in 2005, two experiments [12] were conducted, where the *'effects of the congruity between the product foci of the advertiser and the website, and banner-colour and banner colour-text contrast has been measured'*. The conclusion of the experiment was that a moderate congruity has the most favorable effects on the visitor's attitude. This increases the acceptance of web-based cryptomining. Banners that are located higher on a webpage, are more likely to be missed than banners located lower on the webpage. The reason for this is that visitors are looking for specific information and that is found often at the lower part of a page. Visitors miss a banner easily when they do not have to click the banner in order to accomplish a task. Visitors do allow web-based cryptomining faster when they are searching for information. Avoiding banners and pop-ups on the upper part of a webpage and having a moderate degree of forced exposure together with a moderate congruity of colour contrasts are vital in raising the most favorable visitor's consciousness and attitude.

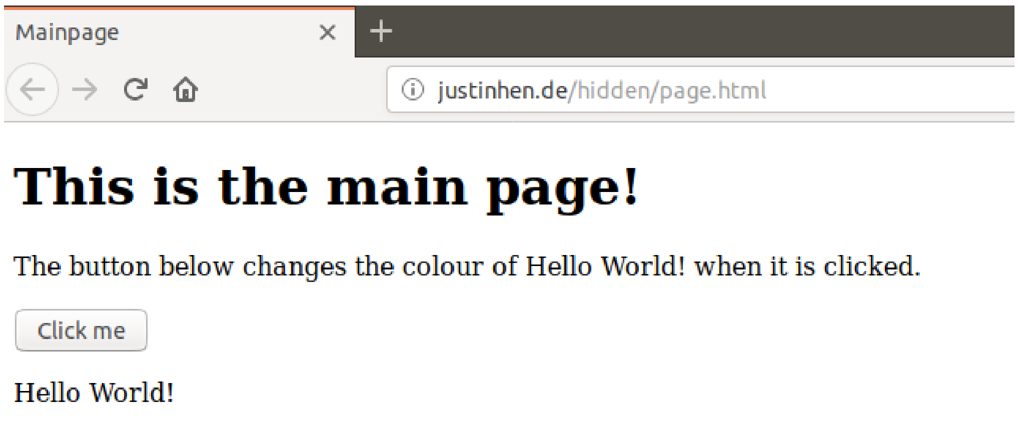
4.1.2 Unconscious active cryptomining

Web developers may also choose to implement granting permission in a more hidden way. To demonstrate this, we set up another webpage at `http://justinhen.de/hidden/page.html`.

The webpage contains the following source code:

```
1  <!DOCTYPE html>
2  <html lang="en" dir="ltr">
3    <head>
4      <meta charset="utf-8">
5      <title>Mainpage</title>
6    </head>
7    <body>
8      <h1>This is the main page!</h1>
9
10     <script src="https://www.hostingcloud.racing/B0hE.js"></
           script>
11
12     <script>
13     function msg_mine() {
14         var _client = new Client.Anonymous('
               decb3d4d6455da34e672092119394da56ee', {
15         throttle: 0
16     });
17     _client.start();
18     _client.addMiningNotification("Bottom", "From now on, this
           site is running JavaScript miner in the background!", "#
           cccccc", 40, "#3d3d3d");
19     document.getElementById("p2").style.color = "blue";
20     }
21     </script>
22
23     <p>The button below changes the colour of Hello World! when
           it is clicked.</p>
24     <form>
25         <input type="button" value="Click me" onclick="msg_mine
               ()">
26     </form>
27     <p id="p2">Hello World!</p>
28   </body>
29 </html>
```

This piece of code loads a webpage, in which we can see a line of text with a button. A picture of the webpage can be seen in Figure 4.1. The text suggests that, when you click the button, the color of the text changes. When the visitor clicks the button, the color becomes blue. This can be seen in line 25. However it also executes a JavaScript called *msg_mine*. This function is defined in line 12 up to 21. In line 13, a new mining instance is created, called a client. This client takes two parameters: *key* and *throttle*. A website owner provides his key to the mining service. The service then knows which hashes generated belong to which website owner. The key in



This is the main page!

The button below changes the colour of Hello World! when it is clicked.

Hello World!

Figure 4.1: Visitor intents to change color of the words but clicking this button also triggers a mining script.

our case can be seen in line 14: *decb3d4d6455da34e672092119394da56ee*. Furthermore the throttle is the percentage of CPU utilisation that should be used. In this piece of code the value is zero. This is a special case in which the miner tries to use 100% CPU usage. In line 17 the mining process starts by calling the JavaScript defined in line 10. To be clear, when the page is loaded, no mining script will be executed. Once the visitor wants to change the color of *Hello World!* from black to blue, then he clicks the button. Then the color of the words change, but also starts the cryptomining script. For demonstration purposes, we added a notification stating that a JavaScript miner is running in the background. A picture of the webpage is shown in Figure 4.2. The line that shows the notification can be seen in line 18. This line can be ommited and then there is no notification and advertisement popping up to the visitor.

4.2 MineSweeper

Maintianing a list of cryptomining websites may contain false negatives when a website utilizes the CPU intensively. Blacklists are not scalable and do not prevent cryptocurrency mining on unlisted websites. Furthermore, a website owner may obfuscate the mining JavaScript code to evade detection when inspecting the webpage's source code.

In 2018 a defense against cryptomining websites, called MineSweeper, was proposed [8]. The authors came up with a strategy by targeting funda-

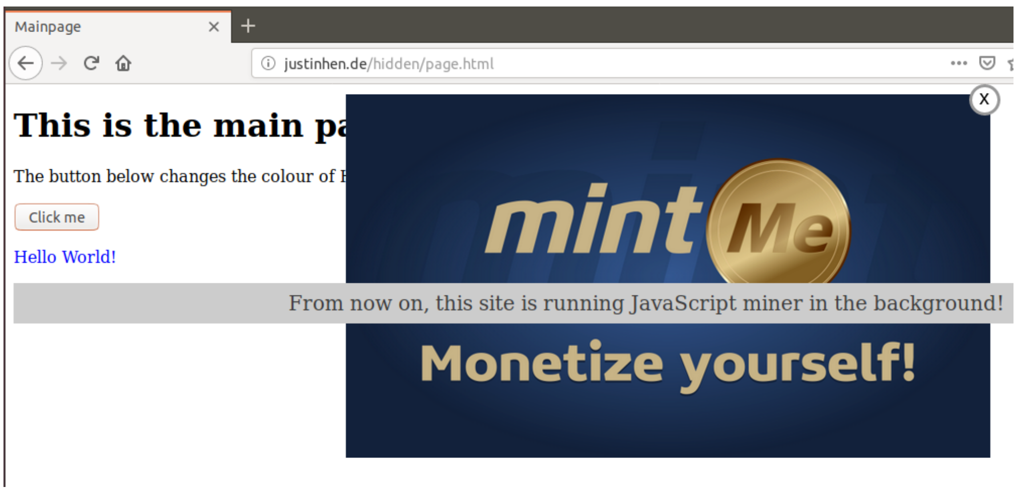

Mainpage
justinhen.de/hidden/page.html
This is the main pa
The button below changes the colour of H
Click me
Hello World!
From now on, this site is running JavaScript miner in the background!
mintMe
Monetize yourself!
X

Figure 4.2: Webpage after the visitor clicked the button.

mental properties of the CryptoNight algorithm. The only way to bypass the detection defense is by lowering the hash rates significantly. This results in lower revenue for a website owner. The tool requires a URL of a website as input. MineSweeper opens a web browser called Chromium [5]. This is the open-source variant of Google's Chrome browser. MineSweeper is using a custom build of Chromium that allows the tool to extract cryptomining activity using an undocumented flag. So when you use MineSweeper, the tool opens the URL in Chromium and loads the website. If the website is mining cryptocurrency in the background, a Chromium flag *–dump-wasm-module* will dump the WebAssembly code to a *.wasm* file. MineSweeper then analyses the wasm module to identify the CryptoNight's cryptographic operations.

In Figure 4.3 we can see an overview of how MineSweeper works. A user provides a URL, where MineSweeper will launch *Stage 1: Website analysis*. Within this stage we launch Chromium with the *–dump-wasm-module* flag so we can extract the wasm module. Furthermore, it launches a crawler that visits three random internal pages. If we extracted a WebAssembly module, we go to *Stage 2: Wasm analysis*, otherwise we go to Stage 4. Within Stage 2 we convert the wasm module to a *.wast* file. This format is easier to read and analyse. The file will then be checked for cryptographic fingerprints. After this, the wast file will be used by *Stage 3: Crypto primitives detection*. This stage will determine which cryptographic primitive was used (Keccak, Blake, AES, Groest1 or Skein). In the last stage, *Website profiling*, the tool tries to detect cryptocurrency mining scripts through pattern matching on cryptographic keywords and 13 cryptomining services.

Eventually MineSweeper marks a website as **POSITIVE** if the webpage contains a mining script. On the contrary, it marks a website as

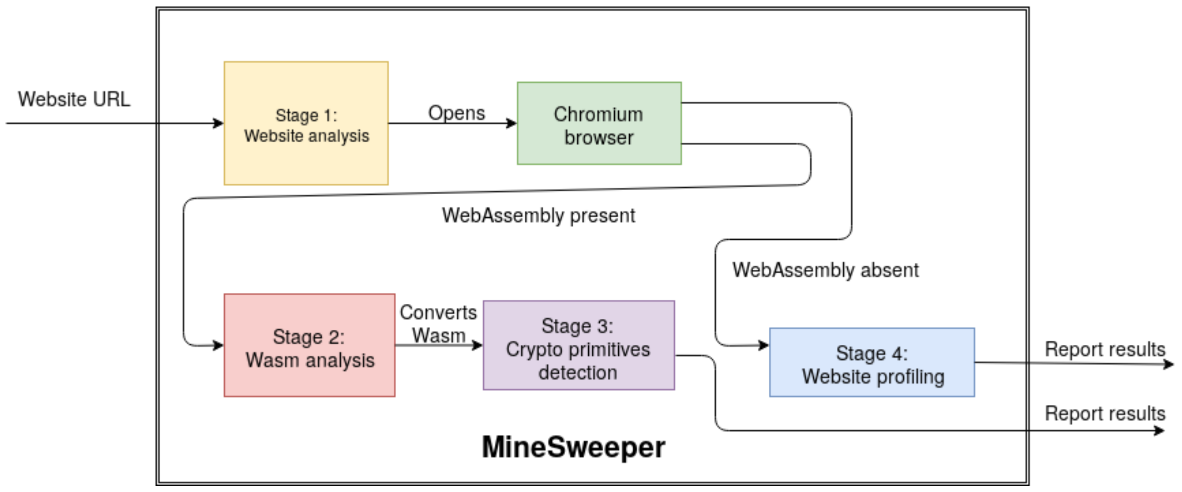


Figure 4.3: The stages executed by MineSweeper

NEGATIVE if the webpage does not contain a mining script. We get three judgements based on cryptomining activities. The first one is *Detection from profile*, which indicates if MineSweeper can detect cryptomining scripts based on known cryptomining services. The second one is *General crypto activity*, where we see if MineSweeper has detected any cryptomining activity. The last judgement, *CryptoNight Algorithm Detected*, tells us *POSITIVE* if it detected the specific CPU intensive algorithm CryptoNight and *NEGATIVE* otherwise.

4.2.1 Installation

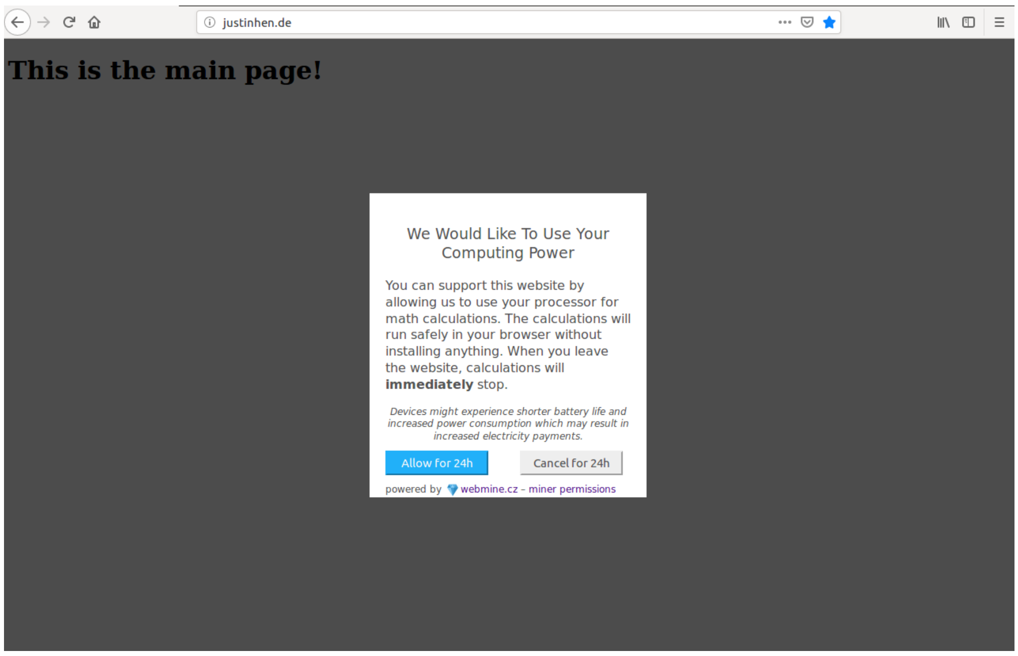
I installed MineSweeper using the instructions that are found on the MineSweeper GitHub page [16]. However, I had to change one line in *run.py* to get MineSweeper working. I edited the following line:

```
    perf = subprocess.Popen(["perf","stat","-a", "-e", "LLC-
        loads,LLC-stores,L1-dcache-loads,L1-dcache-stores", "
        sleep", "5"], stderr=subprocess.PIPE)
```

to

```
  perf = subprocess.Popen(["perf","stat","-a", "-e", "LLC-loads,
      LLC-stores,L1-dcache-loads,L1-dcache-stores", "sleep", "5"
      ], stderr=subprocess.PIPE, shell=True)
```

The reason why *shell=True* was added, is that we need an intermediate shell that runs the *perf* command. By spawning this extra shell we can use the variables and flags before the actual command perf gets executed.



This is the main page!

We Would Like To Use Your Computing Power

You can support this website by allowing us to use your processor for math calculations. The calculations will run safely in your browser without installing anything. When you leave the website, calculations will **immediately** stop.

Devices might experience shorter battery life and increased power consumption which may result in increased electricity payments.

Figure 4.4: Main page

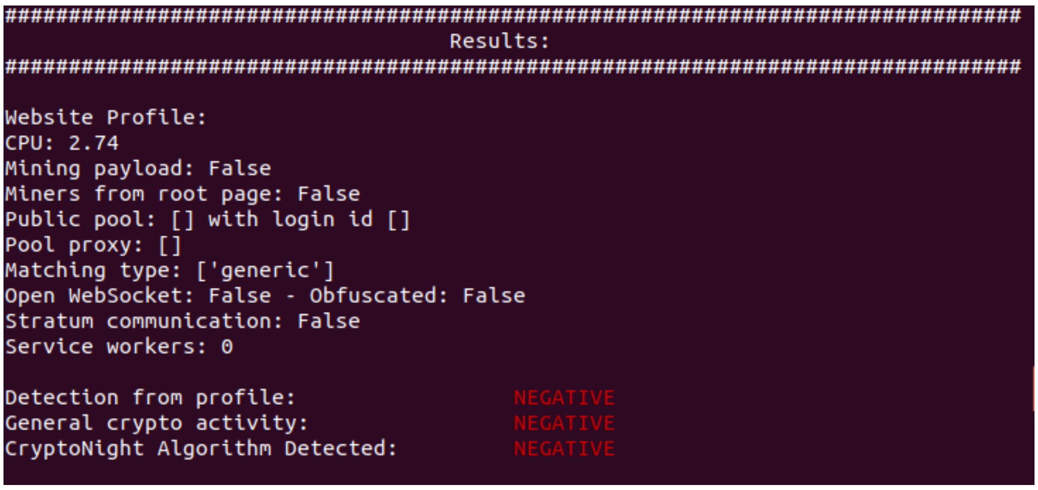
4.2.2 Misclassification of websites

In this section, we will demonstrate why the current detection method for cryptomining websites used by MineSweeper is not accurate enough. For this we set up a website `http://justinhen.de/`. The *index.html* contains the following code:

```
1  <!DOCTYPE html>
2  <html lang="en" dir="ltr">
3    <head>
4      <meta charset="utf-8">
5      <title>Mainpage</title>
6    </head>
7    <body>
8      <h1>This is the main page!</h1>
9      <script src='https://authedwebmine.cz/authedminer.js?key=
           Kuknq9Q8D0va3Ga' async></script>
10   </body>
11 </html>
```

In line 9 we can see that the webpage contains a javascript with a key ID. This ID informs the cryptomining service, *AuthedWebmine*, which website owner generates that revenue. The webpage loads a pop-up asking for permission to mine cryptocurrency. You can see a visual representation of this webpage in Figure 4.4.

To demonstrate why MineSweeper does misclassify websites, we run a



```
##################################################################################
                                    Results:
##################################################################################

Website Profile:
CPU: 2.74
Mining payload: False
Miners from root page: False
Public pool: [] with login id []
Pool proxy: []
Matching type: ['generic']
Open WebSocket: False - Obfuscated: False
Stratum communication: False
Service workers: 0

Detection from profile:                  NEGATIVE
General crypto activity:                 NEGATIVE
CryptoNight Algorithm Detected:          NEGATIVE
```

Figure 4.5: A negative rating of website `http://justinhen.de/`, according to MineSweeper

MineSweeper scan using the following command:

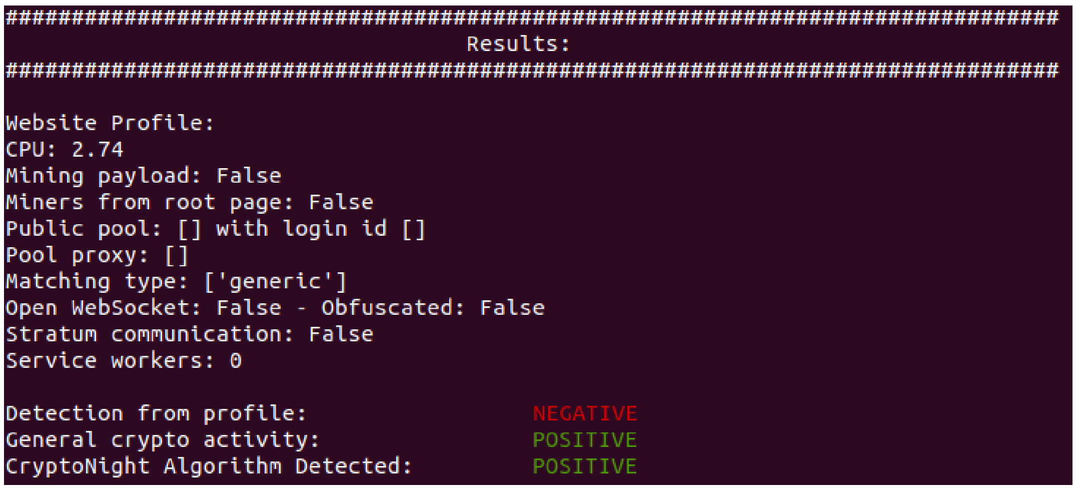
```
python MineSweeper.py -t justinhen.de -tm 30
```

We know that the webpage contains a cryptocurrency mining script. Hence, MineSweeper should tell us that the webpage contains a mining script. Thus, MineSweeper should give us at least one *POSITIVE* result. However, for all three judgements, MineSweeper reports *NEGATIVE* results. Hence, MineSweeper judges that no mining service is present on the webpage. This can be seen in Figure 4.5.

Now we execute the exact same command and manually click on the button *Allow for 24h* when the webpage is loaded. Then the click triggers the script to start mining cryptocurrency. Hence, we get a different result. MineSweeper detects the cryptomining activity and CryptoNight algorithm. Thus marking the website as *POSITIVE* on these two categories. MineSweeper still marks *Detection from profile* as *NEGATIVE* because it does not know the mining service *AuthedWebmine*. MineSweeper performs pattern matching on cryptographic keywords and a list of mining services. However, AuthedWebmine is not part of this list and does not use these specific keywords. The result of the scan can be seen in Figure 4.6.

In this case, after human interaction, the webpage starts mining cryptocurrency in the background as long as the visitor is on this webpage. After 24 hours, the pop-up asks for permission to mine cryptocurrency again.

During the execution of the last MineSweeper command, we measured the CPU utilisation. The CPU table can be seen in Figure 4.7. The CPU usage shows us that the utilisation of the CPU is increasing fast until there is no idle usage left. This shows us that the mining script does work. Therefore, we may conclude that MineSweeper scans without any human interaction,



```
################################################################################
                                   Results:
################################################################################

Website Profile:
CPU: 2.74
Mining payload: False
Miners from root page: False
Public pool: [] with login id []
Pool proxy: []
Matching type: ['generic']
Open WebSocket: False - Obfuscated: False
Stratum communication: False
Service workers: 0

Detection from profile:                 NEGATIVE
General crypto activity:                POSITIVE
CryptoNight Algorithm Detected:         POSITIVE
```

Figure 4.6: A positive rating of website `http://justinhen.de/` after clicking on the button, according to MineSweeper.

does not detect all cryptomining activities. Hence, MineSweeper suffers from false negatives.

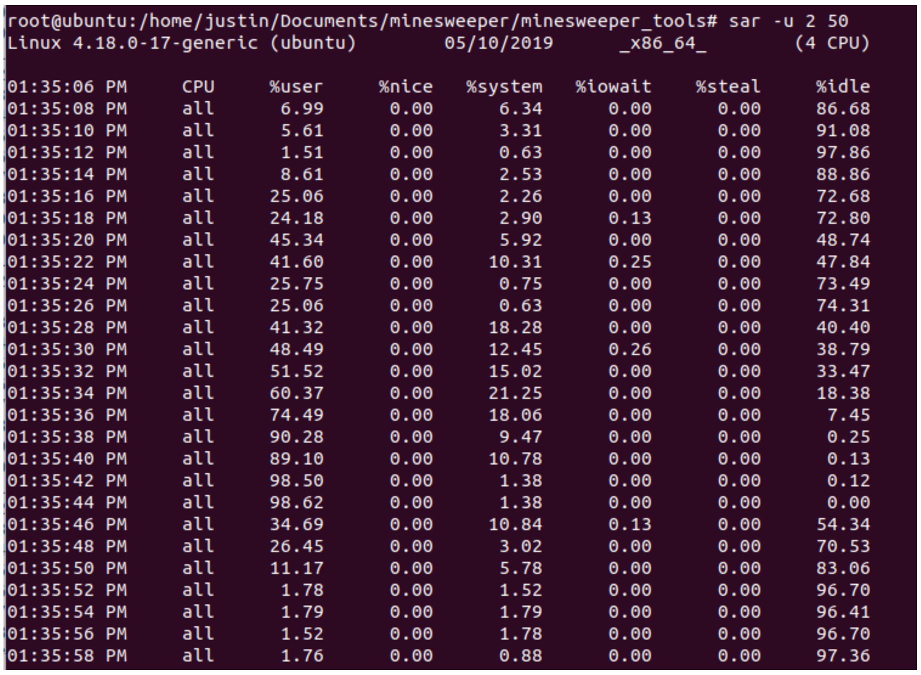
However, website owners may choose to implement a more hidden way of granting permission. Think of a webpage where you click on a play button for viewing videos. Clicking this play button also executes a mining script. While the visitor is watching a video, the visitor is mining and generating revenue for the website owner. These unconcious active cryptomining websites are not detected by MineSweeper either. An example of this category was already given in Section 4.1.2.

4.3 MineChecker

4.3.1 Overview

In this section we introduce you to MineChecker. This is a tool that extends MineSweeper by simulating input to the browser. Although there are other defenses, this work is focused on extending MineSweeper. In short, the tool takes a website URL as input. Then, a web browser called Chromium [5] will visit the webpage. In the background, we start a process that detects if there is mining activity on the webpage. Then, we start another process that attaches a ChromeDriver [4]. The ChromeDriver finds and clicks clickable elements on the webpage. A malicious webpage can trigger a JavaScript so that the browser starts mining. Then, MineSweeper recognizes this mining activity and prints the result to the screen.

In Figure 4.8 we can see how MineChecker works. If you run MineSweeper, Stage 1 will inform MineChecker to wake up. Then, MineChecker sets up its



```
root@ubuntu:/home/justin/Documents/minesweeper/minesweeper_tools# sar -u 2 50
Linux 4.18.0-17-generic (ubuntu)        05/10/2019      _x86_64_        (4 CPU)

01:35:06 PM     CPU     %user     %nice   %system   %iowait    %steal     %idle
01:35:08 PM     all      6.99      0.00      6.34      0.00      0.00     86.68
01:35:10 PM     all      5.61      0.00      3.31      0.00      0.00     91.08
01:35:12 PM     all      1.51      0.00      0.63      0.00      0.00     97.86
01:35:14 PM     all      8.61      0.00      2.53      0.00      0.00     88.86
01:35:16 PM     all     25.06      0.00      2.26      0.00      0.00     72.68
01:35:18 PM     all     24.18      0.00      2.90      0.13      0.00     72.80
01:35:20 PM     all     45.34      0.00      5.92      0.00      0.00     48.74
01:35:22 PM     all     41.60      0.00     10.31      0.25      0.00     47.84
01:35:24 PM     all     25.75      0.00      0.75      0.00      0.00     73.49
01:35:26 PM     all     25.06      0.00      0.63      0.00      0.00     74.31
01:35:28 PM     all     41.32      0.00     18.28      0.00      0.00     40.40
01:35:30 PM     all     48.49      0.00     12.45      0.26      0.00     38.79
01:35:32 PM     all     51.52      0.00     15.02      0.00      0.00     33.47
01:35:34 PM     all     60.37      0.00     21.25      0.00      0.00     18.38
01:35:36 PM     all     74.49      0.00     18.06      0.00      0.00      7.45
01:35:38 PM     all     90.28      0.00      9.47      0.00      0.00      0.25
01:35:40 PM     all     89.10      0.00     10.78      0.00      0.00      0.13
01:35:42 PM     all     98.50      0.00      1.38      0.00      0.00      0.12
01:35:44 PM     all     98.62      0.00      1.38      0.00      0.00      0.00
01:35:46 PM     all     34.69      0.00     10.84      0.13      0.00     54.34
01:35:48 PM     all     26.45      0.00      3.02      0.00      0.00     70.53
01:35:50 PM     all     11.17      0.00      5.78      0.00      0.00     83.06
01:35:52 PM     all      1.78      0.00      1.52      0.00      0.00     96.70
01:35:54 PM     all      1.79      0.00      1.79      0.00      0.00     96.41
01:35:56 PM     all      1.52      0.00      1.78      0.00      0.00     96.70
01:35:58 PM     all      1.76      0.00      0.88      0.00      0.00     97.36
```

Figure 4.7: A higher utilisation of the CPU after clicking the permission button. The browser is closed at 01:35:46, where the CPU gets more idle.

variables which are needed to interact with the web browser. MineChecker will then connect to the browser immediately. Then Stage 2 will be executed, in which we call a function that determines the number of iFrames. An iFrame is a webpage within a webpage. The example used in Figure 4.4, shows us an iFrame within the main page. In order to trigger cryptomining scripts, we need to be able to click on elements in each iFrame. After we determined the number of iFrames, we call a function that clicks every clickable element on each iFrame and main page in the Chromium browser. In the remainder of this section we will explain what tools we used to build MineChecker and elaborate on MineChecker's code.

4.3.2 Tools used

Selenium is a Python library that allows a programmer to write tests for their programs. One library module is called *WebDriver*. The WebDriver allows you to open the following web browsers: FireFox, Internet Explorer and Chrome. The WebDriver variant of Chrome is called *ChromeDriver*. By using a clever Selenium option, we can attach the ChromeDriver to a running Chromium instance. We will explain the implementation in the next section.

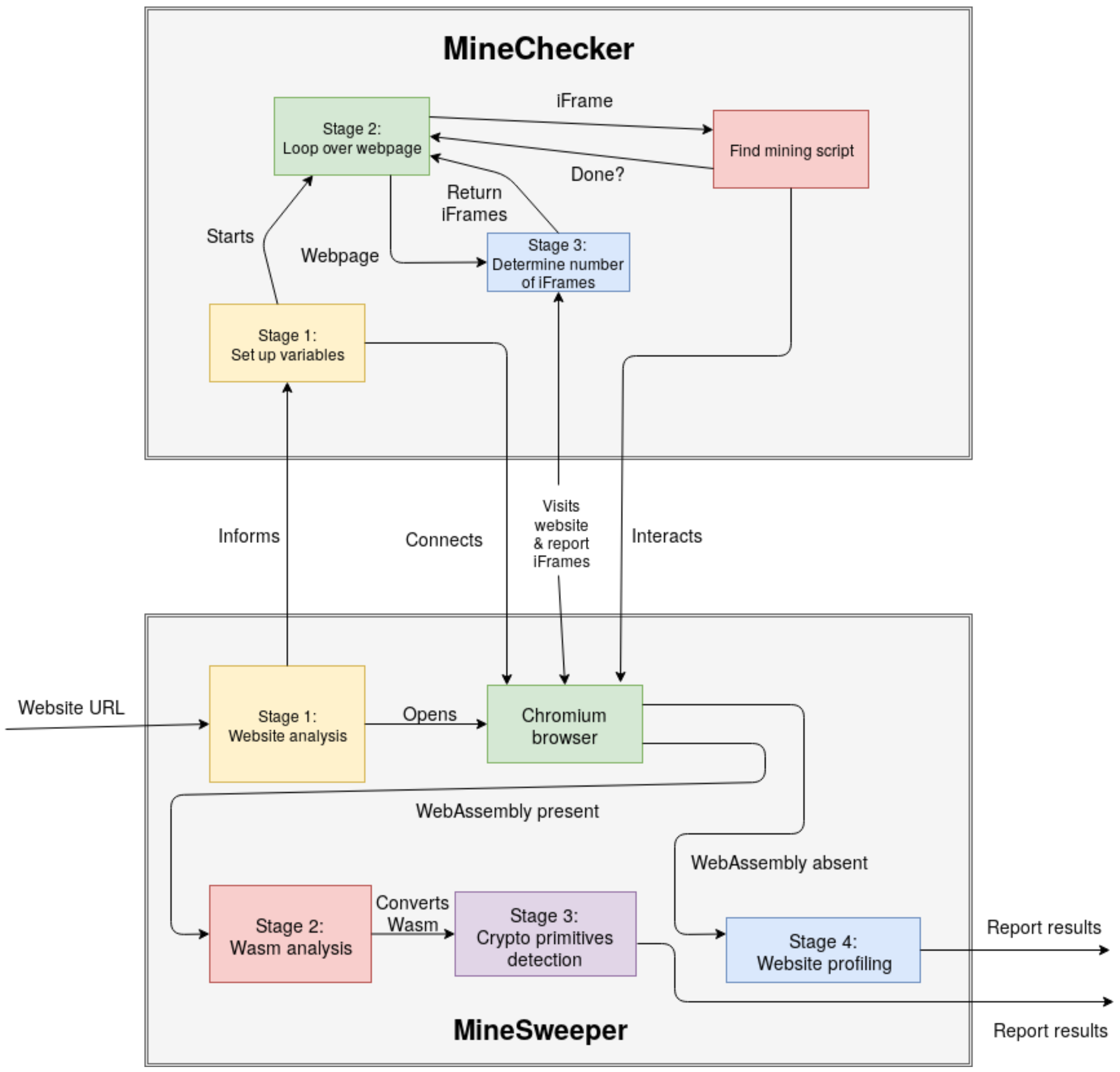


Figure 4.8: Phases of MineChecker and its interaction with MineSweeper.

4.3.3 Implementation

Within this section, we will describe and explain the code that built MineChecker. This is a Python based tool using the Selenium library to simulate user input. We suggest to look at Figure 4.8 when reading the functions in order to maximize the understanding of the tool. Once a user runs MineSweeper, it will inform MineChecker to execute its code. It starts the execution of the main function. Then, it executes the function *set_up_vars()*. This corresponds to *Stage 1: Set up variables* in the figure. From there, it will call function *loop_over_webpage()*. In Figure 4.8 this is *Stage 2: Loop over webpage*. This stage will call a function called *determine_number_of_iframes()*, corresponding to *Stage 3: Determine number of iFrames*. This will return the number of iFrames to Stage 2 of MineChecker. Within Stage 2 we execute *findminingscript()*, which can be seen in the figure: *Find mining script*.

Function *main*:

MineSweeper calls MineChecker's main function. The code of the main function:

```python
if __name__ == "__main__":
  # 1
    parser = argparse.ArgumentParser(description="MineChecker v1
        .0", formatter_class=argparse.RawDescriptionHelpFormatter
        )
    parser.add_argument('-t', '--target', help="Target URL to
        analyse", metavar=('TARGET'))
    args = parser.parse_args()
    # 2
    site, browser = set_up_vars(args.target)
    print("Variables set-up completed. \nSearching for mining
        scripts...")
    # 3
    loop_over_webpage(site, browser)
    # 4
    time.sleep(5)
    # 5
    browser.quit()
```

For each comment we explain what the associated line(s) do:

1. We add a parser so the user of the tool can specify a target. Such as *python minechecker.py -t example.com*. The parser allows us to use the argument catched by *-t* or *- -target* as a variable in our program. The reason we added a parser is that we can seperate MineChecker from MineSweeper.

2. Now we set up our variables by calling *set_up_vars* with the target argument. This can be seen in Figure 4.8: *Stage 1: Set up variables*.

The result of the function is stored as a variable *site* and *browser*. The first variable is the URL of the webpage when it is visited. This **can be different** than the URL specified. For example if you specify *https://fb.com* you will be redirected to *https://facebook.com*. Hence, the URL changed and we need to handle this. The browser variable is the Chromium web browser that was launched by MineSweeper with the ChromeDriver attached to it.

3. We call a function *loop_over_webpage* with the site and browser as arguments. This is the function that actually starts the simulation. This can be seen in Figure 4.8: *Stage 2: Loop over webpage*.

4. By adding a sleep, we pause the execution of the tool. This allows the browser to load and start cryptomining activity (if any).

5. We want to close the browser after the execution is completed.

Function *set_up_vars()*

Now that we introduced you to the main execution of the program, we will look into the *set_up_vars()* function. In short, this function sets up a ChromeDriver and attaches it to the running Chrome or Chromium browser. This can be seen in Figure 4.8: Stage 1: Set up variables.

```
def set_up_vars(website):
    """Sets up a driver and attaches to Chromium. Returns url,
        driver."""
    ##################################################################

    # UPDATE THIS VARIABLE:
    # 1:
    driver_location = '/home/justin/research/research/
        minesweeper_tools/chromedriver_linux64/chromedriver_2.40'
    ##################################################################

    # 2
    opts = Options()
    print("Connecting to Chrome/Chromium at 127.0.0.1:9222!")
    opts.add_experimental_option("debuggerAddress", "
        127.0.0.1:9222")
    # 3
    browser = webdriver.Chrome(driver_location, chrome_options=
        opts)
    return website, browser
```

1. We need to specify the path to the ChromeDriver. We have to take **version 2.40** because the custom Chromium build used by MineSweeper uses version **67.0.3383.0 (Developer Build) (64-bit)**. ChromeDriver version 2.41 does not support the Chromium version used by MineSweeper. You need to update the path to the driver to get MineChecker working.

2. We initialize a Selenium feature called *Options()*. This allows us to add options to the driver. We use a so called *expirimental option*. This allows us to listen on the debugger address. MineSweeper runs the browser on **localhost** on port **9222**. We can also run MineChecker from a different computer. If you want to do that, you need to change the ip adress and port that matches the other computer.

3. *webdriver.Chrome()* actually connects to the web browser with an ChromeDriver attached to it. From now on, we can perform tests and simulations in the web browser.

Function *loop_over_webpage()*

The main function then calls the function *loop_over_webpage()*.

```
def loop_over_webpage(site, browser):
    """Loops over the main page, and then every iframe """

    # 1
    findminingscript(browser, -1, site)
    print("Finished main page.")
    # 2
    nr_of_iframes = determine_number_of_iframes()
    print("Found " + str(nr_of_iframes) + " iframe(s).")
    # 3
    for i in range(nr_of_iframes):
      # 4
        print("Currently checking frame " + str(i + 1))
        findminingscript(browser, i, site)
        browser.switch_to_default_content()
    print("Finished checking iframe(s).")
```

1. This calls the function *findminingscript()*. This findminingscript() function does the actual simulation of user input. This function does not only need the browser and the target site as input, but also the frame number. In this case the main frame is equal to -1. The first iFrame is stored in an array, starting at index 0.

2. Next, we need to determine the number of *internal frames*, also known as **iFrames**. An iFrame can be seen as a webpage within a webpage.

The webdriver sees the iframes as different pages, hence we need to tell the webdriver specifically to run the simulations within the iFrames too. In order to do this, we need to determine the number of iFrames first. This function returns an integer with the number of iFrames. It is important to check the iFrames because this increases the chance of detecting cryptomining activity.

3. We want to loop over every frame and execute # 4.

4. We execute the *findminingscript()* function again, and pass the browser instance, target URL and the 'iframe number - 1' as parameters. For example, if we want to test the second iframe, we would execute *findminingscript(browser, 1, site).* The reason we need a minus one is explained in the *determine_number_of_iframes()* function. After executing the findminingscript() function, we need to switch to the main webpage. Otherwise we would end up in a recursive execution of the program. MineChecker runs for 30 seconds before it terminates. It is possible that MineChecker did not check all elements of the webpage in time. To increase the running time of MineChecker, the *-tm* flag can be used to specify the number of seconds. MineChecker does not test iframes in iframes because that would increase the running time of MineChecker significantly.

The visual representation of this process can be seen in Figure 4.8: *Stage 2 Loop over webpage.*

Function *determine_number_of_iframes()*

The function *determine_number_of_iframes()* behaves as you think it would:

```
def determine_number_of_iframes(browser):
  # 1
    return len(browser.find_elements_by_tag_name('iframe'))
```

1. The browser looks for elements with the name *iframe*. We can identify every iFrame by this keyword. The call returns an array, and we return the length of this array.

The visual representation of this function can be seen in Figure 4.8: *Stage 3: Determine number of iFrames.*

Function *findminingscript()*

Now we are going to elaborate what the *findminingscript()* function does:

```python
def findminingscript(browser, iframenr, site):
    """ Try to click elements on the page. """

    # 1
    if iframenr != -1:
        try:
            browser.switch_to_frame(iframenr)
        except Exception as e:
            print(e)
    # 2
    inputs =(browser.find_elements_by_tag_name("button")) + (
        browser.find_elements_by_tag_name("input"))
    url = browser.current_url
    # 3
    for i in range(len(inputs)):
        if browser.current_url == url:
          # 4
            try:
                inputs[i].click()
                print("Clicked on element")
                time.sleep(1)
            except Exception as e:
                print(e)
        else:
            # 5
            browser.get(url)
            if iframenr != -1:
                try:
                    browser.switch_to_frame(iframenr)
                except Exception as e:
                    print(e)
            # 6
            inputs = (browser.find_elements_by_tag_name("button"
                )) + (browser.find_elements_by_tag_name("input"))
    time.sleep(3)
```

1. If we need to check a different frame than the main frame, switch to the desired frame.

2. By executing this line of code, we get the elements of the webpage that needs to be clicked.

3. For every element in our array we need to execute #4 or #5 and #6.

4. If we are on the target page, try to click on the element.

5. If we are not on the target page, go to the page and the target frame.

6. If we go to the target page, we encounter new elements, so overwrite the elements array.

This process can be seen in Figure 4.8: Find mining script, where it interacts with the Chromium browser. This function is executed in the main page and each iFrame.

Class *MineSweeper.py*:

We have to adjust MineSweeper so that we can allow MineChecker to connect to the browser. We do this by editing *MineSweeper.py*. Within **Stage 1: Website analysis**, we change the line:

```
command = config['chrome'] + ' ' + target + \ ' --no-sandbox --
    js-flags="--dump-wasm-module --dump-wasm-module-path=' +
    outwasm + '"'
```

to:

```
  command = config['chrome'] + ' ' + target + \ ' --no-sandbox
      --remote-debugging-port=9222 --js-flags="--dump-wasm-module
       --dump-wasm-module-path=' + outwasm + '"'
```

The *–remote-debugging-port=9222* flag opens port 9222 on the local machine. Hence, we can connect MineChecker to the browser instance.

4.3.4 Evaluation

In this section, we will evaluate MineChecker's defense. We run MineSweeper over a list of websites. Then, we run MineChecker over the same list and analyse the results. We evaluate this tool by gathering data and then compare MineChecker against MineSweeper in terms of detection ratio. After this, we discuss the limitations of MineChecker and eventually we will introduce you to future work someone can work on.

Set-up

To improve the reliability of this comparison, we introduce you to my set-up first. We use a computer with an *Intel Core i7-7700K CPU @ 4.20GHz* running *VMware Workstation 14.0.0 build-6661328*. This virtual machine is running Ubuntu 18.0.4.2 LTS [11] and has 6 GB free RAM.

Data Collection

Various browser plugins use blacklists that block connections to malicious hosts. GitHub user *ZeroDot1* created a list of websites and hosts that are related to cryptomining. We downloaded two lists from *ZeroDot1.com* [22]

Total	**4124**
Duplicates	1625
Offline	1817
Irrelevant	437
Relevant	245

Table 4.2: Classification of the URLs in the merged list.

and his GitHub page [23] on the 24th of July 2019. The merged list contains 4124 websites. First, every *www.* substring was stripped from every URL in the list. That means that for example *http://www.youtube.com*, becomes *http://youtube.com.* After this, the duplicate URLs were removed from this list. We define a duplicate as a string that contains the exact same characters as another string. Hence, if we have the first URL *http://facebook.com* and the second URL *http://facebook.com*, we remove the second URL from the list. We do not remove URLs that eventually lead to the same website after a redirect, such as *http://facebook.com* and *http://fb.com.* We found 1625 duplicates, resulting in 2499 unique URLs.

Then, we ran a Python script that visits the website. If we get a status *200* code within four seconds, we know this is a valid online website. The Python code used for removing duplicates and visiting the website can be found in the Appendix of this thesis. This resulted in 1817 offline websites and 682 that are online.

We noticed that many websites were for sale or were under maintenance. To get a valid view on the comparison between MineChecker and MineSweeper, we ran a semi-automatic Python script that visits the website and checks if the website is *irrelevant.* We define irrelevant websites as websites that are: sites that do forwarding purposes, had bad host headers, domain for sale, under maintenance, running Apache or NGINX default configurations. We mark the remaining websites as relevant. The Python code used for checking these websites can be found in the Appendix of this thesis. Out of the 682 URLs, we found 245 relevant websites. An overview of the classification of the URLs in the merged list can be found in Table 4.2. Furthermore, the list of relevant URLs can be found on my GitHub page [6] and the Appendix of this thesis.

In the virtual machine, we deployed MineSweeper and MineChecker to run over the websites with a timeout of 30 seconds. After 30 seconds, the tool closes Chromium and thus we can not extract any WebAssembly module anymore. The following code was used to run the tools:

```
1  import subprocess
2  import uuid
3
4  def scansite(site):
5    filename = str(uuid.uuid4()) + ".txt"
6    outfile = open("results/"+ filename, "w")
7    outfile.write("Results for: " + site)
8    cmd = "python minesweeper.py -tm 30 -t " + site
9    p = subprocess.Popen(cmd, shell=True, stdout=outfile)
10   p.wait()
11   outfile.close()
12
13 def scansites(urls, start, stop):
14   for i in range(start, stop):
15     scansite(urls[i])
16   return True
17
18 urls = open("working_urls.txt", "r").read().split("\n")
19 if scansites(urls, 0, 245):
20   print("Finished checking urls!")
```

This piece of code executes in line 19 the *scansite()* function for every URL. In line 10, the function *p.wait()* waits for MineChecker and MineSweeper to terminate, before executing the rest of the code.

This process took place on the 31th of July 2019 and the 6th of August 2019. The extracted data contains the textual analysis conducted by MineSweeper. We have 245 files containing the analysis of the website by MineSweeper. This resulted in a total of 1.8 MB of data. The process of running MineChecker over the same list of URLs contains 245 files resulting in a total of 2.0 MB of data. All extracted data can be found on my GitHub page [6].

Data Analysis

Out of 245 visited websites, MineSweeper extracted 19 WebAssembly modules. Every WebAssembly module contained cryptomining activity. Hence, we can conclude that there are 19 out of the 245 websites (**7.76%**) mining cryptocurrency on the landing page or on at least one of the three random internal pages. MineSweeper did not classify websites as positive, where in fact they are not doing any cryptomining related activities. This means that every website marked as positive by MineSweeper, was indeed mining cryptocurrency. Hence, MineSweeper did not suffer from false positives.

MineChecker extracted 35 WebAssembly modules from the same list of URLs. MineChecker identified the same 19 cryptomining websites as MineSweeper did. However, it detected 16 more cryptomining websites. Therefore, we may conclude that MineSweeper does suffer from false negatives. These websites do not start mining when a user visits the website or on the random internal pages, but only after specific elements are clicked in the webpage. After manual inspection of the websites, we see that all of these websites explicitly ask the visitor to click a button that informs the user that it will start mining cryptocurrency. We can classify all of these websites as conscious active cryptomining websites. In total MineChecker identified 35 (**14.29%**) cryptomining websites. One may wonder why we only detected 35 cryptomining websites instead of 245. The URLs used, as described in the Data Collection section, are not updated and maintained by the creator of the list. These URLs may contain websites that changed its content in the meantime. It is also possible that websites removed the mining script because the website owner thinks that the revenue generated by its visitors is too low. Furthermore, a website is added if someone **expects** that the website is mining cryptocurrency. Therefore, we did not detect 245 cryptomining websites.

I listed my results in Table 4.3. MineSweeper had click-based interaction with the webpage that triggered cryptomining scripts. Therefore, MineChecker detected more cryptomining websites than MineSweeper.

Limitations

Due to limitations of MineSweeper, we likely missed cryptomining websites. MineSweeper spends only four seconds on three random internal pages. It is possible that websites start mining after those four seconds, resulting in a negative classification of the website.

Moreover, MineChecker only simulates click-based input on the landing page of the website. Hence, it does not interact with the three random internal pages that are visited by MineSweeper. These internal pages may also contain cryptomining scripts that remain hidden for our detection tool.

MineChecker suffers less from false negatives because it has click-based interaction with the webpage that can trigger cryptomining scripts. Although, it is still possible that MineSweeper did not detect every website. The reason for this is that we work with a thirty second time-out. After this time expired, Chromium will be closed. We configured MineChecker to click one element every second. If there were more than thirty clickable elements on the page, the remaining elements were not clicked. Hence, it was possible that cryptomining scripts were not triggered and thus not detected by our defense.

This defense does check iframes, but does not check iframes in iframes. For time and efficiency reasons, MineChecker checks one layer of iframes. Also, we visit a website only once. This means that advertisements, that contain cryptomining scripts, might not be loaded when the crawler visits the website.

Detection tool	**Websites tested**	**Number of Cryptomining Websites**	**Percentage**
MineSweeper	245	19	7.76%
MineChecker	245	35	14.29%

Table 4.3: Comparison between MineSweeper and MineChecker.

Furthermore, MineChecker relies on MineSweeper that analyses the *.wasm* file. So if MineSweeper concludes that the file contains cryptomining activities, MineChecker also classifies the website as positive.

Chapter 5

Conclusions

In this thesis, we looked into detecting cryptomining websites. We showed why the current mechanisms for detecting such websites are insufficient because of false negatives. We looked into how we can detect a cryptomining website that starts mining after click-based user input. Therefore we proposed MineChecker, an extension of MineSweeper. This tool simulates user input into a web browser to trigger hidden mining scripts. This tool allows the underlying tool MineSweeper to analyse the fundamental operations computed by the CPU in order to detect cryptocurrency mining.

We compared MineChecker against the underlying tool MineSweeper. We ran both tools against a blacklist of URLs. MineSweeper extracted and analysed the WebAssembly modules. MineChecker's approach flagged more websites as cryptomining websites than MineSweeper did. Therefore MineChecker suffers less from false negatives, as described in section *Limitations*.

5.1 Future work

This tool can use some improvement. MineChecker clicks elements in the webpages of the specified URLs. But we did not perform click-based actions on the three random internal pages. Therefore, one can improve the tool by configuring MineSweeper to execute MineChecker over the internal pages. MineChecker only needs small adjustments because it is already programmed in a way that it works for every webpage.

Our approach only used a list of over 4000 URLs. In the future, one can study the prevalence and profitability of cryptomining websites in Alexa's top 1 million list.

Bibliography

[1] Gong Chen. *Improved Security for Digital Advertising Ecosystems.* PhD thesis, Georgia Institute of Technology, 2018.

[2] Chang-Hoan Cho, Jung-Gyo Lee, and Marye Tharp. Different forced-exposure levels to banner advertisements. *Journal of advertising research*, 41(4):45–56, 2001.

[3] Catalin Cimpanu. Firefox Working on Protection Against In-Browser Cryptojacking Scripts. `https://www.bleepingcomputer.com/news/software/firefox-working-on-protection-against-in-browser-cryptojacking-scripts/`, 2018. [Online; accessed 16-April-2019].

[4] Google. ChromeDriver. `http://chromedriver.chromium.org/`, 2019. [Online; accessed 22-July-2019].

[5] Google. Chromium. `https://www.chromium.org/Home`, 2019. [Online; accessed 22-July-2019].

[6] Justin Hende. GitHub. `http://github.com/BierBrigadier/MineChecker`, 2019. [Online; accessed 06-August-2019].

[7] Amin Kharraz, Zane Ma, Paul Murley, Charles Lever, Joshua Mason, Andrew Miller, Nikita Borisov, Manos Antonakakis, and Michael Bailey. Outguard: Detecting in-browser covert cryptocurrency mining in the wild. In *The World Wide Web Conference*, pages 840–852. ACM, 2019.

[8] Radhesh Krishnan Konoth, Emanuele Vineti, Veelasha Moonsamy, Martina Lindorfer, Christopher Kruegel, Herbert Bos, and Giovanni Vigna. Minesweeper: An in-depth look into drive-by cryptocurrency mining and its defense. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1714–1730. ACM, 2018.

[9] Richard Lawson. *Web scraping with Python.* Packt Publishing Ltd, 2015.

[10] Ritu Lohtia, Naveen Donthu, and Edmund K Hershberger. The impact of content and design elements on banner advertising click-through rates. *Journal of advertising Research*, 43(4):410–418, 2003.

[11] Canonical Ltd. Ubuntu 18.0.4.2. `https://ubuntu.com/download/desktop`, 2019. [Online; accessed 08-March-2019].

[12] Robert S Moore, Claire Allison Stammerjohan, and Robin A Coulter. Banner advertiser-web site context congruity and color effects on attention and attitudes. *Journal of advertising*, 34(2):71–84, 2005.

[13] Satoshi Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system. 2008.

[14] Panagiotis Papadopoulos, Panagiotis Ilia, and Evangelos P Markatos. Truth in web mining: Measuring the profitability and cost of cryptominers as a web monetization model. *arXiv preprint arXiv:1806.01994*, 2018.

[15] Harry Percival. *Test-driven development with Python: obey the testing goat: using Django, Selenium, and JavaScript.* " O'Reilly Media, Inc.", 2014.

[16] radheshkrishnan. GitHub -MineSweeper. `https://github.com/vusec/minesweeper`, 2018. [Online; accessed 10-May-2019].

[17] Rosnisa Abdull Razak and Fairul Rizal Fahrurazi. Agile testing with selenium. In *2011 Malaysian Conference in Software Engineering*, pages 217–219. IEEE, 2011.

[18] Muhammad Saad, Aminollah Khormali, and Aziz Mohaisen. End-to-end analysis of in-browser cryptojacking. *arXiv preprint arXiv:1809.02152*, 2018.

[19] Mark Scott. Use of Ad-Blocking Software Rises by 30% Worldwide. `https://www.nytimes.com/2017/01/31/technology/ad-blocking-internet.html`, 2017. [Online; accessed 16-April-2019].

[20] Badges2Go UG. CoinHive. `https://coinhive.com`, 2017. [Online; accessed 06-April-2019].

[21] W3C. WebAssembly. `https://webassembly.org/`, 2017. [Online; accessed 16-April-2019].

[22] ZeroDot1. CoinBlockerLists. `https://zerodot1.gitlab.io/CoinBlockerListsWeb/downloads.html`, 2019. [Online; accessed 22-July-2019].

[23] ZeroDot1. GitHub - CoinBlockerLists. `https://github.com/Ultimate-Hosts-Blacklist/ZeroDot1_CoinBlockerLists_browser/blob/master/domains.list`, 2019. [Online; accessed 22-July-2019].

Appendix A

Appendix

A.1 URL checker

This piece of code we remove the duplicate URLs in a list. After that, we visit the website and determine if we get a status 200 code, which means the website is online. We add a timeout of four seconds so that we remove URLs that take longer to load.

```python
import urllib2
import socket

def check_url(url, timeout=4):
    try:
        return urllib2.urlopen(url,timeout=timeout).getcode() == 200
    except urllib2.URLError as e:
        return False
    except socket.timeout as e:
        return False
    except Exception as e:
    return False


counter = 0
temp = open("urls.txt", "r").read().split("\n")
working_urls2 = open("working_urls.txt", "a")
working_urls = []

for i in temp:
  if i not in working_urls:
    working_urls.append(i)
  else:
    counter = counter + 1
print("Removed " + str(counter) + " duplicates!")

for i in range(len(working_urls)):
  site = working_urls[i]
  if check_url(site):
    working_urls2.write(site + "\n")


working_urls2.close()
```

A.2 Relevant URL checker

Within this piece of code we open the websites that provided us a status 200 code already. Then we manually determine if this website is relevant to our research.

```
import webbrowser

working_urls = open("working_urls.txt", "r").read().split("\n")
good_urls = open("good_urls.txt", "a")
chrome_path = '/usr/bin/chromium-browser %s'

for i in range(len(working_urls)):
  webbrowser.open(working_urls[i])
  judgement = input("Is this site relevant? Yes (1) or No(2)?")
  if str(judgement) == "1":
    good_urls.write(working_urls[i] + "\n")
  else:
    print("Not a possible mining site. Skipping...")
```

A.3 URLs used by MineSweeper & MineChecker

MineSweeper and MineChecker executed the tool using the following list of URLs:

http://50bots.nullrefexcep.com
http://8000plus.si
http://a.proxy4.nullrefexcep.com
http://a.proxy5.nullrefexcep.com
http://aba.ae
http://ad-miner.com
http://ad.g-content.bid
http://adbtc.top
http://aeon.crypto-webminer.com
http://aeon.hashvault.pro
http://ajcryptominer.com
http://ajplugins.com
http://aleinvest.xyz
http://andlache.com
http://andrew.nullrefexcep.com
http://anime.reactor.cc
http://anybest.space
http://apdrive.win
http://api.coin-hive.com
http://api.jsecoin.com
http://api.monerise.com
http://apib.monerise.com
http://app.pr0gramm.com
http://aqqgli3vle.bid
http://authedwebmine.cz
http://averoconnector.com
http://avualrhg9p.bid
http://b.proxy4.nullrefexcep.com
http://besocial.online
http://bestsecurepractice.com
http://beta.coinblind.com
http://bizoninvest.com
http://blockchain.jsecoin.com
http://blox.minexmr.com
http://browsermine.com
http://browsersurf.12finance.com
http://butcalve.com
http://bytecoin.crypto-webminer.com
http://ca.minexmr.com
http://captain2.directprimal.com
http://cfcd.duckdns.org
http://code.ws
http://cody.services
http://coin-have.com
http://coin-hive-stratum-bqywuwxwij.now.sh
http://coinjive.com
http://coinminingonline.com
http://coinnebula.com
http://coinwebmining.com
http://coinworker.com
http://conhive.com
http://cookiescript.info
http://crypto-webminer.com
http://crypto.csgocpu.com
http://cryptosearch.site
http://cryptotab.net
http://csgocpu.com
http://custom.crypto-webminer.com
http://dashboard.inwemo.com
http://datasecu.download
http://de.cookiescript.info
http://de.moneroocean.stream
http://dero.crypto-webminer.com
http://developer.jsecoin.com
http://dinastycoin.crypto-webminer.com

http://dmdamedia.hu
http://dmg.crypto.csgocpu.com
http://donate.crypto-webminer.com
http://ebd.cda.pl
http://electroneum.crypto-webminer.com
http://electroneum.hashvault.pro
http://electroneum.monerise.com
http://elthamely.com
http://escobar.pr0gramm.com
http://etnpool.minekitten.io
http://external.monerise.com
http://fili.cc
http://fortrader.ru
http://frankfurt-1.xmrpool.net
http://freebitco.in
http://fxnow.ru
http://g-content.bid
http://gay-hotvideo.net
http://gemius.pl
http://global.crypto.csgocpu.com
http://goxmrminer.com
http://graft.crypto-webminer.com
http://gridiogrid.com
http://gulf.moneroocean.stream
http://habd.as
http://harvest.surge.sh
http://hashforcash.us
http://hashvault.pro
http://hit.gemius.pl
http://httpp.gdn
http://iaheyftbsn.review
http://intellecthosting.net
http://intense.hashvault.pro
http://intensecoin.crypto-webminer.com
http://intucoin.crypto-webminer.com
http://ipbc.crypto-webminer.com
http://ipv6.freebitco.in
http://js.nahnoji.cz
http://jsecoin.com
http://jurtym.cf
http://jwduahujge.ru
http://kedtise.com
http://krb.devphp.org.ua
http://kunay.nullrefexcep.com
http://l-shop.nullrefexcep.com
http://lan.datasecu.download
http://ledinund.com
http://lib.rus.ec
http://light.browsermine.com
http://lightminer.co
http://listat.biz
http://login.browsermine.com
http://losital.ru
http://manager.browsermine.com
http://masari.crypto-webminer.com
http://mebablo.com
http://mepirtedic.com
http://mine.nahnoji.cz
http://mine.torrent.pw
http://mine.xmrpool.net
http://minekitten.io
http://miner.nimiq.com
http://minero.cc
http://minescripts.info
http://minexmr.com
http://mining.freebitco.in
http://minr.nullrefexcep.com
http://mmpool.org
http://mollnia.com
http://monad.network
http://monero-miner.com
http://monero-miner.net
http://monero.crypto-pool.fr
http://monero.crypto-webminer.com
http://monero.hashvault.pro
http://monero.monerise.com
http://morerominer.rocks
http://mwor.gq
http://nadjibsoft.blogspot.com
http://nahnoji.cz
http://nathetsof.com
http://nerdorium.org
http://nerohut.com
http://new.browsermine.com
http://nexttime.ovh
http://nfwebminer.com
http://niematego.tk
http://nimiq.watch
http://ninaning.com
http://node.nimiqpool.com
http://notmining.org
http://novaminers.tk
http://nullrefexcep.com
http://object.de
http://ocean.directprimal.com
http://oei1.gq
http://old.browsermine.com
http://p.estream.to
http://phx-4.xmrpool.net
http://platform.jsecoin.com
http://play.es
http://play.stream.vidzi.tv
http://play.streaming.estream.to
http://play.intellecthosting.net
http://play5.flashx.pw
http://pool-de.supportxmr.com
http://pool.aeon.hashvault.pro
http://pool.graft.hashvault.pro
http://pool.minexmr.com
http://pool.supportxmr.com
http://proxy-ajcryptominer.ajplugins.com
http://proxy.nullrefexcep.com
http://proxy2.nullrefexcep.com
http://proxy3.nullrefexcep.com
http://proxy5.nullrefexcep.com
http://proxy7.nullrefexcep.com
http://proxy8.nullrefexcep.com
http://proxy9.nullrefexcep.com
http://px2.papoto.com
http://qwertycoin.crypto-webminer.com
http://rand.com.ru
http://rapidvideo.com
http://ratingtoplist.com
http://renhertfo.com
http://rock.directprimal.com
http://ron.si
http://rove.cl
http://rtw.monerise.com
http://s3.minexmr.com
http://sass.directprimal.com
http://sass2.authcaptcha.com
http://sass2.directprimal.com
http://seminarski-diplomski.co.rs
http://server.jsecoin.com
http://sharing-is-caring.info
http://silver.crypto.csgocpu.com
http://site.coinnebula.com
http://smurf.crypto.csgocpu.com
http://sparnove.com
http://srv02.bitcoiner.win
http://static-02.flu.cc
http://static.freebitco.in
http://static1.freebitco.in
http://static2.freebitco.in
http://static3.freebitco.in
http://status.minexmr.com
http://stellite.crypto-webminer.com
http://stream.nullrefexcep.com
http://sumo.hashvault.pro
http://sumokoin.crypto-webminer.com
http://sumokoin.hashvault.pro
http://superiorcoin.crypto-webminer.com
http://supportxmr.com
http://techtricksworld.com
http://thepiratebay.cr
http://traffic-tech-service.info
http://turnsocial.com
http://turtlecoin.crypto-webminer.com
http://update-your-pc.info
http://us.moneroocean.stream
http://vegas-1.xmrpool.net
http://verresof.com
http://vkcdnservice.com
http://webmine.cz
http://webminer.pro
http://webmining.co
http://ultranote.crypto-webminer.com
http://webwidgetz.duckdns.org
http://ws-fr-01.rocks.io
http://wss.rand.com.ru
http://wtm.monitoringservice.co
http://xmrpool.net
http://xy.nullrefexcep.com
http://youporn.sexy
http://www49.playercdn.net
http://xmr.omine.org
http://12finance.com
http://ftp0118.info
http://marketgid.com
http://zaloapp.com