Malicious Mining Program in Browser Survey

謝秉昂
電機四 B03901195

楊書文
資工四 B03902130

黃楷傑
資工四 B03611034

陳致維
電機三 B04901165

ABSTRACT (5%)

Cryptominig is a new way to profit since the rising of cryptocurrency. However, cryptomining consumes a lot of computing resourses and it's getting harder to profit as an individual miner. To gather more computing power, attackers "borrow" the victims' web browsing devices' computing power by embeding mining program in the websites they are browsing. Existing defensing approach is mainly carried out by a browser extenstions. What they do includes: A. Blocking a list of URLs to known servers of the mining program provider and the proxy servers of the mining pools. B. Check the source code of the website to see if there are certain keywords that indicates it is a mining code. C. Check CPU usage to see if CPU activity is unreasonably high. However, above approaches cannot block attackers that manage to maintain their own server and proxy. We propose a new method called "CodeDist" which compares the distance between well known mining codes and currently browsing website using Abstract Syntax Tree (AST). If the similarity passes a threshold, we assume it is mining. Our method is better than keyword detection since it detects the code pattern instead and better than URL blacklisting since this whole detection process is done before users' devices connecting to other servers and proxies. However, CodeDist is not able to detect mining code that is entirely rewritten or designed.

KEYWORDS
Cryptomining, Cryptojacking, CPU, Abstract Syntax Tree

1 INTRODUCTION (15%)

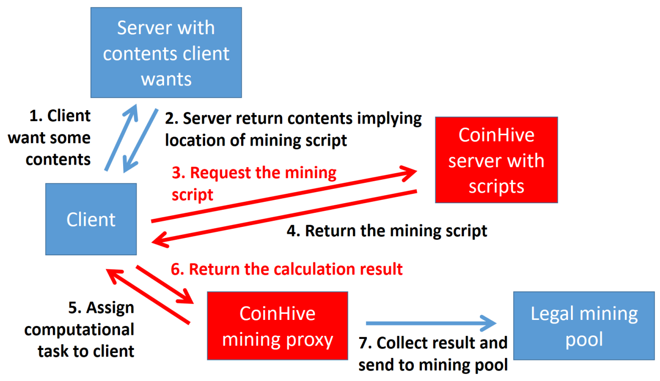
Cryptomining is a new way to profit since the rising of cryptocurrency. To successful profit from mining, miners have to provide their devices' computing power. However, as many more miners joining, due to the nature of cryptomining, it is much more difficult to profit as an individual miner that only have access to limited computing power. A mining pool in the context of cryptocurrency mining, is a way that multiple miners share their processing power over a network and split the reward according to the amount of work one has contributed.
A new type of cyber attack is one trying to steal other's computing resources and make the victims' contribution under the name of the attacker. Web mining is an approach when content providers embed mining code in their websites that 'borrows' some computing resources from the clients' machine to make some profit. This is not a problem as long as their viewers are notified of the situation and the amount of resources they use are reasonable. Mining code providers such as Coinhive claims that the original purpose of their service is to provide an alternative to embedded ads. The problem arises when an attacker didn't notify the clients and 'over-borrows' their resources. This is really easy to happen since weather or not to notify the user is not enforced in the mining code.
Existing methods to defense these kind of attacks are mostly implemented as web browser extensions. What the extensions do includes: A. Blocking a list of URLs to known servers of the mining program provider and the proxy server of the mining pools. B. Check the source code of the website to see if there are certain keywords that indicates it is a mining code. C. Check CPU usage to see if CPU activity is unreasonably high. However, these approaches cannot block attackers that manage to maintain their own server and proxy servers.
In this article, we are proposing a new method to defense this kind of attack. We name it "CodeDist". How it works is we make use of the Abstract Syntax Tree, which is a way to represent the structure of a source code, regardless of those factors that doesn't affect how the actual program works such as change of variable names. We then calculate a

relative distance between the web code we are examining and some well-known mining code. If the distance is smaller than a threshold, then we assume the target embedded with mining codes.
Since our method attempts to block mining website before it sends and receive from external servers and proxies and even before the code is actually executed. Our main contribution is to bring a new research direction on cryptojacking defense.

2 Problem Definition(10%)

2-1 Problem Description

We illustrate the process of cryptojacking by the following figure. First, when the victim visited a website that was compromise by the attacker or the website itself wants to earn money by mining on its visitors, the website will response with the html file inferring the malicious server address where the malicious javascript (mining script) stored. And, victim will request such script from the malicious server (Step 3). Then, after receiving the malicious javascript file (Step 4), the victim will start to do mining action and submit the result of mining back to mining pool (Step 7). Some other mining API might adopt the procedure described above, which is simpler than mainstream. Mainstream of mining API such as CoinHive adopt an additional step to share the profit with the website by asking clients to communicate with a proxy set up by coinHive (done by mining script, Step 5 & 6). Such proxy will assign each clients different tasks, accumulate the calculation result, and send the result to any legal mining pool in the name of CoinHive. After gaining profit, CoinHive will finally share the profit with websites using its mining scripts.


Server with contents client wants
1. Client want some contents
2. Server return contents implying location of mining script
CoinHive server with scripts
3. Request the mining script
Client
4. Return the mining script
6. Return the calculation result
5. Assign computational task to client
CoinHive mining proxy
7. Collect result and send to mining pool
Legal mining pool

2-2 Threat Model

The threat model in our scenario has two parts - what we can defense and what we cannot defense. In the "what we can defense" part, we have two assumptions. One, if the attacker can build their own server. Second, if the server alter the function name or variable name. In the "what we cannot defense" part, we also have two assumptions. One, if the attacker drastically changes the code structure. Second, if the attacker writes their own mining code.

3 Related Work (10%)

3.1 Existing Defense Approach

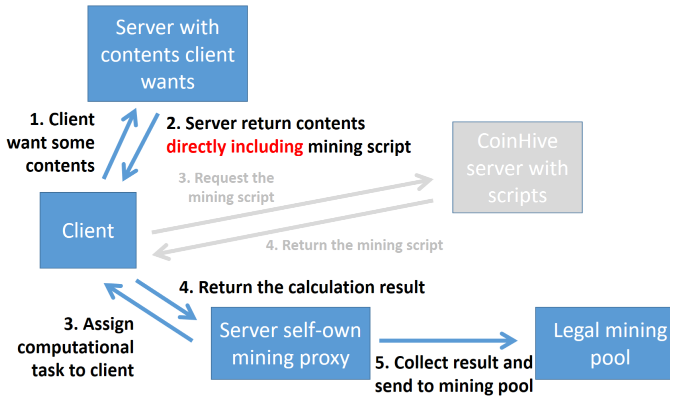
Nowadays, most defense against malicious cryptojacking are done by installing extension on the browser. And its approach can mainly classified into two types. Firstly, examine the connection of requests and responses. Secondly, see what codes are actually executing. So, in the following paragraphs, we will talk how the traditional approaches defend in detail.

3.2 Blocking of connection

We illustrate the process of cryptojacking by the following figure. First, when the victim visited a website that was compromise by the attacker. The website will response with the html file containing the malicious server address where the malicious javascript stored. And, victim will request the file from the malicious server (Step 3). Then, after receiving the malicious javascript file (Step 4), the victim will start to do mining action and submit the result of mining back to mining pool (Step 5 & 6).

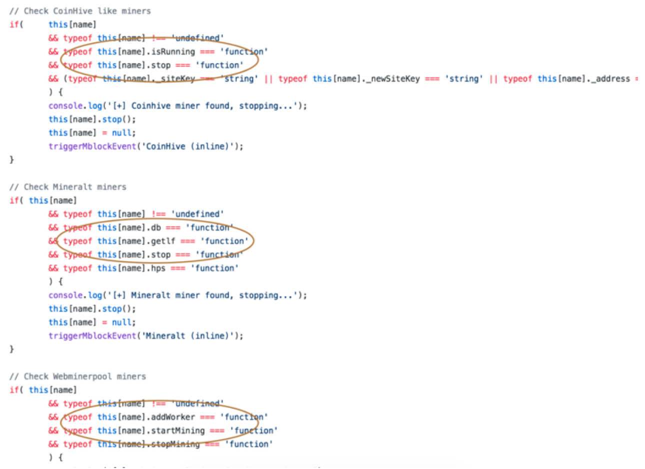
Step 3 is where the existing approach can do defense action. And the principle of this way is to block the compromised server URL. In order to do this, existing approach maintain a database. By maintain this database, every time we request a URL, we compare it with the malicious website in the database. If matching, it means that the place we want to go is malicious, so we abort that connection. Therefore, we will not get the malicious code to do mining action.

But this approach has some flaws inherently. First, cost to maintain that database. With the time going, this database will grow, and the overhead will gradually become a problem if the size of database is too big. Secondly, it's easily to be bypassed if the attacker built their own server and mining proxy. We illustrate this by the figure under this paragraph. In this condition, the original defense approach need some time to include the new malicious website URL in the database. Hence, this time interval can be a dangerous duration that can be exploited by the attacker. Besides, the cost of building a self-owned mining proxy is an easy job that can be done quickly.


Server with contents client wants
1. Client want some contents
2. Server return contents directly including mining script
CoinHive server with scripts
3. Request the mining script
Client
4. Return the mining script
4. Return the calculation result
3. Assign computational task to client
Server self-own mining proxy
5. Collect result and send to mining pool
Legal mining pool

3-3 Source code checking

This approach collects every javascript file that sent by the server and check if these files contained some mining action.



```
// Check CoinHive like miners
if(     this[name]
        && typeof this[name] !== 'undefined'
        && typeof this[name].isRunning === 'function'
        && typeof this[name].stop === 'function'
        && (typeof this[name]._siteKey === 'string' || typeof this[name]._newSiteKey === 'string' || typeof this[name]._address =
        ) {
        console.log('[+] Coinhive miner found, stopping...');
        this[name].stop();
        this[name] = null;
        triggerMblockEvent('CoinHive (inline)');
}

// Check Mineralt miners
if( this[name]
        && typeof this[name] !== 'undefined'
        && typeof this[name].db === 'function'
        && typeof this[name].getlf === 'function'
        && typeof this[name].stop === 'function'
        && typeof this[name].hps === 'function'
        ) {
        console.log('[+] Mineralt miner found, stopping...');
        this[name].stop();
        this[name] = null;
        triggerMblockEvent('Mineralt (inline)');
}

// Check Webminerpool miners
if( this[name]
        && typeof this[name] !== 'undefined'
        && typeof this[name].addWorker === 'function'
        && typeof this[name].startMining === 'function'
        && typeof this[name].stopMining === 'function'
        ) {
```

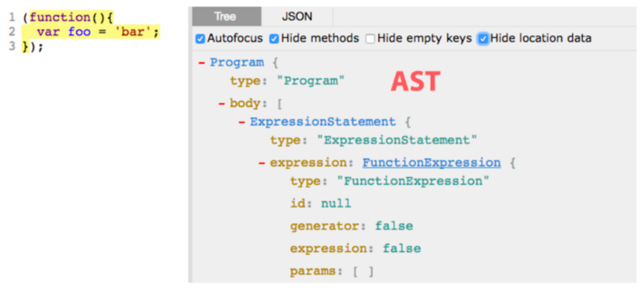
This approach is very effective because most attacker just use mining from the Coinhive or other cryptojacking service. However, this way is not smart. The reason is that the checking mechanism is too naïve and fixed. It just checks the existence of some function name or class type of javascript.

At this point, we may raise two questions. One. what if attackers just change function name or class type. Second, what if attacker drastically alter the structure of the mining code and let look like a totally different codes that we cannot detect by using the tradition method.

4 Result(50%)

4.1 Abstract Syntax Tree

It is concept that will be used in our approach, its main idea is that we can represent every in a tree form. For example.



```
(function(){
  var foo = 'bar';
});
```

☑ Autofocus ☑ Hide methods ☐ Hide empty keys ☑ Hide location data

```
- Program {
    type: "Program"
  - body: [
    - ExpressionStatement {
        type: "ExpressionStatement"
      - expression: FunctionExpression {
          type: "FunctionExpression"
          id: null
          generator: false
          expression: false
          params: [ ]
```

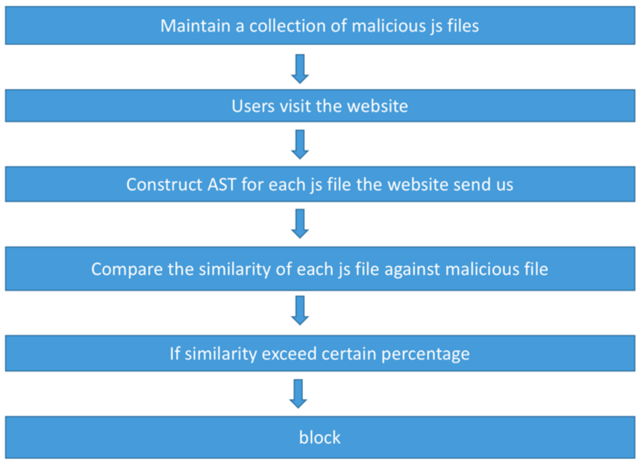
4-2 Our Approach

Simply say, the approach we adopt has two steps. First step – we construct abstract syntax tree for every code that we received. Second step – we compare the similarity by using abstract syntax tree to judge if this code is malicious code or not. In the next paragraph, we will explain it in detail.

Our approach has some preparation, we have to collect the possible attacker mining codes. And this is not a very hard thing to do. Because the malicious code that are existing on the market are not many. By collecting the malicious codes, we build the abstract syntax tree for each.

After preparation, we can use this work as a tool to help us to judge if the codes we receive are malicious or not. And we compare the abstract syntax tree of the code we received with the abstract syntax tree of the database we maintained. By comparing them, we will get a number (***S***) that represents the similarity of code we received and the malicious code. Because the final result we want is not just a number, what we want is the True/False result that helps us to block it or not accept it. To do this, we need some rule, so we set up a threshold. This threshold (***T***) is a number that lie between zero to one. If ***S*** is bigger than ***T***, then we block it, otherwise.

The following picture is a summary of our approach


Maintain a collection of malicious js files
Users visit the website
Construct AST for each js file the website send us
Compare the similarity of each js file against malicious file
If similarity exceed certain percentage
block

4.3 Analysis & Discussion

But, this method has some overhead we need to consider. For example, the comparison is a process that takes some time. Therefore, it may affect the loading speed when we visited the webpage. To avoid this problem, we let the codes we receive execute first. Then we do our comparison in the background. By doing comparison "asynchronously" rather than "synchronously", we will not affect the loading speed of webpage. Hence, the user experience will not be harmed. And the mining action will just take few seconds that is not a big problem. So, we strike a balance between security and usability at the same time.

The advantage that our approach have is flexibility. Because if the attacker wants to bypass our defense, they need to write their own mining codes. And it's not an easy job. Therefore, we enhance the difficulty of cryptojacking. Secondly, we can also defer the attack scenario that if the attacker that build their server. This attack scenario is always be a tedious and unsolvable job for existing method.

5 Conclusion & the future work(5%)

5.1 conclusion

Even though the existing defense approach can block the most cryptojacking easily, but actually it is easy to bypass if the attacker just glimpse the source code of the existing defense approach. So what the work we have done is to let the attacker take more effort to come up with a new attack way or directly write a totally differently mining code. To sum up in a word, we make the cryptojacking becomes more hard.

5.2 future

Unfortunately, we have to admit that our approach cannot block all malicious code. The reason is that we still not detect the core property that only the ctptojacking have. We tried to find out but fail, because it involves the more things that related to binary or low – level computer knowledge. In addition, due to the approached we implemented is through the extension of the browser. So, if we want to know more about if the webpage is malicious or not, we need to get more information about victim's low-level component activity. But it is prohibited in the normal case

REFERENCES(5%)

[1]https://twitter.com/bad_packets/status/940333744035999744
[2]https://blog.techbridge.cc/2016/07/16/javascript-jsfuck-and-aaencode/
[3]https://github.com/keraf/NoCoin
[4]https://www.theverge.com/2012/4/7/2931600/hotel-caught-injecting-advertising-into-web-pages-on-complimentary-wi
[5]https://www.ithome.com.tw/news/120941
https://blog.gtwang.org/web-development/coinhive-website-crypto-miner-tutorial/
[6]https://blog.sucuri.net/2017/12/cloudflare-solutions-keylogger-on-thousands-of-infected-wordpress-sites.html
[7]https://www.ithome.com.tw/news/119243
[8]https://www.ithome.com.tw/news/120028
[9]https://blog.trendmicro.com/trendlabs-security-intelligence/malvertising-campaign-abuses-googles-doubleclick-to-deliver-cryptocurrency-miners/