Journal of Computer Virology and Hacking Techniques
https://doi.org/10.1007/s11416-022-00420-7

ORIGINAL PAPER

MinerAlert: an hybrid approach for web mining detection

Franco Tommasi[1] · **Christian Catalano**[1] · **Umberto Corvaglia**[1] · **Ivan Taurino**[1]

Received: 25 September 2021 / Accepted: 10 February 2022

Abstract
The introduction of new memory-based crypto-mining techniques and the rise of new web technologies like WebAssembly, made the use of browsers for crypto-currencies mining more and more convenient and popular. That, in turn, originated a new form of computer piracy, called cryptojacking, which is rapidly gaining ground on the web. A cryptojacking site exploits its visitors' hardware resources to secretly mine crypto-currencies. This paper analyzes current web-based cryptojacking detection methods in order to propose a novel hybrid strategy. Current detection methods are found to require either considerable computer administration skills or execution privileges usually not available to common users. In this view, a method, named MinerAlert, has been designed and proposed, aiming at detecting in real-time sites performing cryptojacking. To address the limitations of current methods, the method implementation has been achieved through a browser extension. The present paper describes the method's details and its implementation. It also reports the experimental results of its utilization, showing its positive performances in terms of ease of use, successful detections and speed.

Keywords Malware · Cryptojacking · Web mining detection · Crypto-mining

1 Introduction

This paper aims at contributing to the current methods for detecting the unauthorized use of crypto-currencies mining techniques on a user's machine. Specific attention will be devoted to hidden web-based crypto-mining through a user's browser [1,2]. As in cryptojacking, this sort of crypto-mining makes use or the victim's CPU cache memory to execute the PoW algorithm [3–5]. This type of algorithms is largely used in most recent varieties of Altcoint [6–8] (Ethereum, Litecoin, Zcash, Monero, etc.) crypto-currencies. The improvements the present approach introduces are not only related to the ways to detect the mining but are also designed to make easy for a generic, inexperienced user to take advantage of them. For this purpose, no particular administrative permission and no special configuration are required. All is needed is the installation of a browser extension. The described technique is based on performance measures and behavioural features. Indeed, through their analysis it is possible to accurately discriminate between the presence or the absence of mining code executed on the machine. Such analysis is performed in real-time, during the normal user's navigation. Behaviours are evaluated for each visualized page (e.g. through the detection of WebSocket and Web Worker). By contrast, performance features are evaluated by a benchmark, specifically through the results of a customized version of the "Schönauer Triad Benchmark". All the measures and the observation are combined and rated through the Support Vector Machine (SVM) algorithm [9] to discriminate cryptojacking sites.

The paper is organized as follows. Section 2 proposes a short description of software solutions for browser-based mining detection already available on the market and in academic literature. Section 3 introduces the concept of "Mining Services". Developers or attackers could profit from such services to pursue their crypto-mining activities at clients' expenses. The services are typically based on sites offering easy API for crypto-mining. Section 4 presents a new hybrid approach for web mining detection, named MinerAlert. The proposed technique is based on the analysis of two differ-

✉ Christian Catalano
christian.catalano@unisalento.it

Franco Tommasi
franco.tommasi@unisalento.it

Umberto Corvaglia
umberto.corvaglia@unisalento.it

Ivan Taurino
ivan.taurino@unisalento.it

[1] Dipartimento di Ingegneria dell'Innovazione, University of Salento, Via per Monteroni, 73100 Lecce, Italy

Published online: 12 March 2022

ent aspects: the performances of the machine's hardware resources and the behavior of the visited web page. Section 5 details the implementation of the proposed solution, describes the datasets used for the training and testing phase, the workflow of the implemented software and the results obtained. Section 6 discusses the benefits and strengths of the proposed solution and highlights the differences between that and other currently adopted solutions. Section 7 highlights a few limitations of the proposed solution and lists possible suggestions for future improvements. Section 8 summarizes the research work done and emphasizes the importance of improving cryptojacking detection techniques.

2 State of the art

A fair choice of software solutions for browser-based mining detection are already available on the market as browser's extensions. However, it must be noted that many, if not most, of them use static (and, therefore, not of much help) detection techniques. On the other hand, most of the software solution proposed in literature are intrinsically complex and of little use for an average user. Among most known market solutions, *NoCoin* [10] and *MinerBlock* [11] may be mentioned. The static detection techniques used by these tools are basically of two types:

- *The search for particular keywords inside the page JavaScript code* The technique aims at identifying specific "Mining Services" [6,12,13] that is libraries implementing methods and functions needed for mining(PoW operations, information exchange with the mining pool, etc.)
- *Detection through blacklists* The technique takes for granted the existence of a list of malicious domains and IP addresses. Obviously such list must be constantly maintained and frequently updated to be of any use. A list of this type is maintained by "CoinBlockerLists" [14]

However both above techniques are ineffective as they can be easily eluded, the first by obfuscating JavaScript code, the second constantly changing domain and IP address [15–17]. Researches in the field brought to the production of software tools like MineGuard [18], CMTracker [19] and MineSweeper [13]. When compared with the above mentioned commercial solutions, such tools are actually based on more complex and effective procedures for mining detection. As an example MineGuard uses a low level approach, based on the analysis of periodic variations of specific performance indexes (Hardware Performance Counters). Another mining detection mechanism, proposed by Konoth et al. [13], consists in a static analysis of the Wasm code executed by the page. Such analysis is performed at run-time through the scan of the textual version of the Wasm code (.wat file) found in the web page. Both techniques bring to the fore how the execution of memory-based algorithms implies a large number of Load and Store operation on the cache.

3 Mining services

The growth of web-mining led to the rise of several "Mining Services". Developers could profit from such services to pursue their crypto-mining activities at victims' expenses. The services are typically based on sites offering easy API for crypto-mining and charging around 20-30% of the mined crypto-currencies. In 2017, Coinhive was the first to revisit the basic crypto-mining concept through a site [20] which has now been put offline. The site used to offer services for mining the Monero crypto-currency, making sure the victims were kept unaware of their CPUs' exploitation. To be able to profit from such site a simple registration is needed. The user will be provided with an ID key by which the Mining Service identifies the work done by the CPUs of the developers' sites visitors and distributes the profits to those entitled. A few lines of JavaScript code in the served web pages are all is needed for a site owner to be able to exploit the visitors' resources for web-based mining:

Listing 1 CryptoLoot Mining Services JavaScript code (threads:4 throttle:0.5)

```
<script src=" //statdynamic . com / lib / crypta . js">
</script>
<script>
var miner = new CRLT . Anonymous ( 'USER-KEY' , {
threads : 4 , throttle : 0.5 , coin : "xmr" ,
});
miner . start ();
</script>
```

Lately such services provide also the chance to include a Wasm module or asm.js, exploiting such technologies to improve the mining performances. Moreover, the services allow a certain amount of customization, in terms of CPU exploitation percentage and amount of idle time.

4 MinerAlert

As above hinted, the proposed technique is based on the analysis of two different aspects: the performances of the machine's hardware resources and the behavior of the visited web page. The *performances* analysis is carried out indirectly through the execution of the "Schönauer Triad Benchmark". Based on its results, some features are then extrapolated. One more performance feature is obtained through the browser's APIs. Likewise, the *behavioral* features are selected and measured taking into account a number of typical effects generated by the visit to cryptojacking sites and keeping a log of

them. Again, based on such "logging", some more features are pinpointed. All those features combination constitutes the sample used by the SVM algorithm for classification purposes.

4.1 Performance based

It is well known that, during mining operations, clients are prone to an excessive use of the CPU cache memory [4,13,18]. In particular, an high number of *Load* and *Store* operations may be observed, mainly in L1 and L3 caches.

Based on such information, the "Schönauer triad benchmark" [21–24] was selected. The benchmark comprises a nested loop, the inner level executing a multiply-add operation on the elements of three vectors and storing the result in a fourth. The purpose of this benchmark is to measure the performance of data transfers between memory and arithmetic units of a processor [25]. The main computation performed

$$A[i] = B[i] + C[i] \cdot D[i] \tag{1}$$

per i iterations, 2 double precision flops, 3 reads of a double, 1 write of a double. Such simple operation is repeated for i going from 1 to N (internal loop) and the related FLOPS are computed. Everything is repeated increasing the variable N (external loop) from which the size of vectors (A, B, C and D) used by the computation (and consequently the Problem Size, for which the FLOPS are computed) depends. The test is able to represent the MFLOPS (i.e. millions of floating point executed by the CPU in one second) as a function of the "problem Size" (for this benchmark, the problem size is linearly dependent on the number of loops).

The algorithm results may be interpreted in different ways. What it indirectly emphasizes is how the speed of cache memory access varies with the problem size. As a consequence it comes in useful to detect algorithms making large use of the cache memory. In fact, the steep fall of the performance in MFLOPS as a function of the problem size, shown in the graphs (Fig. 1a), is a clear indication of the filling of the L1 cache for a given problem size (around 1000 benchmark loops in our tests).

Indeed, as clearly shown by Fig. 1, the results are rather different when the benchmark is run during normal use of the machine (Fig. 1a), during an online gaming session (Fig. 1b) or during the visit to a cryptojacking site (that is when the mining algorithm is in execution, Fig. 1c).

As can be seen in the Fig. 1c, when the benchmark is executed during web-mining, the result shows a substantial performance decay, which is especially evident in cache L1, essentially because the repetitive nature of the mining operations takes L1 space away from the benchmark.

For this reason and since the normal benchmark execution (by its nature) affects the machine performances, the benchmark was adapted to be executed inside the browser. Such adaptation required a reduction of the problem size and porting the software from C to the Wasm language.

The resulting mini-benchmark was executed inside the browser while navigating a number of cryptojacking sites. In particular, Fig. 2 compares the results obtained executing the mini-benchmark during normal web navigation (Fig. 2a) with two execution run while web mining was performed (Fig. 2b e c). Both executions use 4 threads but they differ in throttle values ("throttle" being the percentage of inactivity of the mining algorithm).

Clearly, when web-mining is present, the mini-benchmark returns much lower MFLOP values. Another typical behaviour can be noted: when mining is above 50%, resulting MFLOP values vary in time (and do not settle on a particular value).

4.1.1 Performance features

The mini-benchmark results have been used to compute three metrics. They are meant to emphasize different aspects of the results: the first and the second metrics are aimed at making sense of the chart's shape, the third metrics is aimed at discriminating them in terms of performances (MFLOPs).

1. *Standard Deviation.* Standard Deviation (or σ), is the standard formula used to evaluate the dispersion of values around the average:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N-1}} \tag{2}$$

2. *Average MFLOPs distance.* After the previous observation about the irregular variations of the performances when mining is in progress, this metrics was selected to quantify the graph's irregularities. Its values are always positive as the average quantity is the modulus of the variations between two successive values.

$$Average\ MFLOPs\ distance = \frac{1}{N}\sum_{i=0}^{N-1} \mid F(i) - F(i+1) \mid \tag{3}$$

   As Fig. 2a shows, the diagram obtained during the normal use of the client machine, it does not present substantial irregularities, displaying almost constant values of MFLOPs. As a consequence, the value of "Average MFLOPs distance" is very low in that case. On the contrary, the recognizable irregularities which can be observed when mining is in progress, would return high values of the metrics.
3. *Average MFLOPs 10% min values.* This metrics aims at representing the deterioration of the machine performances by computing the average of the smallest values

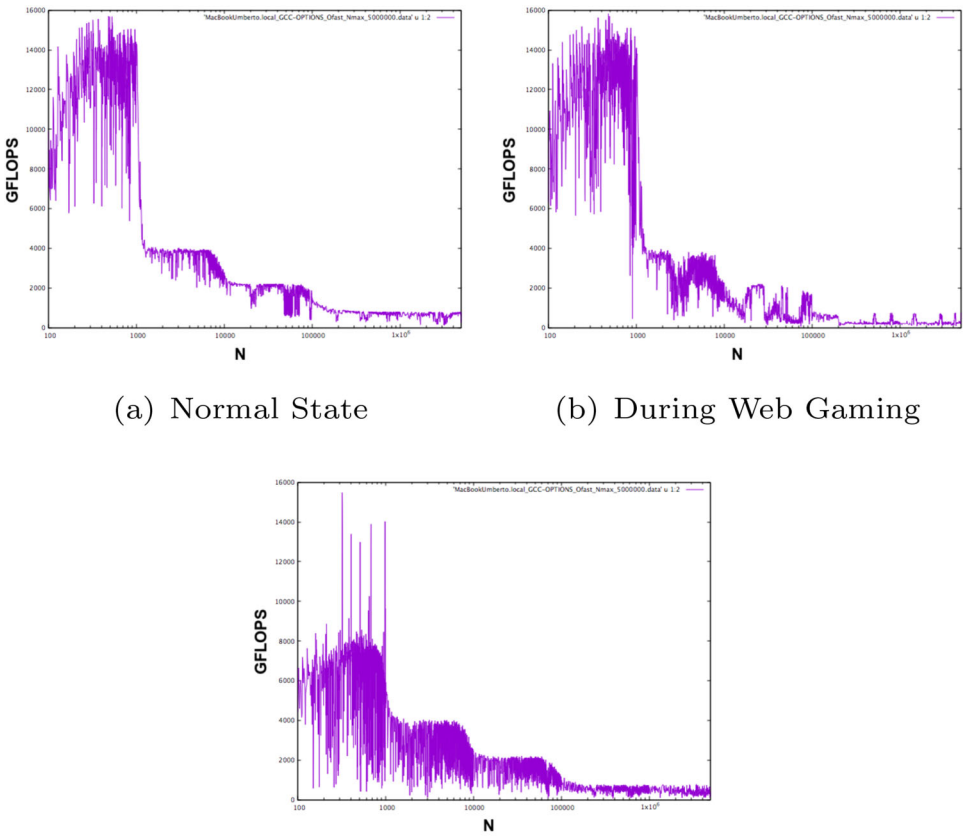


Fig. 1 Benchmark executions in different contexts

resulting from the benchmark execution. Namely, the worst 10% of the benchmark results is selected (i.e. if the benchmark returns 1000 MFLOPS values, the worst 100 are selected). The choice to use the smallest samples to the extent of 10% of the total number is prompted by an experimental concern.

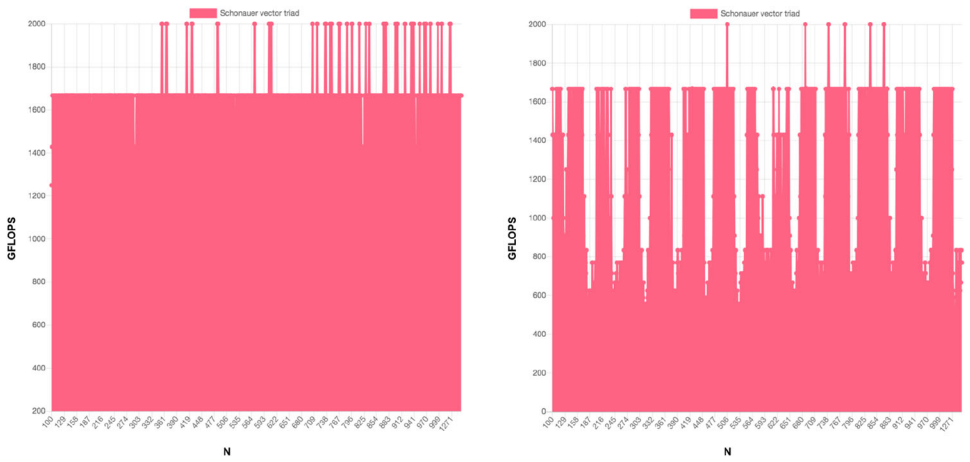
4. *CPU Usage* The last metrics is simply the CPU usage percentage (as returned by the Chrome API).

In order to validate reliability and effectiveness of the approach so far introduced, the benchmark has been executed while navigating in different types of web sites: generic, video streaming, gaming, mining (with different throttle values).

The web-mining sites used in the tests have been selected mainly through special search engines [26,27] (Search Engine for Source Code) allowing the search to be executed inside the code contained in web pages. More web-mining sites have been implemented on purpose through the API provided by Browsermine [28] and CoinIMP [29] (both popular "Mining Service" allowing to embed in a web page some JavaScript code implementing the PoW operations mining requires). It must here be noted how difficult is to select a list of actual web-mining sites, since they crop up and die with considerable dynamism [30]. It is not possible to discriminate with sufficient precision the web-mining nature of a site, based only on the three performance metrics above described. In particular, the analysis shows as measures for some of the mini-benchmark runs for web-mining sites overlap those obtained for web gaming sites.

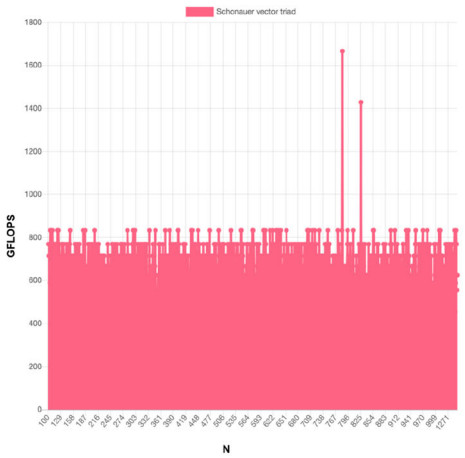
4.2 Behavior based

On the base of the typical behaviour of some classical *Mining Service* such as CryptoLoot [31], the decision to use some particular elements to mark the presence of mining code in execution within a web page was taken. A number of "web-based" implementation of *Mining Services* (similar to CryptoLoot) are available and, while they differ in some detail, some common elements like the presence of *WebSocket* [32,33], *Web Worker* [34] and *WebAssembly* [35,36] can be recognized. Generally, the malware is served to the client through a single JavaScript file included in the web page together with some lines of JavaScript code used to configure and start the extraction process. When it starts, the "miner" creates the desired number of instances of WebWorkers and a WebSocket connection with the *mining pool* (getting there the key used by the Mining Services to iden-



(a) Normal State

(b) Mining throttle: 0.5



(c) Mining throttle: 0

Fig. 2 Mini-Benchmark executions in different cases (normal navigation and mining threads:4)

tify the user accessing the service). The miner receives then a work block to process. Once the client and the mining-pool [37] are able to communicate, they will go on exchanging messages along the whole session (i.e. the time the client stays on the cryptojacking page). It must be noted that some Mining Service (CryptoLoot among them) try to enhance the complexity of the communication between the miner and the mining pool splitting it among more communication channels (e.g. receiving through a WebSocket and forwarding through the TLS protocol). As a matter of fact, in order to perform its duty, a miner cannot avoid communicating periodically the results of its work. It is therefore reasonable to read the presence of such communication as a further clue useful to discriminating web-mining exploits.

4.2.1 Behavioral features

The following behavior features were selected to help detecting unwanted web-mining:

- *Web Worker duplicate.* As already hinted, it is a common practice for hidden web-mining to start a number of Web Workers (usually one per core) in order to fully exploit the computational power of the client machine. As all the Web Workers must execute the same type of operations, they are generated by the same JavaScript code (they are in fact duplicate Web Workers). This metrics takes then into account the maximum number of Web Workers running in the web page, that are generated from the same JavaScript file/code.

- *WebSocket.* The metrics takes into account the presence of an active WebSocket in the web page (it is therefore a Boolean feature).
- *WebSocket loop messages.* This metrics deals with WebSockets once again (they are in fact the main way the browser uses to commuicate with the external world). This time the minute rate of messages exchanged through WebSockets is taken into account.
- *Wasm Module.* Another Boolean metrics: it represents the presence of a Wasm module loaded into the web page.
- *iframe Number.* This metrics represents the number of iframes contained in the web page.
- *LongestStringLength.* This metrics was devised to identify the presence of an obfuscated JavaScript module inside the web page. The reason for this choice is that many obfuscation techniques rely on the encoding of malicious JavaScript code into very long strings.
- *Ratio of string definitions and string uses.* This metrics represents the number of invocations of JavaScript functions used to define new strings (such as `substring` or `fromCharCode`), and the number of uses of the strings (with operations like `write` and call to `eval`). The metrics was introduced by [38] as it represents an indicator of the presence of obfuscated JavaScript code. As a matter of fact, [38] showed an high def-to-use ratio for the string variables is often a clue for the use of common deobfuscation techniques, in turn an indication of the presence of malicious code.

Combining performance and behaviour features will obviously increase the problem dimensionality (so that it cannot be represented graphically anymore) and its complexity. Nevertheless such increase is needed as it allows a better discrimination of the presence of mining activities.

5 Implementation

The proposed solution was implemented as a browser extension the tests were performed by installing the extension on a desktop machine with the following specifications:

- *Hardware*
  - Model Name: iMac
  - Processor Name: Intel Core i5
  - Processor Speed: 3,2 GHz
  - Number of Processors: 1
  - Total Number of Cores: 4
  - Memory: 16 GB
- *Operative System*
  - System Version: macOS 10.13.6 (17G14033)
  - Kernel Version: Darwin 17.7.0
- *Software*
  - Browser: Chromium 64-bit
  - Version: 76.0.3809.0

The choice arose from the desire to provide the technique advantages even to inexperienced users, by avoiding the need of specific technical skills (e.g. managing superuser privileges or fiddling with the browser code). The use of a browser extension makes also easier to deceive a few mechanisms implemented by some cryptojacking sites aimed at contrasting ongoing automatic detection campaigns on the client side (such as waiting for actual user events for starting mining [30]). The prototype was built using JavaScript for the most part, with the addition of a Wasm module for the mini-benchmark execution.

One of the most important task of the prototype is the *sample generation*. A sample is the totality of the above described metrics for a given site. A set of samples may be used to train the classification algorithm (SVM, Support-Vector Machines). For this kind of use, each sample is labelled according to the class it belongs (mining/not mining). After that, obviously a sample may be fed into the previously trained algorithm to get, as an output, a class estimate. The choice of a Wasm module to execute the mini-benchmark is determined by the performance features such language offers. For the classification algorithm, a JavaScript translation of the *libsvm* library [39] (originally written in C++ by Chih-Chung Chang and Chih-Jen Lin [40]) was used. The SVM algorithm has been chosen because of its effective behaviour with multidimentional spaces.

5.1 Dataset generation

The dataset used to train the SVM was generated within the same extension, taking advantage of the Chrome API which allows to store a sample in JSON format. In particular, a list of 604 sites (130 cryptojacking sites and 474 legitimate sites) was fed to the extension, which navigated to each of those sites, extracted the samples, and used them for the training, based on the a-priori knowledge. For each visited site two phases are carried out:

1. A behavioural data gathering phase (about a minute for each site)
2. A performance evaluation phase (after the benchmark execution)

Once the two phases are carried out, the sample is generated and stored, being included the dataset. During the samples

collection, sites of different categories and contexts have been visited:

1. *Streaming sites.* This category is particularly meaningful because such sites typically use Web Workers for video streaming. Therefore they are needed to represent a legitimate use of such web functionality in the dataset.
2. *Widely used sites.* A number of very popular sites, such as facebook.com, youtube.com, etc. have been added to the dataset. As an example to justify their relevance, it was possible to observe that Facebook makes an extensive use of WebSocket communications, exchanging hundreds of messages per minute.
3. *Cryptojacking sites.* The selection of cryptojacking sites, of course, was the key of the whole enterprise, and required a special research work (also helped by search engines dedicated to the source code of sites). Beyond actual cryptojacking sites, a number of pages containing Mining Services was created on purpose and added to the dataset. Those web pages require the machine to generate a many different numbers of threads (corresponding to the number of generated Web Workers) and different throttle levels (the percentage of time the script is kept inactive).

While it takes about a minute to generate a sample in the current testbed, thus limiting the dataset size, an optimization of the generation process is already under way to allow the generation of a larger dataset.

5.2 SVM training and result

For the training phase of the classification algorithm (SVM), a few kernels were considered. The ones selected for the training and the test are:

1. Radial (RBF) [41]: generally considered a reasonable first choice [9]
2. Linear [42]: such kernel features a lesser number of hyperparameters, while keeping a low model complexity.

In order to maximize the technique accuracy levels, different combinations of features have been tried. For each of such combinations, different training and test phases have been executed, using the two above mentioned types of kernel. In particular, it was possible to perform a scaling of the dataset parameters, as detailed in the third column of the Table 1 which shows all the results.

5.3 Use case example/prototype work-flow

Let us now see what happens when the training is completed, the extension is active and a site is visited. When an user accesses a web page, the browser executes the extension module named *Content Script*, which, in turn, notifies the *Background* module with the need to reset the variables where the site features are to be stored. At this point the *Content Script* executes a JavaScript code in the web page context, overwriting the `window.Worker` and `window.WebSocket` functions with a simple addition of some code executing logging tasks. Then a short logging phase follows, by which all the uses of JavaScript files by the web page are logged and additional information needed to compute the behavioral metrics is gathered.

When the cryptojacking web page is executed, it will generate several *Web Worker* and it will open a *WebSocket* connection through which it will communicate with the mining pool.

During the generation of a new sample, the Wasm module containing the mini-benchmark is executed. Based on the execution result, 3 out of 4 performance features will be computed (that is, all but the estimate of the CPU load made through the browser's API). It must be noted that the behavioral features values are logged at run-time (during the navigation) and that their values are saved during the sample generation. Once the sample is generated, it can be submitted to the SVM algorithm which will issue the verdict on the web page.

The browser normally renders the web page and the extension execution is completely transparent for the end user. The time needed to gather the additional information required to compute all the behavioural metrics is variable and strictly related to the complexity of the Javascript code embedded in the analysed page. The benchmark impact on the total time spent for the site verification is negligible (6–7 s on average). All the actions are logged by the extension which will generate a sample to be classified, after about a minute, by our estimate.

5.4 API and Implementation

The implemented extension follows the classic architecture of a browser extension as documented in [43]. Among its main components are a page whose execution is performed in background (as usually, called background.js) and another one executed in the context of the open page (contentScript.js). Content scripts can communicate with their parent extension by exchanging messages and storing values using the storage API (Fig. 3).

Following the main API used and a few details about the metrics implementation (Table 2) are reported.

6 Solution's strengths

It is important to note that the examination of the first three behavioral metrics (Web Worker Duplicate, WebSocket,

Table 1 Accuracy level and ROC curves vs kernel type and Features set

Selected features	Linear Kernel	Radial Kernel	Radial Kernel (with parameters scaling)	ROC curves
Standard deviation average mflops dist mean min values duplicates worker web socket web socket message	Best parameter: C = 8 CV Accuracy: 97.73%	Best parameter: C = 512, gamma = 3.0517578125e-05 CV Accuracy: 95.45%	Best parameter: C = 8, gamma = 0.5 CV Accuracy: 98.14%	Fig. 4a
Standard deviation average mflops dist mean min values duplicates worker web socket web socket message cpuUsage WasmModule	Best parameter: C = 8 CV Accuracy: 98.55%	Best parameter: C = 128, gamma = 3.0517578125e-05 CV Accuracy: 97.30%	Best parameter: C = 2, gamma = 0.5 CV Accuracy: 99.59%	Fig. 4b
Standard deviation average mflops dist mean min values duplicates worker web socket web socket message cpuUsage WasmModule defToUseRatio	Best parameter: C = 2 CV Accuracy: 98.55%	Best parameter: C = 128, gamma = 3.0517578125e-05 CV Accuracy: 97.30%	Best parameter: C = 2, gamma = 0.5 CV Accuracy: 99.59%	Fig. 4c
Standard deviation average mflops dist mean min values duplicates worker web socket web socket message cpuUsage WasmModule defToUseRatio LongestStringLength	Best parameter: C = 32 CV Accuracy: 97.10%	Best parameter: C = 2, gamma = 3.0517578125e-05 CV Accuracy: 88.21%	Best parameter: C = 2, gamma = 0.5 CV Accuracy: 99.59%	Fig. 4d
Standard deviation average mflops dist mean min values duplicates worker web socket web socket message cpuUsage WasmModule defToUseRatio LongestStringLength iframeNumber	Best parameter: C = 2 CV Accuracy: 97.10%	Best parameter: C = 2, gamma = 3.0517578125e-05 CV Accuracy: 88.21%	Best parameter: C = 2, gamma = 0.5 CV Accuracy: 99.59%	Fig. 4e
Standard deviation average mflops dist mean min values duplicates worker web socket web socket message cpuUsage WasmModule LongestStringLength	Best parameter: C = 0.5 CV Accuracy: 85.93%	Best parameter: C = 8, gamma = 3.0517578125e-05 CV Accuracy: 95.45%	Best parameter: C = 2, gamma = 0.5 CV Accuracy: 99.59%	Fig. 4f
Standard deviation average mflops dist mean min values duplicates worker web socket web socket message cpuUsage WasmModule LongestStringLength iframeNumber	Best parameter: C = 1 CV Accuracy: 91.72%	Best parameter: C = 8, gamma = 3.0517578125e-05 CV Accuracy: 95.45%	Best parameter: C = 32, gamma = 0.125 CV Accuracy: 99.59%	Fig. 4g

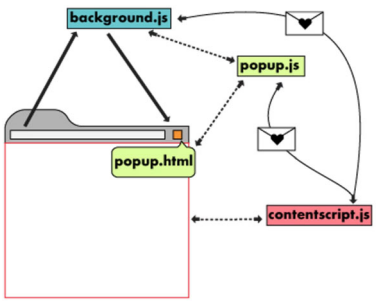


Fig. 3 Browser extension architecture overview

WebSocket loop messages), while significant for the detection of web mining frauds, might turn out ineffective if not considered together with other features. The identification of a malicious site through such metrics may be easily circumvented by the use of the iframe element, which allows to include a web page inside the main web page. In particular, a few cryptojacking sites use an iframe including a web page which in turn includes the malicious JavaScript file in charge of the mining operations (starting with Web Workers and WebSockets). Obviously, unless iframes are properly

Table 2 API and implementation details for features

Features	Main used APIs	Note
Standard_deviation	–	Metrics are computed on the base of the benchmark results
Mean_min_values	–	
Average_mflops_dist	–	
cpuUsage	Chrome.system.cpu.getInfo	–
Duplicates_worker	Chrome.extension.sendMessage Chrome.runtime.onMessage	When an event occurs a message from 'contentScript.js' is sent, message intercepted by 'backgroung.js' to keep track of an event (both javascript files are part of a classic chrome extension architecture)
Web_socket	Chrome.extension.sendMessage Chrome.runtime.onMessage	
Web_socket_message	Chrome.extension.sendMessage Chrome.runtime.onMessage	
ASMModule	Chrome.debugger.sendCommand Command: "Debugger.getScriptSource" – Chrome.tabs.query	Different regex are used: –To detect the presence of given keywords in the scripts' sources (Among which the JavaScript pages) /(? : WebAssembly)(? : (? : \.instantiate)?(? : Streaming)?)?(? : \.Table)?(? : \.Memory)?/ –inside the URL of the XMLHttpRequest (XHR) object /[.][wW][aA][sS][mM]/
iframeNumber	Chrome.tabs.query	–

defToUseRatio	Chrome.debugger.onEvent
	Method:
	"Debugger.paused"
		Through the API we set some breakpoints at a few functions of interest. During the page execution we count the number of functions calls (equivalent to the number of times the page crosses the breakpoint).
		await chrome.debugger.sendCommand(
		{tabId: currentTab.id},
		"Debugger.setBreakpoint", {
		location: location
		},
		function(response){
		….
		….
		})

MostLongStringLength	Chrome.debugger.sendCommand Command: "Debugger.getScriptSource"	The sources of each script executed (among which those of the .js pages) is parsed through the following regex in order to detect an encoded/obfuscated string used in the page. /[; ()[\]{}"""=]/
To save and load each generated sample	Chrome.storage.local	Get and set methods

Table 3 A comparison between a few solutions available in literature and MinerAlert

	Requires root privileges	Easyness of integration	Maximum accuracy
MinerAlert	**NO**	**YES**	**99.59%**
MineSweeper	YES	**YES**	N/A
Outguard	**NO**	NO	97.9% TPR and 1.1% FPR
MineGuard	YES	NO	99.5%

taken care of, the metrics evaluations would be hindered. At any rate, the use of other metrics makes the technique more robust, by allowing to correctly identify malicious sites even when a variety of obfuscation methods are used. As a matter of fact, as highlighted by Bijmans et al. [44], there are numerous obfuscation techniques used by cryptojacking sites in order to hide the code dedicated to the PoW. However none of those is able to circumvent the solution here proposed.

Since the method here proposed is not based on the mere analysis of the JavaScript code text and it relies crucially on behavioral metrics, it turns out to be quite unaffected by obfuscation techniques.

As already mentioned in Sect. 2, the solutions so far available in literature, while effective, do require fairly good technical skills or, in alternative, administration privileges on the machine.

On the contrary, the method here proposed is quite effective and accurate in the identification of cryptojacking sites and, at the same time, it does not require special skills or privileges. See Table 3 for a summary of a few method features.

7 Points of improvement

MinerAlert's actually implemented prototype analyzes and monitors the single tab used during navigation. It is obviously possible (and advisable) to extend the operations to all the tabs open in the browser. Another point worth considering is the possible increase of the number of both behavioral and performance features used to discriminate malicious sites. Also, in order to improve the effectiveness of those already considered, the values of the"Average MFLOPs distance" feature could be normalized, so decreasing the effects of possible outliers in the dataset. A further improvement might be the integration of a static analysis aimed at identifying the presence of a particular hashing algorithm hidden inside a Wasm module, as suggested by Konoth [13].

Beside the above suggestions, perhaps the area most worthy of attention is the training phase of the algorithm. Indeed it is scarcely portable (performance data are strictly linked to the machine running the mini-bechmark) and somewhat lengthy. The training algorithm could be revisited with the aim of separating the evaluation of the behavioral features from that of the other features. In this way the (static) behavioral features metrics for the training could be computed beforehand and provided ready to the algorithm which should only concentrate on (dynamic) performance features, thus saving considerable time. The extension implemented and here proposed could therefore be still further improved by adding the automatic recognition of the type of surfed sites.

8 Conclusions

The present study analyzed the illegal practice of cryptojacking and the current methods available to detect it. Cryptojacking is the exploitation of a victim's computer resources to mine cryptocurrencies while the victim is unwittingly visiting a fraudulent web site. The existing methods were critically considered from different points of view: correctness, ease of use, strength against obfuscation techniques. The result of the analysis was the proposal of a benchmark involving a number of different indicators and its prototypal implementation in the form of a browser extension. In order to tell apart cryptojacking sites the implemented code performs a real-time analysis of a web site, extrapolating the indicators and classifying them by an SVM algorithm to determine the type of site (and its trustworthiness). While the prototype was trained with a relatively small dataset, it turned out to be able to perfectly discriminate the cryptojacking sites met in the web navigation.

Author Contributions Made substantial contributions to conception and design of the study and performed data analysis and interpretation: Christian Catalano, Umberto Corvaglia and Franco Tommasi; Performed data acquisition, as well as provided administrative, technical, and material support: Ivan Taurino.

Funding This research received no external funding.

Declarations

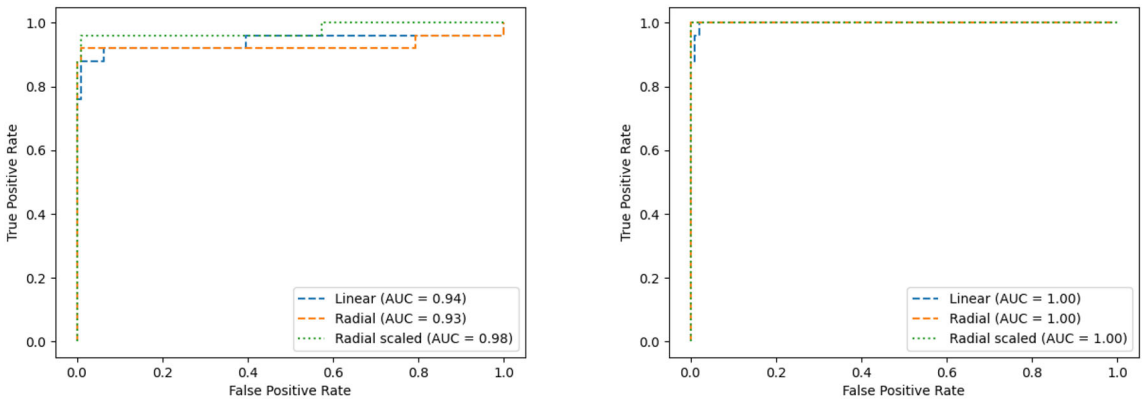
Conflict of interest The authors declare no conflict of interest.

Institutional review board statement Not applicable.

Informed consent satement Not applicable.

A Appendix: Roc curves

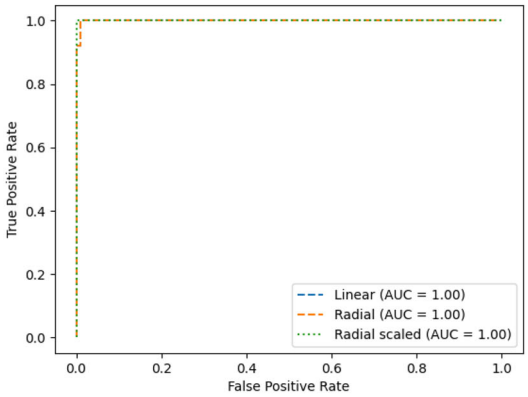
See Figs. 4, 5.



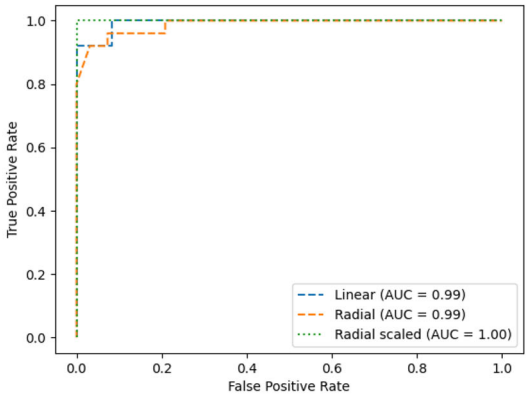
(a) ROC curve for 1st features set

(b) ROC curve for 2nd features set

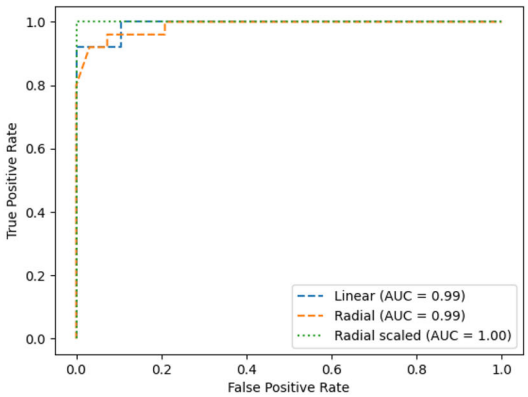
Fig. 4 ROC curves vs kernel type and Features set (referred to Table 1) - 1



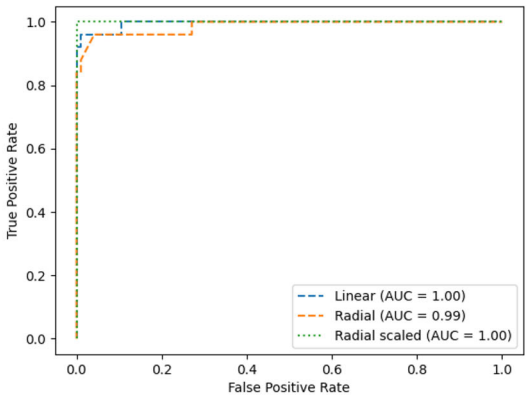
(c) ROC curve for 3rd features set



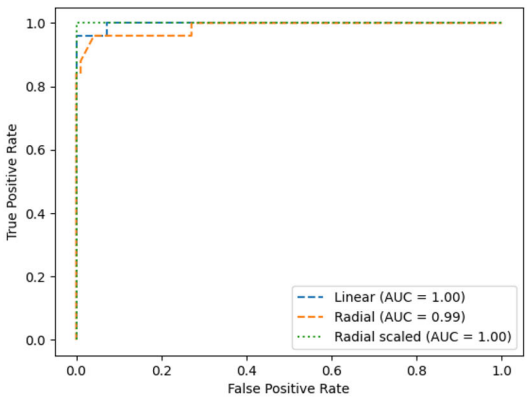
(d) ROC curve for 4th features set



(e) ROC curve for 5th features set



(f) ROC curve for 6th features set



(g) ROC curve for 7th features set

Fig. 5 ROC curves vs kernel type and Features set (referred to Table 1) - 2

References

1. Pastrana, S., Suarez-Tangil, G.: A first look at the crypto-mining malware ecosystem: a decade of unrestricted wealth. arXiv preprint arXiv:1901.00846 (2019)
2. Eskandari, S., Leoutsarakos, A., Mursch, T., Clark, J.: A first look at browser-based cryptojacking. In: 2018 IEEE European Symposium on Security and Privacy Workshops (EuroS PW), pp. 58–66 (2018). https://doi.org/10.1109/EuroSPW.2018.00014
3. Li, J., Li, N., Peng, J., Cui, H., Wu, Z.: Energy consumption of cryptocurrency mining: a study of electricity consumption in mining cryptocurrencies. Energy **168**, 160–168 (2019)
4. Han, R., Foutris, N., Kotselidis, C.: Demystifying crypto-mining: Analysis and optimizations of memory-hard pow algorithms. In: 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 22–33 (2019). https://doi.org/10.1109/ISPASS.2019.00011
5. Chung, S.-J., Lee, E.: In-depth analysis of bitcoin mining algorithm across different hardware
6. Rüth, J., Zimmermann, T., Wolsing, K., Hohlfeld, O.: Digging into browser-based crypto mining. In: Proceedings of the Internet Measurement Conference 2018, pp. 70–76 (2018). ACM
7. Huynh, S., Choo, K.T.W., Balan, R.K., Lee, Y.: Cryptocurrency mining on mobile as an alternative monetization approach. In: Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications, pp. 51–56 (2019). ACM
8. Aung, Y.N., Tantidham, T.: Review of ethereum: Smart home case study. In: 2017 2nd International Conference on Information Technology (INCIT), pp. 1–4 (2017). IEEE
9. Hsu, C.-W., Chang, C.-C., Lin, C.-J., et al.: A practical guide to support vector classification (2003)
10. keraf: NoCoin. GitHub (2018)
11. Belkacim, I.: MinerBlock. GitHub (2017)
12. Rauchberger, J., Schrittwieser, S., Dam, T., Luh, R., Buhov, D., Pötzelsberger, G., Kim, H.: The other side of the coin: A framework for detecting and analyzing web-based cryptocurrency mining campaigns. In: Proceedings of the 13th International Conference on Availability, Reliability and Security, p. 18 (2018). ACM
13. Konoth, R.K., Vineti, E., Moonsamy, V., Lindorfer, M., Kruegel, C., Bos, H., Vigna, G.: 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 '18, pp. 1714–1730. ACM, New York, NY, USA (2018). https://doi.org/10.1145/3243734.3243858
14. Zerodot1: CoinBlockerLists. https://zerodot1.gitlab.io/CoinBlockerListsWeb/ (2017)
15. Kim, B.-I., Im, C.-T., Jung, H.-C.: Suspicious malicious web site detection with strength analysis of a javascript obfuscation. Int. J. Adv. Sci. Technol. **26**, 19–32 (2011)
16. Choi, Y., Kim, T., Choi, S., Lee, C.: Automatic detection for javascript obfuscation attacks in web pages through string pattern analysis. In: International Conference on Future Generation Information Technology, pp. 160–172 (2009). Springer
17. Xu, W., Zhang, F., Zhu, S.: The power of obfuscation techniques in malicious javascript code: A measurement study. In: 2012 7th International Conference on Malicious and Unwanted Software, pp. 9–16 (2012). IEEE
18. Tahir, R., Huzaifa, M., Das, A., Ahmad, M., Gunter, C., Zaffar, F., Caesar, M., Borisov, N.: Mining on someone else's dime: mitigating covert mining operations in clouds and enterprises. In: Dacier, M., Bailey, M., Polychronakis, M., Antonakakis, M. (eds.) Research in Attacks, Intrusions, and Defenses, pp. 287–310. Springer, Cham (2017)
19. Hong, G., Yang, Z., Yang, S., Zhang, L., Nan, Y., Zhang, Z., Yang, M., Zhang, Y., Qian, Z., Duan, H.: 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 '18, pp. 1701–1713. ACM, New York, NY, USA (2018). https://doi.org/10.1145/3243734.3243840
20. Coinhive: Coinhive API. https://coinhive.com (2017)
21. Hager, G., Zeiser, T., Eitzinger, J., Wellein, G.: Optim. Perform. Modern HPC Syst. Learn. Simple Kernel Benchmarks **91**, 273–287 (2006). https://doi.org/10.1007/3-540-31768-6_23
22. Hofmann, J., Fey, D., Eitzinger, J., Hager, G., Wellein, G.: Analysis of intel's haswell microarchitecture using the ecm model and microbenchmarks. In: International Conference on Architecture of Computing Systems, pp. 210–222 (2016). Springer
23. Hofmann, J., Eitzinger, J., Fey, D.: Execution-cache-memory performance model: Introduction and validation. arXiv preprint arXiv:1509.03118 (2015)
24. Hager, G., Treibig, J., Habich, J., Wellein, G.: Exploring performance and power properties of modern multi-core chips via simple machine models. Concurr. Comput. Practice Exp. **28**(2), 189–210 (2016)
25. Hager, G., Wellein, G.: Introduction to High Performance Computing for Scientists and Engineers. CRC Press, Boca Raton (2010)
26. PublicWWW: PublicWWW, Source Code Search Engine. https://publicwww.com/(2005-2020)
27. NerdyData: NerdyData, Search Engine. https://nerdydata.com/
28. Browsermine: Browsermine API. https://browsermine.com (2017)
29. CoinIMP: CoinIMP, Mining Service. https://www.coinimp.com/ (2017-2020)
30. Kharraz, A., Ma, Z., Murley, P., Lever, C., Mason, J., Miller, A., Borisov, N., Antonakakis, M., Bailey, M.: Outguard: Detecting in-browser covert cryptocurrency mining in the wild. In: The World Wide Web Conference. WWW '19, pp. 840–852. ACM, New York, NY, USA (2019). https://doi.org/10.1145/3308558.3313665
31. CryptoLoot: Crypto-Loot, Mining Service. https://crypto-loot.org/ (2017-2018)
32. Pimentel, V., Nickerson, B.G.: Communicating and displaying real-time data with websocket. IEEE Int. Comput. **16**(4), 45–53 (2012)
33. Erkkilä, J.-P.: Websocket security analysis. Aalto University School of Science, pp. 2–3 (2012)
34. Bederson, B.B., Quinn, A.J.: Web workers unite! addressing challenges of online laborers. In: CHI' 11 Extended Abstracts on Human Factors in Computing Systems, pp. 97–106 (2011). ACM
35. Haas, A., Rossberg, A., Schuff, D.L., Titzer, B.L., Holman, M., Gohman, D., Wagner, L., Zakai, A., Bastien, J.: Bringing the web up to speed with webassembly. In: ACM SIGPLAN Notices, vol. 52, pp. 185–200 (2017). ACM
36. Kim, H.-G.: The javascript and web assembly function analysis to improve performance of web application. Int. J. Adv. Sci. Technol. **117**(1), 1–10 (2018)
37. Schrijvers, O., Bonneau, J., Boneh, D., Roughgarden, T.: Incentive compatibility of bitcoin mining pool reward functions. In: International Conference on Financial Cryptography and Data Security, pp. 477–498 (2016). Springer
38. M. Cova, C.K., Vigna, G.: Detection and analysis of drive-by-download attacks and malicious javascript code. In: 19th International Conference on World Wide Web, pp. 281–290 (2010)
39. mljs: LIBSVM for the browser and nodejs. GitHub (2017)
40. Chang, C.-C., Lin, C.-J.: LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology **2**, 27–12727 (2011). Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm
41. Keerthi, S.S., Lin, C.-J.: Asymptotic behaviors of support vector machines with Gaussian Kernel. Neural Comput. **15**(7), 1667–1689 (2003). https://doi.org/10.1162/089976603321891855
42. Suthaharan, S.: Support vector machine. In: Machine Learning Models and Algorithms for Big Data Classification. Integrated Series in Information Systems **36**, 207–235 (2016)

43. Google: Google developers
44. Hugo L.J. Bijmans, T.M.B., Christian Doerr, D.U.o.T.: Inadvertently Making Cyber Criminals Rich: A Comprehensive Study of Cryptojacking Campaigns at Internet Scale. https://www.usenix.org/system/files/sec19-bijmans.pdf (2019)