This article has been accepted for publication in IEEE Access. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3204814

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.
Digital Object Identifier 10.1109/ACCESS.2017.DOI

CIRCUIT: A JavaScript Memory Heap-Based Approach for Precisely Detecting Cryptojacking Websites

HYUNJI HONG†, SEUNGHOON WOO†, SUNGHAN PARK†, JEONGWOOK LEE, AND HEEJO LEE
Department of Computer Science and Engineering, Korea University, Seoul 02841, Korea
(e-mail: {hyunji_hong, seunghoonwoo, sunghan-park, wjddnrdl65, heejo}@korea.ac.kr)
†: These authors contributed equally to this work.
Corresponding author: Heejo Lee (e-mail: heejo@korea.ac.kr).

This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2019-0-01697 Development of Automated Vulnerability Discovery Technologies for Blockchain Platform Security, No.2022-0-00277 Development of SBOM Technologies for Securing Software Supply Chains, No.2022-0-01198 Convergence Security Core Talent Training Business, and No.IITP-2022-2020-0-01819 ICT Creative Consilience program).

ABSTRACT Cryptojacking is often used by attackers as a means of gaining profits by exploiting users' resources without their consent, despite the anticipated positive effect of browser-based cryptomining. Previous approaches have attempted to detect cryptojacking websites, but they have the following limitations: (1) they failed to detect several cryptojacking websites either because of their evasion techniques or because they cannot detect JavaScript-based cryptojacking and (2) they yielded several false alarms by focusing only on limited characteristics of cryptojacking, such as counting computer resources. In this paper, we propose CIRCUIT, a precise approach for detecting cryptojacking websites. We primarily focuse on the JavaScript memory heap, which is resilient to script code obfuscation and provides information about the objects declared in the script code and their reference relations. We then extract a reference flow that can represent the script code behavior of the website from the JavaScript memory heap. Hence, CIRCUIT determines that a website is running cryptojacking if it contains a reference flow for cryptojacking. In our experiments, we found 1,813 real-world cryptojacking websites among 300K popular websites. Moreover, we provided new insights into cryptojacking by modeling the identified evasion techniques and considering the fact that characteristics of cryptojacking websites now appear on normal websites as well.

INDEX TERMS Browser Security; Web Security; Cryptojacking

I. INTRODUCTION

Cryptojacking is a well-known cyberattack that applies victims' computer resources (*e.g.*, CPU and memory) for cryptocurrency mining without the consent of the victims. The cryptocurrency that is generated during the mining process can be hijacked by attackers for profit. Previously, cryptojacking was executed by inducing users to execute malicious programs, similar to existing malicious attacks, *e.g.*, trojan and ransomware attacks. Recently, however, the more threatening cryptojacking has appeared, which has been implemented based on the modern web environment, and the malicious script code of cryptojacking is automatically executed on the client side when a user visits a cryptojacking website.

Hence, detecting cryptojacking websites and filtering them out in the web environment is crucial for protecting user resources. However, the precise detection of cryptojacking websites is complex and prone to errors. As script code obfuscation techniques are frequently applied to cryptojacking websites, it is increasingly growing more challenging to detect cryptojacking based on the static analysis approach. Furthermore, the cryptojacking websites' characteristics (*e.g.*, running numerous threads or consuming high resources of victims' computers) now appear on various normal websites (*e.g.*, live-streaming websites), thereby complicating the precise detection of cryptojacking websites.

Existing cryptojacking detection approaches mainly use the following four techniques: blacklisting-based [14], [16],

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Access. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3204814

[25], [37], [42], [45], resource monitoring-based [38], [40], thread count-based [38], [41], and WebAssembly-based techniques [35], [42]. Although they all provide insights into detecting cryptojacking, they have limitations in terms of the precise detection of cryptojacking. Blacklisting-based approaches stored the characteristics appearing on cryptojacking websites as blacklists (*e.g.*, domain, script code, and external server link) and determined a website that contains the stored blacklists as cryptojacking websites. However, these approaches failed to detect several cryptojacking websites because they can be easily bypassed by simple evasion techniques, such as script code obfuscation or a domain generation algorithm (DGA), which steadily changes the external server link. By contrast, resource monitoring-based and thread count-based approaches, which focus on cryptojacking requiring several computer resources and threads, respectively, yield numerous false positives because even recent normal websites, require several threads and high computer resource consumption (*e.g.*, web-game or streaming sites). Last, the WebAssembly-based approaches exhibit low detection coverage because they cannot detect the most common JavaScript-based cryptojacking websites.

Our approach. In this study, we propose CIRCUIT, a precise approach for detecting cryptojacking websites. We define a unit called a *reference flow*, which represents cryptojacking behavior and is robust against JavaScript code obfuscation, and use it to detect cryptojacking websites.

We mainly focus on the JavaScript memory heap of the websites. The memory heap reveals the declared objects in the website's script code and their reference relations. To execute cryptojacking, a website should run multiple threads using web workers (see Section II-A). Hence, CIRCUIT extracts the behavior of each thread separately from the heap graph, which is called the *reference flow*. Subsequently, CIRCUIT extracts all reference flows from known cryptojacking websites, stores them as the signature of cryptojacking, and compares all reference flows of the target website with the stored cryptojacking signatures. If at least one reference flow of the target website is similar to the cryptojacking signature, then the website is identified as a cryptojacking website. As we focused on the memory heap, CIRCUIT can robustly detect cryptojacking websites even with an obfuscated script code. In addition, CIRCUIT can detect cryptojacking websites more precisely than existing approaches by detecting the reference flow containing the actual cryptojacking behavior rather than simply focusing on the characteristics of several threads or high resource consumption, commonly appearing on normal websites.

Evaluation. For the experiment, we collected over 300K real-world websites, including the Alexa top 100K and Majestic top 200K websites. Among them, CIRCUIT detected 1,813 cryptojacking websites with cryptojacking behaviors, most of which used evasion techniques to avoid cryptojacking detection. CIRCUIT responded flexibly to evasion techniques in four categories based on the evasion techniques modeled in the experiment. Furthermore, by analyzing the distribution of the number of threads of the collected websites, we demonstrated the limitations of the existing resource monitoring and thread-count-based approaches and proved the efficiency of CIRCUIT from the perspective of precise cryptojacking detection (see Section IV).

Contributions. We summarize our contributions below:

- We propose CIRCUIT, a precise approach for detecting cryptojacking websites based on the JavaScript memory heap. CIRCUIT is robust to evasion techniques applied to cryptojacking websites to avoid cryptojacking detection.
- Although evasion techniques were applied to most of the identified cryptojacking websites, CIRCUIT succeeded in detecting 1,813 cryptojacking websites from 300K real-world websites.
- Modeling evasion techniques to avoid cryptojacking detection allows us to provide new insights into cryptojacking as behaviors previously associated with cryptojacking now appear widely on normal websites.

II. BACKGROUND AND RELATED WORK

This section describes the background knowledge related to cryptojacking (Section II-A) and introduces related works on cryptojacking detection (Section II-B).

A. BACKGROUND AND TERMINOLOGY

1) CRYPTOCURRENCY MINING

Cryptocurrency is a digital asset designed to function as a medium of exchange. *Cryptocurrency mining* (*cryptomining*) is the process of validating a cryptocurrency transactions. To gain cryptocurrencies (*e.g.*, Bitcoin and Ethereum), *Proof-of-Work (PoW)* is performed, which is a blockchain consensus mechanism. In a nutshell, peers (*i.e.*, miners) in the PoW blockchain network solve complex mathematical problems with taxing computational power. The fixed time (*e.g.*, 10 minutes for Bitcoin) rewards (*i.e.*, cryptocurrency) a peer who wins the race and mines the block. Mining is computationally taxing because only the first miner who solves the problem is rewarded. To strengthen the probability of finding a block, miners combine their computational resources through public mining pools.

2) CRYPTOJACKING

Cryptojacking refers to the malicious behavior that intercepts all profits arising cryptomining by using the visitors' resources in a web environment, without their consent. When visiting a website injected with cryptomining, a user's computational resources are hijacked to mine cryptocurrency. Specifically, the web technology evolution, such as JavaScript (JS) and WebAssembly (Wasm), makes it easy to access users' resources and leverage them in the mining process; simply inserting the JavaScript code that supports mining services into the web page can infect website visitors.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Access. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3204814

Moreover, since the cryptojacking code executes automatically and works as a background on the webpage, visitors hardly realize that they are infected. Figure 1 shows the workflow of the cryptojacking process.

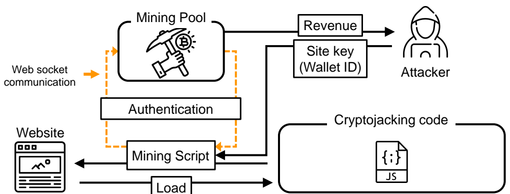


FIGURE 1: Overview of cryptojacking process.

Cryptojacking is executed in the following three steps:

1) **Executing cryptojacking code on a website.** When a user visits a website, the web browser automatically loads the website code files (*e.g.*, necessary libraries and `external resources`) and executes them. As the cryptocurrency script code was previously inserted in the website, it is also executed in this step.
2) **Participating in a mining pool.** The executed cryptojacking code authenticates the visitor's PC by using a predefined mining pool. Thereafter, the visitor (*i.e.*, victim) is forced to participate in the mining pool, organized to mine cryptocurrency.
3) **Mining and gaining profits.** The computer resources of the victim's PC mine the cryptocurrency, and then the mined cryptocurrency is sent to the attacker's digital wallet address, which was previously defined in the cryptojacking code of the website.

Unlike traditional malware, cryptojacking exploits only the victim's computer resources; the victim has a minor infection symptoms, such as slow computer performance or an increase in power consumption, making it difficult to recognize cryptojacking. Furthermore, since cryptojacking runs in a web environment, its execution is less restrictive, and various devices and operating systems may be exposed to cryptojacking. Therefore, cryptojacking has attracted attention as a stable and continuous means of profit for attackers.

3) JAVASCRIPT ENGINE

The workflow of the JavaScript engine, where the cryptojacking code is executed, is shown in Figure 2. The JavaScript engine first analyzes the syntax errors of the script code, and if there are no errors, it starts reading the script code from top to bottom and converts the code into a machine language. To interpret and execute JavaScript code, two large areas are required: the *memory heap* and *call stack* [3], [19], [22].

- **Memory heap.** When variables and objects are declared in the JavaScript code, the JavaScript engine allocates memory to them and stores the allocated memory information in the *memory heap*.
- **Call stack.** When the JavaScript engine finds an executable syntax in the script code, such as a function call, it adds the syntax into the *call stack* and executes the stored syntax one by one according to the last-in-first-out (LIFO) format.

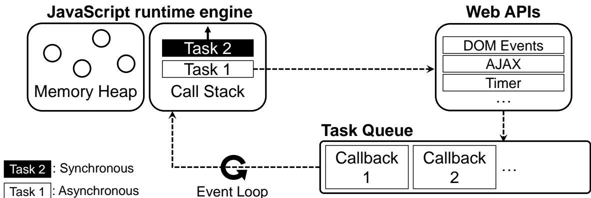


FIGURE 2: The workflow of the JavaScript engine. The JavaScript engine only handles one task at a time stored in the call stack, *i.e.*, single-thread process.

If an asynchronous function is executed (*e.g.*, a callback function), the JavaScript engine calls the *web API*, which is provided by the browser. The web API stores an asynchronously executed function in the *task queue*. Thereafter, the *event loop* [15] checks the status of the *call stack* and *task queue*, and when the *call stack* becomes empty, the first callback of the *task queue* is put into the *call stack* and executed.

4) WEB WORKER

JavaScript has become one of the most popular languages [9], [10], [33], and the cryptojacking that leverages it has also been on the rise recently [5]. In JavaScript, web workers enable multi-threaded processing. Previously, JavaScript only supported a single-thread process, meaning that JavaScript could only process one task at a time. Therefore, when a task was performed, the following task waited until the previous task was completed. If websites had heavy tasks that could not afford a single thread, they became unresponsive due to the overhead. To address this problem, a web worker [30], [31] was introduced to support a multithread process in JavaScript. As cryptomining requires a lot of resources to recursively check the validity of several blocks connected to a cryptocurrency network (*i.e.*, a heavy task), it is indispensable that browser-based cryptomining is implemented through a multi-thread process. Consequently, the appearance of web workers has a significant influence on making cryptojacking more active.

5) DATA TYPES IN JAVASCRIPT

In JavaScript, data types belong to two categories: *primitive value* and *reference value* [6].

- **Primitive value:** When primitive values are assigned to variables, they are stored in fixed sizes in the memory; therefore, they are stored on the call stack along with the actual values. JavaScript provides the following types of primitive values called *wrapper objects*: `number`, `string`, `boolean`, `null`, `undefined`, and `symbol` [1].

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Access. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3204814

- **Reference value:** When the variables are not assigned to *wrapper objects*, they are used as reference values. The size of the reference value is not fixed; therefore, it is stored in the heap along with its location; variables only have memory addresses instead of values for data. All data types, except *wrapper objects*, are contained in the reference variables (*e.g.*, array, object, and function).

As an example of these two data types, Listing 1 presents the difference between primitive and reference values.

Listing 1: Example code showing the difference between primitive and reference values.

```
1  //Case (1): primitive value
2  var a = 100;
3  var b = a;
4  a = 99;
5  console.log(b); //100
6
7  //Case (2): reference value
8  var a = { num : 100 };
9  var b = a;
10 a.num = 99;
11 console.log (b); //99
```

Case (1) in Listing 1 presents the case when a primitive variable is copied to a certain variable. Since the value of the variable is copied, variable `b` outputs the previous value of variable `a`. By contrast, case (2) in Listing 1 presents the case when a reference variable is copied to a certain variable. As the reference value stores the address in the memory, variable `b` is changed along with the modification of variable `a` because values with the same memory address always refer to the same data; this allocation of memory addresses to access data is referred to as a *reference* in JavaScript.

6) PROTOTYPE-BASED LANGUAGE IN JAVASCRIPT

To understand code reuse in JavaScript, we introduce the concept of prototype-based programming language in JavaScript. As explained in Section II-A5, most variables are *objects*, except for those assigned a primitive type. Every object in JavaScript has a property that has keys and values, and this property is called a *prototype* [21], [26]. When creating an object, it can inherit methods and properties from a parent object in a template format; this is called the *prototype chain* [20] (see Listing 2).

Listing 2: An example of JavaScript prototype chains.

```
1  var foo = {foo : "foo"};
2  var a = Object.create(foo);
3  var array = [];
4  var func = function () {
5     console.log("foo");
6  }
7  Window instanceof Object // True
8  /* "Window" refers to the page itself where the script
9     is currently running */
```

Listing 2 presents an instance of the JavaScript code used to describe the prototype chain, and Table 1 lists the prototype chains for the corresponding code. As the basic type of JavaScript is the object, all elements, such as functions and arrays, are linked to a top-level object, `Object.prototype`. The top-level object has null as its prototype; therefore, the prototype chain ends.

TABLE 1: Prototype chains for Listing 2.

#Line	Prototype Chain
Line #1	foo → Object.prototype → null
Line #2	a → foo → Object.prototype → null
Line #3	array → Array.prototype → Object.prototype → null
Lines #4 - #6	func → Function.prototype → Object.prototype → null

B. RELATED WORK

Several existing approaches detect and prevent threats caused by cryptojacking. We reviewed four types of existing approaches: (1) blacklisting-based, (2) resource monitoring-based, (3) thread count-based, and (4) WebAssembly-based approaches.

(1) Blacklisting-based approach. These approaches store elements with unique cryptojacking characteristics (*e.g.*, external resources links and script codes) as keywords in the blacklist and use them to detect cryptojacking [14], [16], [25], [37], [42], [45]. If the stored keywords are detected on a website (*e.g.*, if the domain of a website is the same as a blacklisted domain), the website is considered as a cryptojacking website. This approach is useful for detecting cryptojacking when an attacker fetches and abuses known cryptojacking code.

(2) Resource monitoring-based approach. A resource monitoring approach is based on the fact that cryptojacking is a resource-intensive task [38], [40]. This method detects a website as a cryptojacking website if the computer resources (*e.g.*, CPU usage) exceed a predetermined threshold when visiting the website. In particular, this approach has been highlighted as a new detection mechanism because it is not affected by script code obfuscation and is more convenient than a blacklisting-based approach requiring continuous management of blacklists.

(3) Thread count-based approach. As cryptojacking requires continuous mining, a thread with a separate execution space was created to proceed with mining. Unlike a normal website, the number of threads on a cryptojacking website is proportional to profitability [38], [41]. Consequently, several approaches have found a difference in the number of threads between cryptojacking and normal websites, and proposed methods can be utilized for cryptojacking detection [41]. This approach detects cryptojacking more flexibly than blacklisting-based or resource-monitoring-based approaches.

(4) WebAssembly-based approach. Wasm is a binary instruction format that can run in modern web browsers along with JavaScript [32]. It provides near-native performance for web applications, and any language (*e.g.*, C, C++, and Rust) can be compiled. Owing to the advantages of Wasm, an increasing number of attackers are using Wasm to employ cryptojacking websites [44]. In light of this, several approaches [35], [42] targeted cryptojacking websites based on Wasm, and proposed detection methods using static and dynamic features related to Wasm (*e.g.*, by counting Wasm instructions).

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Access. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3204814

Limitations of existing approaches. Existing approaches provide insights into detecting cryptojacking websites; however, we confirmed that each has limitations in precisely detecting cryptojacking websites.

Blacklisting-based approaches have two main limitations. As this approach is solely dependent on the stored keywords, keywords related to cryptojacking must be periodically collected; thus, when new cryptojacking appears, it is impossible to detect until the relevant keyword is stored in the blacklist. Furthermore, attackers can easily bypass blacklist-based detection by creating keywords that are not included in the blacklist using obfuscation or DGA. Hong *et al.* [38] and Konoth *et al.* [42] systematically analyzed cryptojacking. Specifically, Hong *et al.* [38] determined the life cycle of cryptojacking websites and the proper blacklist updating period, and proved that *it was not enough* to detect cryptojacking by relying only on the blacklist. By contrast, resource monitoring-based approaches have a false-positive problem. Recently, it is more common to provide extensive work to the web environment (*e.g.*, real-time video streaming) that shows high resource usage. Therefore, simply relying on resource usage monitoring can result in normal websites with high resource usage being mistaken as cryptojacking websites. In addition, thread count-based approaches cannot precisely detect cryptojacking because normal websites using multiple threads have appeared. Finally, Wasm-based approaches exhibited low detection coverage; even if several websites that employed Wasm were malicious, only 0.16% of the websites used Wasm among the Alexa Top 1 million websites [44]. As the proportion of Wasm-based websites is insignificant, JavaScript-based cryptojacking websites should be included in the scope of detection.

III. DESIGN OF CIRCUIT

This section introduces the CIRCUIT methodology, which focuses on detecting JavaScript-based cryptojacking websites and is robust against JavaScript code obfuscation. Figure 3 shows the high-level workflow of CIRCUIT.

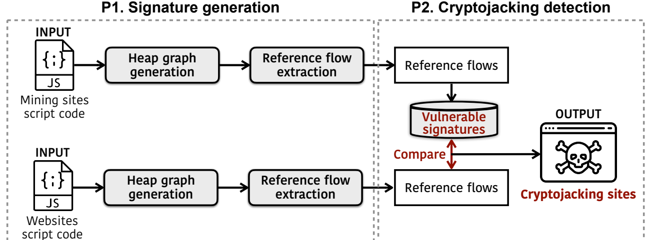


FIGURE 3: High-level workflow of CIRCUIT.

A. OVERVIEW

CIRCUIT comprises the following two phases: (1) P1 for generating signatures and (2) P2 for detecting cryptojacking. In P1, CIRCUIT first generates a *heap graph* that shows the behavior of the script code running on the website to detect cryptojacking, even if its script codes are obfuscated. CIRCUIT then extracts *reference flows*, that refer to the reference relations between objects in JavaScript. As the reference flows can denote the call flow of objects, we decided that the reference flows would represent cryptojacking behaviors. Therefore, CIRCUIT stores the reference flows of known cryptojacking websites as cryptojacking signatures. In P2, CIRCUIT compares the reference flows of the target websites with the signatures. If the reference flow of the target website resembles that of cryptojacking websites, CIRCUIT identifies the target website as a cryptojacking website.

Key idea. CIRCUIT utilizes the fact that script code obfuscation does not directly affect the information stored in memory, and web threads are stored in the memory area as objects. Thus, it is very flexible for indistinguishable script codes and can be analyzed by classifying web threads individually. If a mining-related thread is discovered on a website, it is identified as a cryptojacking site.

To precisely detect cryptojacking sites, we leveraged two key observations as follows:

1) **Form of cryptojacking code reuse.** Cryptojacking source code is generally provided by vendors through external links, and attackers utilize it in the form of third-party libraries [4], [36], [38].
2) **Distinguishable behaviors of cryptojacking.** To gain benefits, cryptojacking should perform its own mining behaviors, distinguishable from normal websites, *e.g.*, as joining a mining pool → mining cryptocurrency → sending rewards to attackers.

These two observations provide the following intuition: since cryptojacking is utilized in a third-party library form (*i.e.*, cryptojacking families), the JavaScript call stack and memory heap are comparable among websites using the same cryptojacking [41]. Furthermore, each cryptojacking contains its behavior; therefore, we can use the behavior as the signature of cryptojacking and detect cryptojacking websites by analyzing whether a particular website contains the same or similar behaviors of cryptojacking.

B. SIGNATURE GENERATION (P1)

This section introduces the methodology for heap graph generation (Section III-B1) and reference flow extraction (Section III-B2).

1) HEAP GRAPH GENERATION

First, CIRCUIT generates a JavaScript memory heap graph from a website. A node in the graph is a set of all the objects in the memory heap of the JavaScript engine, where the object includes special types, such as wrapper objects and window objects. An edge in a graph is a set of values that expresses the reference relation between two objects. In other words, it is a set of values in which a memory address value is allocated to access the corresponding memory address (*e.g.*, variable names).

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Access. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3204814

Let us consider the following code snippet as the running example.

Listing 3: Example of a JavaScript code snippet to illustrate the heap graph generation.

```
1  Class Foo {
2     constructor (value) {
3        this.key = value;
4     }
5  }
6  var a = var Foo("foo");
```

The obfuscated code for Listing 3 is shown in Figure 4 (a). Even if Listing 3 contains only variable declaration statements, the obfuscated code of Listing 3 is difficult to understand. However, the memory heap contains the declared object and variable names (see Figure 4 (b)); therefore, we can identify them via the memory heap. Thus, we only consider the memory heap of JavaScript.

To obtain the memory heap information from a website, we take a snapshot of the website when all contents of the document (*e.g.*, images, scripts, and CSS) are loaded (by load event [34]). As JavaScript is an interpreted language, memory is steadily allocated and deallocated while a website is running. Specifically, the allocated memory is automatically deallocated when the variables and objects corresponding to the memory in the source code are no longer required because of garbage collection (GC) [24]. Fortunately, cryptojacking has a pattern of executing repetitive tasks within a website for mining, and thus, the allocated memory is not deallocated before a user leaves the website; there is no loss of memory information through GC. Therefore, we decide to take a snapshot of the website with all contents of the document loaded. To extract the memory heap of the websites, we can easily obtain the objects declared in the website script code and their reference relations by taking heap snapshots using the JavaScript engine. For instance, the V8 JavaScript engine [29] provides data in JSON format, and object and reference information can be obtained by parsing the corresponding JSON.

Next, as described in Section II-A6, if objects with reference relations are connected, a heap graph is constructed. In the running example (Listing 3), "foo" exists in the `string` node because it belongs to the `wrapper object` as a `string` type of data. As the variable "`key`" refers to the memory address where the value of "foo" is stored, it is converted to an edge and connects "`Foo`". The variable "`a`", created by the constructor function of class "`Foo`" is a value that has the memory address for the created "`Foo`" object, and therefore "`a`" is converted into an edge that connects the node indicating the web page itself and the "`Foo`" node. Thus, to access the value "foo" from a web page, we first access "`Foo`" node by "`a`" edge which has the memory address value of "`Foo`", and then access "foo" node by a `key` edge that also has the memory address of "foo". Figure 4 depicts the overall flow where Listing 3 is converted into a heap graph.

The generated heap graph can express the reference relations between the objects declared on the website; therefore,

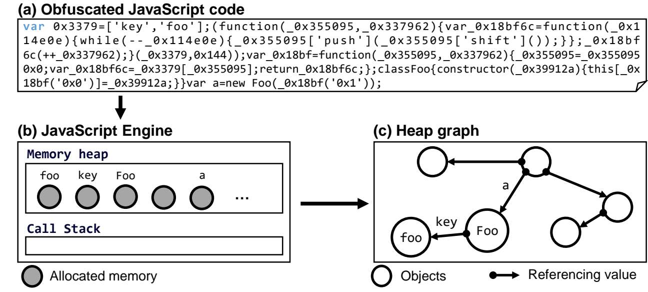


FIGURE 4: Overview of heap graph generation.

we can grasp the passing of all object flows to access a particular object. Consequently, the heap graph can identify and display declared variables or objects, even though the script code of a website is obfuscated.

2) REFERENCE FLOW EXTRACTION

CIRCUIT extracts reference flows from the generated heap graph. Reference flows are defined as the reference relations between objects in JavaScript, which denote the call flows of objects. We first reduce the searching space by focusing on the existence of a multi-thread. As previously explained in Section II-A4, running a multi-thread is an essential property for cryptojacking. Consequently, to determine whether a website runs multiple threads, we confirm whether a web worker exists in the heap graph of the website. In general, if a website runs multi-thread, the `WebWorker` object is contained in the memory heap, as shown in Figure 5. Subsequently, CIRCUIT first finds the `WebWorker` node in the heap graph to determine whether the website runs multi-thread, and thereafter CIRCUIT attempts to extract reference flows from the heap graph.

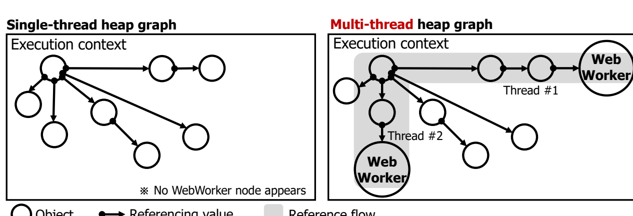


FIGURE 5: The illustration of the difference between a single-thread heap graph and multi-thread heap graph, particularly based on the WebWorker nodes.

Since the `WebWorker` object is created through the constructor function on a website [30], [31], a value that refers to the memory address of the `WebWorker` object must exist, and this value remains as an unique path that can be accessed for use on a website. To obtain a reference flow for each web worker, CIRCUIT defines the execution context [2], which is an environment for executing the JavaScript code as the start node and `WebWorker` object as the end node.

Subsequently, we traverse the heap graph using the depth-first search (DFS) and collect all the nodes and edges passing between the execution context and `WebWorker` object as

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Access. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3204814

the reference flow of each web worker. In the reference flow, information about the objects declared by the web worker and the reference relations between various objects are revealed. In other words, in the reference flow that directly executes cryptojacking, the object and reference relation related to the actual mining process are revealed. Thus, we employ the reference flow to detect cryptojacking websites.

C. CRYPTOJACKING DETECTION (P2)

Next, CIRCUIT detects cryptojacking websites using the extracted reference flows.

Signatures for cryptojacking websites. To identify whether the extracted web worker's reference flow contains cryptojacking behavior, we first explain the cryptojacking structure and how it is accessed and executed. The cryptomining script code has three areas: the `head`, `body`, and `tail`. The `head` is a script code area for importing cryptojacking related resources (*e.g.*, objects and variables) with an external server link. The `body` is a code area that declares the necessary functions and objects before the mining operation is executed on a cryptojacking website. Finally, the `tail` is a code area where an object is created for mininig on the client side, and the mining is executed.

Listing 4: Script code of CoinIMP.

```
1  <script src="https://www.hostingcloud.racing/A8P2.js">
2  </script>
3
4  var a=['G8KsSsOpP8KU', 'fWoRw5DClEjCr ... HDrE4f'];
5  //The script code of the corresponding 'A8P2.js' file
6
7  <script>
8    var miner = new Client.Anonymous('<site-key>');
9    miner.start();
10 </script>
```

For example, Listing 4 represents a real-world cryptojacking code (*i.e.*, CoinIMP). In this code snippet, `lines` #1 and #2 belong to the `head`, `line` #4 belongs to the `body`, and `lines` #8 and #9 belong to the `tail`. As mentioned in Section III-A, the cryptojacking code is mainly distributed in a general third-party form and is executed through the same script code from each cryptojacking vendor. Therefore, if the websites utilize the same cryptojacking vendor, the `head`, `body`, and `tail` of cryptojacking codes will be similar. As the operations performed by cryptojacking, particularly the mining operations performed on the `body`, remain identifiable in the memory heap, we can use this information to detect cryptojacking websites, irrespective of code obfuscation.

Therefore, we collect the cryptomining script code provided by the cryptojacking vendors. To extract reference flows from the collected cryptomining script code, we create an arbitrary website to open a web server inside and embed the collected script code. We then implement a cryptomining website using the collected cryptomining script code by referring to the provided usage document and storing the heap information of the JavaScript engine created when the website is executed. Subsequently, we generate the heap graph from the JavaScript memory heap and then extract the reference flows from each web worker. The extracted reference flows for each vendor are indexed by the name of each vendor. Figure 6 shows examples of the extracted reference flows from seven known cryptojacking websites.

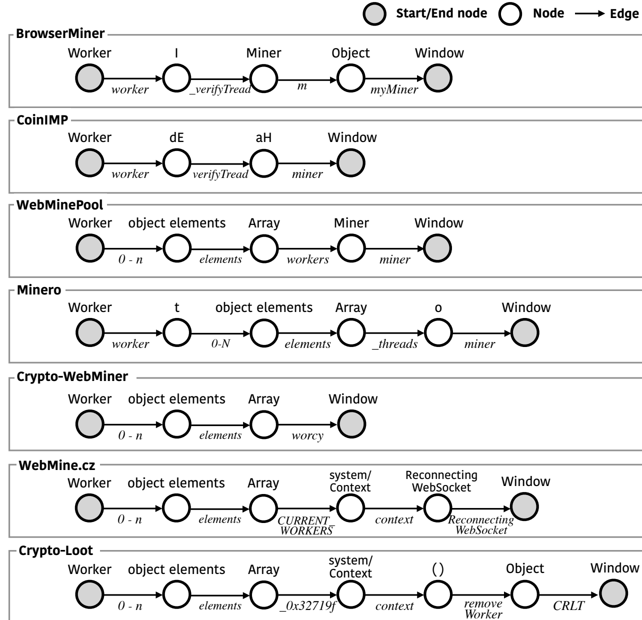


FIGURE 6: Example of extracted reference flows from known cryptojacking websites.

Detecting cryptojacking websites. Finally, CIRCUIT detects cryptojacking websites using extracted cryptomining reference flows. To confirm that a target website contains cryptojacking, we extract all reference flows from the target website and compare every extracted reference flow to the indexed cryptomining reference flows. Here, we employ an edit distance algorithm [17] and calculate the edit distance between all the reference flows obtained from the target website and all the indexed cryptomining reference flows. If any pair shows an edit distance below the predefined threshold (we set 5 as the threshold; see Section IV-A), CIRCUIT identifies the target website as a cryptojacking website. The algorithm that detects cryptojacking websites is presented in Algorithm (1).

IV. EVALUATIONS AND FINDINGS

In this section, we evaluate CIRCUIT. We first evaluated the cryptojacking detection results of CIRCUIT using popular real-world websites. CIRCUIT was tested for its coping ability with techniques used to evade cryptojacking detection (*e.g.*, obfuscation). Finally, we introduced findings on the detected cryptojacking websites. We ran CIRCUIT on a machine with Ubuntu 18.04 LTS, 3.8 GHz AMD Ryzen processor, 32 GB RAM, and 1 TB SSD.

Dataset collection. The experiment collected real-world websites from the dataset. Specifically, we decided to collect

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Access. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3204814

Algorithm 1: Algorithm for detecting cryptojacking sites.

```
Input: K, T
// K: known cryptojacking websites, T: a target website
Output: C
// C: a list of cryptojacking injected websites
1  procedure DETECTINGCRYPTOJACKING(K, T)
2      C ← ∅
3      S_K ← ExtractingSignature(K)
       // A unique set of reference flows of K
4      for T_i in T do
5          S_T ← ExtractingSignature(T_i)
           // A unique set of reference flows of T
6          if S_T == false then
7              continue
8          for t in S_T do
9              for k in S_K do
10                 if IsSimilar(t, k) then
                       // Determining T_i as the cryptojacking website
11                     C.append(T_i)
12     return C
13 procedure EXTRACTINGSIGNATURE(S)
14     if IsWebWorker(S) then
15         R ← ∅ // R: Reference flows
16         H ← takeHeapSnapshot(S)
           // Take a heap snapshot for website
17         startNode, workerNode, NodeEdgeList ← MemoryHeapGeneration(heap)
           // Extract reference flow by searching the nodes with DFS
18         R.append(extractReferenceFlows(startNode, workerNode, NodeEdgeList))
19         return R
20     else
21         return false
```

popular websites that have greater impacts on several users, and then confirmed the existence of cryptojacking websites. We collected 300,000 websites listed in Amazon's Alexa top website service [7] and Majestic [28], which provide the world's most popular website list for free, and then gathered top websites in both lists to confirm the distribution of cryptojacking in the overall Internet environment. Furthermore, to identify the website service field where cryptojacking is distributed, we also collected an additional Alexa category top service [8] that indexes websites by category. We collected a list of 6,000 websites, each with 500 of the most popular rankings for 12 categories. Therefore, we collected 306,000 websites as our dataset to evaluate CIRCUIT (see Table 2).

Memory heap collection. We developed a crawler that stores the memory heap area of a visited website using the *remote interface* [13] and *puppeteer* [27] functions of the Chrome browser [12]. This crawler visited the collected 306,000 websites, and after waiting for the website content to finish loading (*i.e.*, load event), it extracted a snapshot of the memory heap area of the JavaScript engine. Here, if the connection time of the website exceeds 30,000 ms or the website cannot be accessed from the domain name system (DNS) server, the crawler ignores the website. Therefore, our crawler collected memory heap areas from 204,773 websites to evaluate CIRCUIT, and the results are summarized in Table 3.

TABLE 2: Summary of the collected websites for our experiment.

Category	#Total websites
Alexa top 100K websites	100,000
Majestic top 200K websites	200,000
Alexa category top websites	
Adult	500
Arts	500
Business	500
Computers	500
Games	500
Health	500
Home	500
Kids and Teens	500
News	500
Recreation	500
Reference	500
Regional	500
Total	306,000

TABLE 3: Summary of the collected memory heap from the website dataset.

Category	#Websites	#Heap extracted†	Collection date
Alexa top 100K	100,000	82,081	June 28, 2022
Majestic top 200K	200,000	117,833	June 12, 2022
Alexa category top	6,000	4,859	May 31, 2022
Total	306,000	204,773	N/A

†: The number of websites from which the memory heap was successfully extracted.

A. DETECTION OF CRYPTOJACKING IN THE REAL-WORLD WEBSITES

Methodology. First, we extracted seven reference flows from the seven known cryptojacking websites as signatures for cryptojacking behaviors (see Figure 6). Thereafter, from the 204,733 heap graphs generated for common websites (Table 3), we extracted 49,791 reference flows related to web workers. The number of reference flows related to web workers is significantly below the number of heap graphs because we ignored websites that only executed a single-thread (see Section III). We then compared the extracted reference flows to the stored cryptojacking signatures by employing the Python library to obtain the edit distance between the two reference flows. Specifically, we used the `networkx` library, which contains the "`similarity.optimize_graph_edit_distance`" function that measures the difference between two graphs as an integer greater than or equal to zero; if the distance

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Access. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3204814

is zero, the two input graphs are the same. Hence, we set the threshold to 5 (defined in Section III-C) and determined two graphs (*i.e.*, two reference flows) with an edit distance of below 5 as similar. We decided that the target website that contains a similar reference flow to cryptojacking signatures was the cryptojacking website.

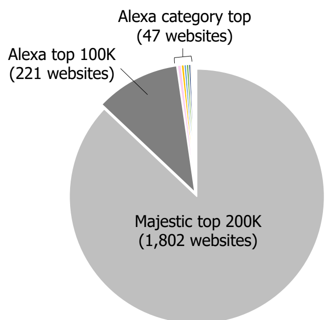


FIGURE 7: Distribution of the cryptojacking websites detected by CIRCUIT.

Detection results. In our experiments, we found that 2,423 reference flows from 1,813 websites are similar to cryptojacking signatures. Figure 7 presents the distribution of cryptojacking websites detected by CIRCUIT; note that several websites belong to multiple groups. From the results, we confirmed the following three observations.

1) Most detected cryptojacking websites (1,802 websites) belong to the Majestic top 200K group.
2) When comparing the results of Alexa top 100K and Majestic top 200K, less popular websites (top 101K to 200K) may contain more cryptojacking behaviors than very popular ones (top 1 to 100K).
3) Cryptojacking websites were hardly discovered in the Alexa category top groups (top 500 per each category).

Since all detected websites contain a reference flow similar to that of cryptojacking websites, the detected websites contain the cryptojacking behaviors, either potentially or directly. Manually inspecting all the detected websites is an error-prone and burdensome task, and thus, we randomly selected 100 websites (6%) and manually checked whether they performed cryptojacking. To verify our results, as most of cryptojacking websites leverage evasion techniques to hide cryptojacking behaviors, we checked the CPU usage of websites, an evaluation method that was used in the existing approaches [35], [38], [42]; since we have already confirmed that the websites detected by CIRCUIT contain cryptojacking signatures, we decided that it was valid to verify them by further investigating the CPU usage. As a result, all the 100 selected websites exhibited over 55% CPU usage; 25 out of the 100 websites showed over 90% CPU usage. The CPU usage of the verified websites was significantly higher than that of the normal websites; the normal websites exhibited below 1% CPU usage on average. This result affirmed that CIRCUIT successfully detected malicious websites that were actually running cryptojacking behaviors.

The main advantage of CIRCUIT is that it has reported fewer false positives. In existing approaches (*e.g.*, Outguard [41]), for example, if the number of threads on a website is greater than the threshold, or if the resource consumption is higher than the threshold, all of them are determined as cryptojacking websites. Although these websites may use the resources of visitors, some of them ask for the consent of the visitor, and most of them have a lower influence on visitors than cryptojacking websites in terms of resource consumption. Thus, we can argue that our result is more precise and compact because CIRCUIT detects only cryptojacking websites that clearly contain the cryptojacking behavior.

B. EVASION TECHNIQUES

As cryptojacking websites were blocked by the emergence of several applications, such as Dr.Mine [16] and MinerBlock [25], attackers started hiding the mining script code to avoid cryptojacking detection. Therefore, we gathered the evasion techniques found in our experiment and summarized them as the following four evasion models (**E1** to **E4**). Figure 8 shows the heap graphs for each evasion technique.

E1: Obfuscating the script code. Obfuscation is obfuscating and compressing cryptojacking script codes on a website, or to hide notable keywords in the script code using the `CharCode` or `eval` function. This is one of the representative evasion techniques used to avoid cryptojacking detection, which makes it difficult to detect cryptojacking using a static analysis technique [39], [43]. An instance of code obfuscation is presented in Listing 5, and the entire heap graph of the corresponding code is presented in Figure 8 (b).

Listing 5: Example code for obfuscating the script code.

```
1 var _0x532b=[{_0x462888["push"](_0x462888["shift"] ());}...];
```

Blacklisting based approaches, particularly when utilizing the script code of cryptojacking as a blacklist, may fail to detect code-obfuscated cryptojacking websites. Since CIRCUIT leverages memory heap information rather than script code, we can detect cryptojacking websites regardless of script code obfuscation.

E2: Modifying external server link. This technique bypasses cryptojacking detection by changing the link to load the cryptojacking script code to a random value. In extreme cases, the external server link provided by the cryptojacking vendor is first fetched by the attackers and stored in their web server, and then the cryptojacking code is loaded on its own. Here, detection is bypassed by changing the name of the script file containing the cryptojacking script code to a generic name such as "`jquery.js`" or "`analysis.js`", as described in the below sample code (Listing 6).

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Access. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3204814

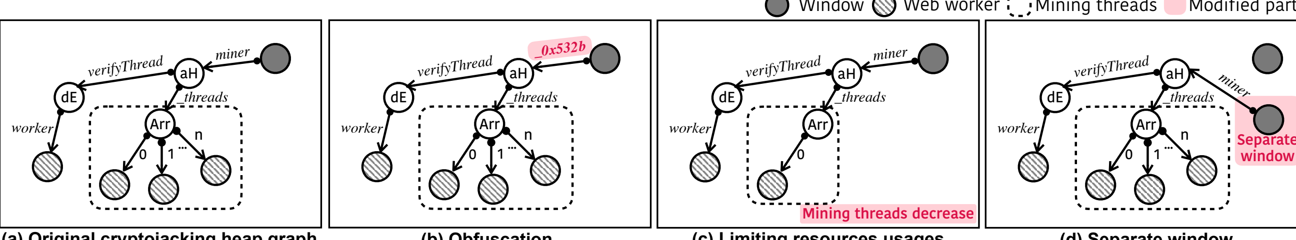


FIGURE 8: Illustrations of heap graphs that change by various evasion techniques. In (c), the same reference flow (*i.e.*, thread #0) as in (a) remains identical even if the evasion technique is applied, thus, the edit distance between the two reference flows is zero. For (b) and (d), changes occurred one by one at the node and edge of the mining reference flow, respectively, but the edit distance between mining thread #0 in (a) and mining thread "0" in both of (b) and (d) is exceedingly small (the measured edit distance is 2). Note that the evasion technique for modifying the external server link does not affect the original heap graph.

Listing 6: Example code for modifying external server link.

```
1 <script src = "/jquery.js"></script>
2 //below code shows the script code in the jquery.js file
3
4 (function () {
5    Mininig script code ...
6 })();
```

Since the memory heap of the script code was not modified by this technique, the detection mechanism of CIRCUIT was not affected by this evasion method.

E3: Limiting resource usage. To disguise a cryptojacking website as a normal website, attackers sometimes limit the computing resource usage during cryptocurrency mining, *e.g.*, by reducing the number of mining threads. This technique does not change significantly in mining script code, but it is an option that is often utilized to bypass the detection method based on resource monitoring. For instance, attackers can leverage this technique by adding the following simple option (*i.e.*, `throttle`) to their script code:

Listing 7: Example code for limiting resource usage.

```
1 var miner = new Client.Anonymous("<SITE-KEY>", {throttle: 0.1});
```

Here, detection methods based on resource monitoring and thread counts may fail to detect cryptojacking websites. However, even if the number of mining threads decreases, the behavior of existing reference flows is maintained (*e.g.*, mining thread #0 in Figure 8 (c)); therefore, CIRCUIT can precisely detect these kinds of cryptojacking websites.

E4: Utilizing a separate window. Cryptojacking websites bypass detection by embedding a separate cryptojacking code, such as `iframe`, on the website, allowing cryptomining without a specific script code. In addition, by applying obfuscation to the embedded cryptojacking code, cryptojacking detection becomes more difficult. The sample code is presented in Listing 8.

Listing 8: Example code for utilizing a separate window.

```
1    <iframe width=0 height=0 frameborder=0
2    src="https://cryptomining.com/mining?key=<SITE-KEY>">
3    </iframe>
```

However, as shown in Figure 8 (d), only the start node of the reference flow is replaced with another object, and there is no change in the internal behavior. Therefore, CIRCUIT can detect cryptojacking websites even if this evasion technique is applied.

C. DISTRIBUTION OF WEBSITES WITH WEB WORKERS

As previously explained in Section II-B, some of the recent approaches to detect cryptojacking have focused on the fact that cryptojacking websites run several threads. AI learning using this indicator effectively detects cryptojacking websites, but several normal websites using multiple web workers have also been mistakenly detected as cryptojacking websites. Therefore, we checked the number of web workers on these websites.

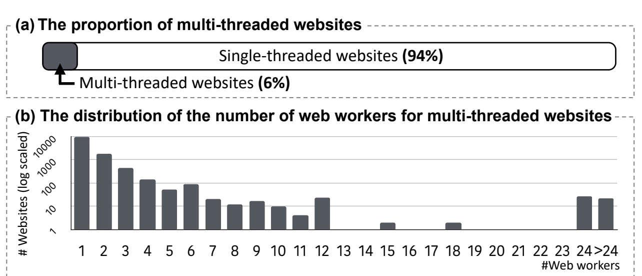


FIGURE 9: Illustration of the proportion of multi-threaded websites (a) and the distribution of the number of web workers in multi-threaded websites (b).

Consequently, 11,898 websites out of the total of 204,773 websites were using at least one web worker (*i.e.*, running multiple threads). Figure 9 (a) shows that websites that use web workers accounts for only 6% of the total collected websites. In addition, Figure 9 (b) shows the distribution of the number of web workers in multi-threaded websites, and it is observed that a website uses at least one to a maximum of 57 web workers, with an average of 1.4 web workers. Thread count-based approaches consider the number of web workers (number of threads) as an indicator; if at least three web workers are included, then we identify a website as a cryptojacking website. Among the 11,898 multi-threaded websites,

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Access. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3204814

413 websites used three or more web workers (3.6%). When we additionally checked all 413 websites manually, websites that were not injected cryptojacking included multimedia processing or functions, such as Google reCAPTCHA, by web workers. In summary, the difference in the number of web workers shows the effect of significantly reducing the scope range in the overall scale when detecting cryptojacking websites. However, for precise detection, the number of web workers cannot be an absolute indicator. Therefore, a more mature approach, such as CIRCUIT, which performs behavior-based detection of cryptojacking through memory heap analysis, is efficient from this perspective.

D. WEBSITES WITH MULTI-SERVICES

To evaluate the effectiveness of CIRCUIT on a website that provides multiple services using different web workers, we examined websites that use multiple web workers. However, in our collected dataset, no website was found to simultaneously provide cryptojacking behavior and normal services; this implies that a website generally uses multiple web workers for the same service or only one web worker.

Therefore, for evaluation purpose, we intentionally inserted cryptomining code into a website that uses a normal web worker service to create a complex structured website that runs multiple web workers in parallel. The cryptomining script code was injected at the client level using the developer tool provided by the browser; this does not affect the web server. Figure 10 shows the generated heap graph focusing on the identified reference flow after injecting the cryptomining code of CoinIMP into the "`057.ua`" website, which uses a web worker intentionally. In the heap graph, a web worker created using Google's reCAPTCHA and a web worker for cryptomining exist simultaneously, together with seven other web workers, as shown in Figure 10. In this example, the existing resource monitoring-based approach or thread count-based approach determines that this website runs cryptojacking before inserting the cryptomining code. In addition, if we obfuscate the cryptomining code and insert it into a website, blacklisting-based approaches fail to detect this website as a cryptojacking website.

By contrast, since CIRCUIT considers an individual reference flow for each web worker, it can detect only web workers related to cryptojacking, even in a complex structure. When similarity was measured based on the reference flow of CoinIMP, the reference flow of the web worker used in Google's reCAPTCHA showed an edit distance of 11.0, whereas the injected cryptomining reference flow showed an edit distance of 2.0. This is not a characteristic of Google reCAPTCHA. For instance, when we measured the similarity between reference flows of "`Video.js`" [23], "`hls.js`" [18], and "`vectortaillay.js`" [11], which are generally executed by various web workers, and the reference flows of CoinIMP, the edit distances were obtained as 13.0, 18.0, and 32.0, respectively. In conclusion, CIRCUIT can precisely detect only web workers related to cryptomining, even on websites with multiple web workers.

V. DISCUSSION

Here we discuss several considerations related to CIRCUIT.

A. CRYPTOJACKING DETECTION BASED ON THE JAVASCRIPT MEMORY HEAP

Handling relatively heavy tasks in a web environment was challenging before the introduction of web workers. The distinction between cryptojacking and normal websites became ambiguous after introducing web workers; hence, methods for detecting cryptojacking websites are required. In addition, cryptojacking websites attempt to avoid detection through various evasion techniques. Therefore, we focused on how to flexibly cope with technologies to avoid detection and how to precisely detect cryptojacking websites. If the memory area allocated to the website is used, the detection ability will not be affected unless the evasion technique directly affects memory. CIRCUIT reduced false positives in cryptojacking detection and showed robust results compared with the existing detection methods. In addition, the analysis results of the evasion techniques and distribution of web workers in the overall web environment proved the necessity and efficiency of approaching memory rather than simply depending on the script code, resource consumption monitoring, or several threads. The detection method using this memory area can flexibly cope with detection bypass technologies, which hinder cryptojacking detection, and will become an important insight for detection methods focusing on accuracy.

B. LIMITATIONS

As CIRCUIT detects cryptojacking websites based on the JavaScript memory heap, it can flexibly cope with detection bypass techniques that do not directly affect the memory heap. However, in some cases, CIRCUIT can report false alarms.

- **Object encapsulation.** Mining-related objects can be abnormally encapsulated; thus, the reference flow can be extremely long in terms of memory, resulting in an extensive editing distance when measuring the similarity between reference flows.
- **Extremely short reference flow.** If the reference flow for a web worker is extremely short, the edit distance between two reference flows with different behaviors can be measured as a very small integer.

A specific reference flow of a web worker found at the site[1] had only three nodes and two edges. If an exceedingly short cryptojacking reference flow is added later, it can lead to false positives. Furthermore, we confirmed that the reference flow of a web worker generated at the site [2] has 41 nodes and 40 edges. If the reference flow containing the cryptojacking behavior becomes long because of abnormal encapsulation, this can lead to false negatives. In addition, the detection

[1]https://www.acs.org/
[2]https://www.chestnet.org/

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Access. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3204814

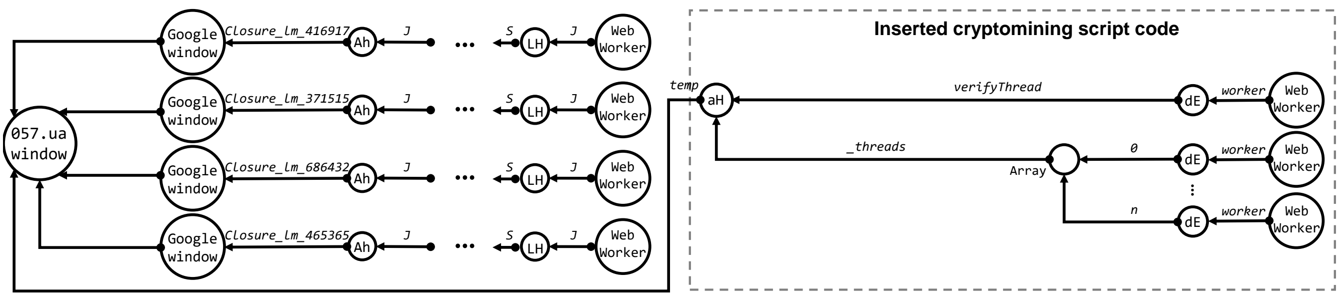


FIGURE 10: Example of a website heap graph that operates multiple services including cryptomining.

of a new cryptojacking code, *i.e.*, unknown cryptojacking, is outside the detection range of CIRCUIT. We are fully aware of this problem, and as a possible solution, we considered taint analysis using the data flow of the `SITE-KEY`. However, we do not have a clear solution as yet; hence we leave this as future work. Finally, CIRCUIT cannot be used to detect Wasm-based cryptojacking websites. Thus far, it has been difficult to understand the memory structure of the web assembly, thus making the direct application of CIRCUIT challenging. However, if sufficient information about the memory structure is provided, the same CIRCUIT algorithm used in JavaScript can be applied to the web assembly.

VI. CONCLUSION

Increasing cryptocurrency values have led to an increase in cryptojacking, which utilizes mining maliciously. Therefore, we propose CIRCUIT, a precise approach for detecting cryptojacking websites based on the JavaScript memory heap. We define a reference flow, which can represent script code behavior for each thread on a website and utilize the reference flow to detect websites with cryptojacking behaviors. CIRCUIT successfully detected 1,813 cryptojacking websites from 300K real-world websites. We demonstrated the efficacy of CIRCUIT by (1) precisely detecting cryptojacking websites using evasion techniques and (2) clearly distinguishing normal websites with similar characteristics to cryptojacking websites. In addition, the model of evasion techniques that we discovered and the distribution of web workers within a website can provide new insights for cryptojacking detection.

REFERENCES

[1] The Wrapper Object, 2016. https://javascriptrefined.io/the-wrapper-object-400311b29151.
[2] Execution context, Scope chain and JavaScript internals, 2017. https://medium.com/@happymishra66/execution-context-in-javascript-319dd72e8e2c.
[3] The Javascript Runtime Environment, 2018. https://medium.com/@olinations/the-javascript-runtime-environment-d58fa2e60dd0.
[4] UK ICO, USCourts.gov... Thousands of websites hijacked by hidden crypto-mining code after popular plugin pwned, 2018. https://www.theregister.com/2018/02/11/browsealoud_compromised_coinhive/.
[5] McAfee Labs Threats Report, 2019. https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-aug-2019.pdf.
[6] What are Primitive and Reference Types in JavaScript?, 2019. https://itnext.io/javascript-interview-prep-primitive-vs-reference-types-62eef165bec8.
[7] Alexa the top sites on the web, 2020. https://www.alexa.com/topsites.
[8] Alexa the top sites on the web by Category, 2020. https://www.alexa.com/topsites/category.
[9] 2021 Developer Survey, 2021. https://insights.stackoverflow.com/survey/2021.
[10] About JavaScript, 2022. https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript.
[11] ArcGIS API for JavaScript, Class: VectorTileLayer, 2022. https://developers.arcgis.com/javascript/3.
[12] Chrome, 2022. https://www.google.com/intl/en/chrome/.
[13] Chrome remote interface, 2022. https://github.com/cyrus-and/chrome-remote-interface.
[14] CoinBlockerLists, 2022. https://zerodot1.gitlab.io/CoinBlockerListsWeb.
[15] Concurrency model and the event loop, 2022. https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop.
[16] Dr. Mine, 2022. https://github.com/1lastBr3ath/drmine.
[17] Edit distance, 2022. https://en.wikipedia.org/wiki/Edit_distance.
[18] hls.js, 2022. https://github.com/video-dev/hls.js.
[19] How JavaScript Works: An Overview of JavaScript Engine, Heap, and Call Stack, 2022. https://dev.to/bipinrajbhar/how-javascript-works-under-the-hood-an-overview-of-javascript-engine-heap-and-call-stack-1j5o.
[20] Inheritance and the prototype chain, 2022. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Inheritance_and_the_prototype_chain.
[21] Javascript — A prototype based language, 2022. https://medium.com/@theflyingmantis/javascript-a-prototype-based-language-7e814cc7ae0b.
[22] JavaScript engine, 2022. https://en.wikipedia.org/wiki/JavaScript_engine.
[23] Make your player yours with the world's most popular open source HTML5 player framework , 2022. https://videojs.com.
[24] Memory Management, 2022. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Memory_Management.
[25] MinerBlock github, 2022. https://github.com/xd4rker/MinerBlock.
[26] Object-oriented programming, 2022. https://en.wikipedia.org/wiki/Object-oriented_programming.
[27] Puppeteer, 2022. https://github.com/puppeteer/puppeteer.
[28] The Majestic Million, 2022. https://majestic.com/reports/majestic-million.
[29] Visualizing memory management in V8 Engine (JavaScript, NodeJS, Deno, WebAssembly), 2022. https://deepu.tech/memory-management-in-v8.
[30] Web Workers, 2022. https://www.w3.org/TR/workers/.
[31] Web Workers API, 2022. https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API.
[32] WebAssembly, 2022. https://developer.mozilla.org/en-US/docs/WebAssembly.
[33] What is JavaScript?, 2022. https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript.
[34] Window: load event, 2022. https://developer.mozilla.org/en-US/docs/Web/API/Window/load_event.
[35] Weikang Bian, Wei Meng, and Mingxue Zhang. Minethrottle: Defending against wasm in-browser cryptojacking. In Proceedings of The Web Conference 2020, pages 3112–3118, 2020.
[36] Hugo LJ Bijmans, Tim M Booij, and Christian Doerr. Inadvertently making cyber criminals rich: A comprehensive study of cryptojacking

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Access. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3204814

campaigns at internet scale. In 28th USENIX Security Symposium (USENIX Security), pages 1627–1644, 2019.

[37] Shayan Eskandari, Andreas Leoutsarakos, Troy Mursch, and Jeremy Clark. A first look at browser-based cryptojacking. In 2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&P), pages 58–66. IEEE, 2018.

[38] Geng Hong, Zhemin Yang, Sen Yang, Lei Zhang, Yuhong Nan, Zhibo Zhang, Min Yang, Yuan Zhang, Zhiyun Qian, and Haixin Duan. How you get shot in the back: A systematical study about cryptojacking in the real world. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS), pages 1701–1713, 2018.

[39] Alexandros Kapravelos, Yan Shoshitaishvili, Marco Cova, Christopher Kruegel, and Giovanni Vigna. Revolver: An automated approach to the detection of evasive web-based malware. In 22nd USENIX Security Symposium (USENIX Security), pages 637–652, 2013.

[40] Conor Kelton, Aruna Balasubramanian, Ramya Raghavendra, and Mudhakar Srivatsa. Browser-based deep behavioral detection of web cryptomining with coinspy. In In Proc. Network and Distributed System Security Symposium (NDSS), 2020.

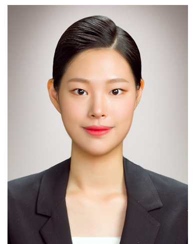
[41] Amin Kharraz, Zane Ma, Paul Murley, Charles Lever, Joshua Mason, Andrew Miller, Nikita Borisov, Manos Antonakakis, and Michael Bailey. Outguard: Detecting in-browser covert cryptocurrency mining in the wild. In The World Wide Web Conference, pages 840–852, 2019.

[42] 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 (CCS), pages 1714–1730, 2018.

[43] Andreas Moser, Christopher Kruegel, and Engin Kirda. Limits of static analysis for malware detection. In 23rd Annual Computer Security Applications Conference (ACSAC), pages 421–430. IEEE, 2007.

[44] Marius Musch, Christian Wressnegger, Martin Johns, and Konrad Rieck. New kid on the web: A study on the prevalence of webassembly in the wild. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pages 23–42. Springer, 2019.

[45] Jan Rüth, Torsten Zimmermann, Konrad Wolsing, and Oliver Hohlfeld. Digging into browser-based crypto mining. In Proceedings of the Internet Measurement Conference 2018, pages 70–76, 2018.



HYUNJI HONG received the B.S. degree in Computer Science and Engineering from Hanshin University, Gyeonggi-do, South Korea, in 2020. She is a M.S. candidate in the Department of Computer Science and Engineering, Korea University, Seoul, South Korea. Her research interests include software security, vulnerability detection, and vulnerability analysis.

SEUNGHOON WOO is the Research Professor in Center for Software Security and Assurance (CSSA), Korea University. He received his B.S., M.S., and Ph.D. degree in Computer Science and Engineering from Korea University. His research interests include software security, vulnerability detection, and software composition analysis. He has published papers on software security and software engineering in top conferences such as S&P, USENIX Security, and ICSE.

SUNGHAN PARK received the M.S. degree in the Department of Computer Science and Engineering from Korea University, Seoul, South Korea, in 2021. His research interests include web security, software analysis and malware detection.

JEONGWOOK LEE is an undergraduate student majoring in Computer Science and Engineering from Korea University, Seoul, South Korea. His research interests include vulnerability detection, malware detection and vulnerability analysis.

HEEJO LEE (Member, IEEE) is the Professor in the Department of Computer Science and Engineering at Korea University, and the director of Center for Software Security and Assurance (CSSA). He received his B.S., M.S., and Ph.D. degree in Computer Science and Engineering from POSTECH. Before joining Korea University, he served as a CTO at AhnLab Inc. from 2001 to 2003, and as a Post-doctorate Researcher at Purdue University from 2000 to 2001. He is an Editor of the Journal of Communications and Networks, and the International Journal of Network Management. He is a founding member and co-CEO of IOTCUBE Inc.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/