Browser-Based Deep Behavioral Detection of Web Cryptomining with CoinSpy

Conor Kelton
Stony Brook University

Aruna Balasubramanian
Stony Brook University

Ramya Raghavendra
IBM Research

Mudhakar Srivatsa
IBM Research

***Abstract*—Although the cryptocurrency hype over the past year may be seen by some as a benign social fad, to the Web community it is the center point for a series of ethically dubious ransomware attacks. Browser based cryptomining, or cryptojacking has gained widespread attention. Cryptojacking consists of Web servers delivering cryptocurrency mining scripts to clients, and using the client resources to play part in a distributed coin mining scheme. Although Web server operators defend the ethics of their involvement by quoting mining as a substitute for advertisement revenue, these scripts can hog massive amounts of client-side resources and can be delivered without client consent, presenting a high potential for abuse. Regardless of how ethical these campaigns are, what remains constant is the need for their detection. While there has been an array of work in defending against such cryptojacking campaigns, these defenses remain quite preliminary. We present CoinSpy, an entirely in-browser tool built using deep learning techniques for the detection of cryptomining activity within Web pages. A key challenge is that there is limited visibility into the client resource usage from within the browser sandbox. CoinSpy extracts several signals from information available from the browser and combines them using deep learning to build a powerful cryptojacking classifier. We argue why CoinSpy is the most robust defense against current and future cryptojacking attacks as compared to recent work, and show that it can detect various cryptojacking campaigns with 97% accuracy.**

I. Introduction

The Web is fast becoming a popular platform for cryptocurrency mining [37], [33], a process called "drive-by-mining" or *cryptojacking*. Cryptojackers exploit resources at the client device when the client is browsing the Web. When a client accesses a cryptojacking site, a specialized script is downloaded to the client computer, often without their knowledge or consent. This script runs in their browser in the background, exploiting their compute resources. There has been an 8500% increase in cryptojacking Web pages [40] and 0.06 - 0.08% of the Alexa Top 1 Million sites have been shown to be delivering cryptojacking scripts [21], [37]. This is largely made possible by the popularization of new families of cryptocurrency (e.g., Monero coin [25]) that enable mining for coins without custom hardware.

Detecting cryptojacking is critical to enable a positive and safe Web experience. Studies show that cryptojacking causes severe performance degradation [39]. Parties complicit to cryptojacking can include APIs for obtaining user consent [24] but these aren't often utilized in practice, making mining an ethical gray area [14], [39].

To date, there has been much work highlighting the presence, economic impact, and obtrusiveness, of Web mining [14], [37], [38]. There has also been an influx of recent works that focus on extracting features, both static and dynamic, of miners to build models for determining cryptomining behavior on Web pages [21], [16], [23].

Here, we present CoinSpy, a tool that focuses on the latter space of cryptojacking detection. To differentiate and improve this space, we design CoinSpy with two main goals.

Firstly, making CoinSpy a future-proof detector is our first-class concern. While many existing detection mechanisms offer well posed means for mining detection in the current landscape, this space remains quite volatile; the erratic nature of the pricing of Web cryptocurrencies has lead to bow-outs [9] from even major players in the space, such as CoinHive [24] which closed operations this past March of 2019. To enable future-proof detection, we learn a model for detection based entirely on *behavioral* features that learn the actual *effects* of mining on the client resources. Behavioral features have been shown to provide better generalization in the realm of malware detection, across a verity of fields [4], [12]. Behavioral features are robust to evasions [16] that are known to exist for static signatures, such as blacklisting, obtained from the these miners. We additionally introduce behavioral variations into the miners in which we train so that the detector will scale to the many different ways which miners are used in the wild. Finally, our model can be *incrementally updated* given that the volatility of this space may produce new miners and hence new behavioral effects on the client resources.

Secondly, we design CoinSpy to operate end-to-end from *within the browser*. An in-browser tool is critical to the deployment of such a system—it allows for in-browser defense upon detection, such as stopping a specific JavaScript, and it exhibits a higher level of trust for users [28]. Further, extension permission models [15] can assure users that we do not send any of their private data used for detection to the external network. Finally access to well known platforms such as the Chrome Web store ensure our defense can be easily accessed and kept up to date for users, helping its future-proof design.

CoinSpy analyzes the compute, network, and memory behaviors caused by cryptojackers running within client browsers. Our studies focus on both the individual effects of cryptojacking on these resources as well as their correlated utilizations. For example, mining requires solving a Proof-of-Work (or PoW) algorithm, a computationally intensive algorithm that heavily consumes compute resources. The CPU consumption exists in high amounts for prolonged bursts and is often performed across parallel threads, which is different

Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb) 2020
23 February 2020, San Diego, CA, USA
ISBN 1-891562-63-0
https://dx.doi.org/10.14722/madweb.2020.23002
www.ndss-symposium.org

compared to the CPU behavior during normal Web browsing. Similarly, PoW's run by mining scripts are memory-hard and make a well defined footprints in memory accesses. Web miners also distribute cryptojacking-related tasks over a bidirectional distribution protocol which makes for unique payload signatures over the network. Further, there is correlation between the network and compute activities—large CPU bursts are followed by brief periods of inactivity in which network resources are used to sync computational results of the mining to servers.

A challenge with our goal of implementing the behavioral detector from within the browser sandbox is that a browser only implementation severely limits visibility into system level resource usages. For example, the CPU L1 cache hits are shown to be a marker for memory-intensive cryptomining PoWs [23]; however, Web browsers do not have visibility into the L1 cache. This makes detection of the cryptojacking a more difficult task. Despite this challenge, we show how we can learn the signature of cryptojacking with only the resource usage information available from within the browser. To this end, CoinSpy builds its signatures by only monitoring the JavaScript stack execution times, network bytes sent/received, and JavaScript heap memory usage, all from within the browser.

CoinSpy collects samples of CPU, memory, and network behaviors and represents them as a timeseries to learn both individual and correlated patterns that point to cryptojacking, all from within the browser. Cryptojacking is a continuous activity that constantly executes PoWs to try and maximize blocks mined and coins earned (See §II). Having a continual measurement of the behavior of the system using a timeseries, versus a point measurement such as a threshold, gives additional power in detection.

There are state-of-the-art cryptomining detectors which can also operate in-browser and act on behavioral features [21], [16]. However the behaviors used by these detectors are either too preliminary or are overly specific. For example, CMTracker [16] uses a CPU thresholding to detect mining that is too rigid to be adjusted for in-the-wild behaviors. Outguard [21] is tied to specific implementation of Chrome and heavily relies on an implementation detail of miners that can be easily changed while still allowing mining to unfold. We explore these aspects in our evaluation of CoinSpy (Section V).

Another main challenge is in combining these time series data to extract a signature for cryptojacking. Combining multivariate time series data is lossy, and shallow learning models perform poorly due to the high dimensionality of the data. To address the shortcomings of shallow learning techniques such as the need for vectorization of multiple time series and using custom feature engineering, we design a deep Convolutional Neural Network (CNN). The construction of the CNN model is, in part, based on intuitions from our analysis of the features using shallow learning. Using the CNN model is also amenable to *incremental* learning to scale to new miners in the cryptojacking landscape.

We evaluate the performance of CoinSpy over multiple Web data sets—a test data set where we inject cryptomining scripts to a 5K benign Websites, a curated in-the-wild data set consisting of known cryptojacking sites, and a short overview of the top Alexa Websites. Our evaluation shows the high accuracy of CoinSpy to the current mining landscape, compares CoinSpy to the state of the art with a focus on why it is more future-proof, and shows the CoinSpy browser extension can detecting mining activity while making little impact on the user experience.

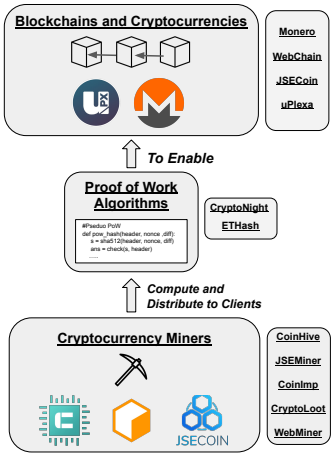


Figure 1: The main entities in Web mining. A cryptocurrency miner needs to solve a complex PoW puzzle to add a block to its blockchain. When the miner adds to the blockchain, it gets cryptocurrency as compensation.

II. Background

In this section, we describe how Web browsers are used to mine digital currency. The design of CoinSpy is based on an in-depth understanding of the mining ecosystem.

A. The Web mining ecosystem

A Web mining environment consists of three main entities, shown above in Figure 1. A given cryptocurrency has its own blockchain. The blockchain stores transaction information about the cryptocurrency that it relates to. In Figure 1, we present some examples of cryptocurrencies–Monero, JSECoin, uPlexa, and Webchain.

To add a new block to the chain requires solving a cryptographic puzzle which can be computed via a computationally hard algorithm called the Proof of Work (PoW). The reward for solving a PoW for a block in a given blockchain is some units of cryptocurrency represented by that blockchain.

There are many cryptocurrencies that exist today. Bitcoin, the most popular cryptocurrency, for example, uses a PoW algorithm that is 10,000× faster on GPU's and specialized hardware (ASICs) compared to commodity CPU's [10]. This kills any competition offered by commodity hardware.

However, there now exist coins that can be mined using *memory bound* hashing PoW algorithms. These PoW algorithms can be computed effectively on stock hardware like desktop-grade CPUs, allowing for profitable commodity mining. CryptoNight [10] and ETHash [44] are two examples of memory-bound PoWs. The commodity hardware used by these miners has extended to desktops, as made available through Web browsers.

Mining through Web browsers is a complex ecosystem. Miners (Figure 1) remain competitive by enlisting the help of Web servers in a collaboration known as a *mining pool*. The Web servers in-turn use the resources of their Web clients to mine coins.

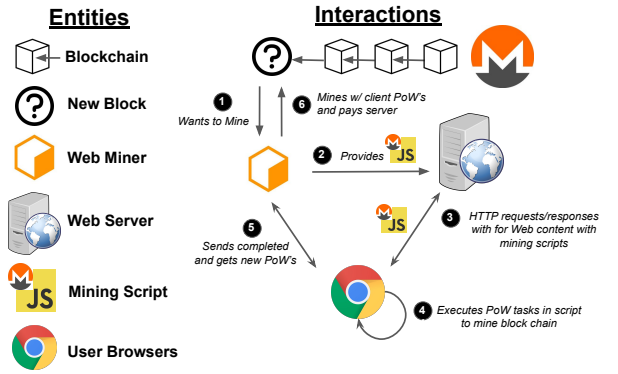


Figure 2: A step-by-step process of how cryptojacking unfolds. Cryptojacking takes place as a three way interaction between the Web miner, the Web server, and the client browser.

B. Web mining at the browser

Figure 2 shows, step-by-step, how miners run the PoWs at the Web clients. Web mining is a three-way interaction between Web miners, the Web servers, and the client's Web browser. A Web miner, often run by a third-party service such as CryptoLoot [9] or JSECoin [17], enlists a Web server to help find clients to compute PoWs; the PoW script is in the form of a JavaScript or even WebAssembly (WASM) [13].

When a browser connects to the Web server to download the Web page, the server sends this PoW script along with the Web content. The PoW script is then actually run on the client device. The script uses the client's compute resources to solve the PoWs and sends PoW guesses back to the Web miner. To increase the rate of PoW completions at the client, the miners typically spawn threads from the browser environment, known as WebWorkers [46]. The Web miner's servers, which actually manage the interactions with the blockchain, are often found from a rotating list of proxy servers as to avoid client-side blacklisting [21], [38]. The Web miner pays the Web server a cut of the cryptocurrency for any blocks successfully mined by the client (or a just fixed payout per compute cycles), similar in fashion to a finder's fee.

III. CoinSpy

CoinSpy accomplishes future-proof, in-browser cryptomining detection through: (i) identifying behavioral cryptomining signatures given our knowledge of the space and capturing three data sources, across compute, memory, and network, that represent them, (ii) combining the signatures, represented as timeseries features, as inputs to a (deep) cryptomining classifier, and (iii) incorporating incremental learning. Figure 3 provides an overview of our full system, fleshing out these three components.

We begin our discussion of the CoinSpy design by discussing the intuition for its features. We follow with a discussion of the CoinSpy deep learning model that uses these features to perform cryptojacking detection.

A. Compute Signature

Cryptojacking scripts require execution of PoW algorithms that are computationally intensive (See section II), occur in bursts to guess solutions to the cryptographic puzzles and sync the results to mining servers, and often execute in the

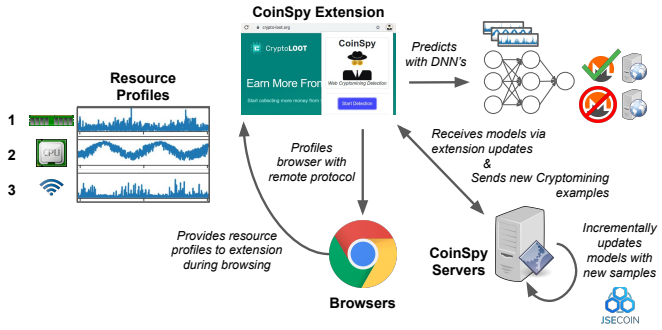


Figure 3: CoinSpy obtains behavioral browser datasources, such as CPU, Memory, and Network profiles, during live page loads on top of unmodified browsers. CoinSpy creates timeseries features from these behavioral data sources and uses them as inputs to a CNN to classify the presence of cryptojacking activity on a given page. The CoinSpy CNN model is incrementally updates the model.

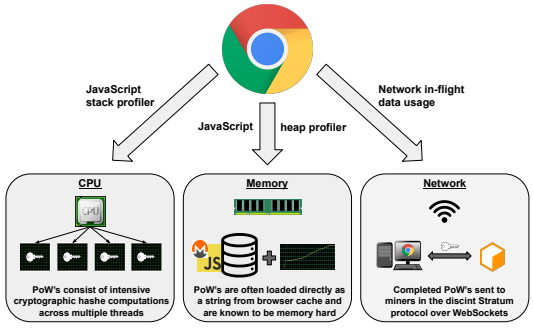


Figure 4: The behavioral impacts that the various parts of the mining ecosystem have from the perspective of client browsers.

context of WebWorkers in multiple threads. Computation is often optimized using very specific tools, such as asm.js or WebAssembly (WASM) [13]. In contrast, conventional JavaScript is executed over a single thread and is more event driven, with shorter computation periods interleaved with asynchronous network activities.

CoinSpy uses measures computational behavior through the browser exposed JavaScript stack profiler. Specifically, CoinSpy takes the stack height of the profiler on each thread and sums across all threads, each sample height represents a point in our compute timeseries. Our intuition here is that the PoW algorithms, due to their high computational cost, should have some subroutines which execute for distinctly high amounts of time. These subroutines can be represented by their stack height, and will be encoded in the *amplitudes* of our time series. How long and how many times this subroutine executes should be encoded into *frequencies* of our time series. Concurrent mining activity is encoded in the amplitude since we sum the signals across all thread. Since miners employ PoW computation in parallel to maximize probability of mining a new block, taking into account excessive parallelism will aid in our detection.

B. Memory Signature

A mining script consists of a single script with two parts, the controller and the worker. The controller is the JavaScript

linked on Web servers. The controller contains the worker as an actual JavaScript variable, which is loaded into JavaScript heap space when the controller is downloaded and executed. The worker (usually a WASM script) is the actual piece of code that runs the PoW Algorithm. This process occurs because a WebWorker may only be spawned from a JavaScript fetched from the same same origin as the original Web server due to security reasons. When the worker is loaded into memory it is referenced by a Data URI to download and execute the PoW algorithm. We show the memory footprint created by this process can be separated from the usual JavaScript memory activities of Web page loads.

To measure memory, we monitor the allocation space of the JavaScript heap of the main thread, and all WebWorker threads, exposed by browsers. CoinSpy samples the heap every millisecond across all threads. We then use the total allocated bytes at each sample as our time series for the memory behavior of pages.

C. Network Signature

Once a mining script runs within a browser, it establishes connections to the Web miner's servers over a WebSocket channel in order to get the PoW tasks corresponding to the current block. The script communicates to the Web miner's servers using a protocol to handle RPC tasks. This communication is handled over a bidirectional WebSocket. This communication pattern is unique and is different from the network activity of benign Web page loads. Further, the network and compute activities of cryptojacking are related—the network communication occurs when the compute is idle. The model we build can capture these correlations.

To this end, CoinSpy tracks all network flows and analyzes the byte size of the network payloads made available through browsers. For this feature, we measure the overall network activity of a Web page (including that of the WebWorker threads) by summing the bytes from all requests that were in-flight during at a given millisecond.

D. Why are these Features Future-Proof?

The intuition as to why these features should scale in future is that, even if some implementation details of Web mining scripts change, they still must use some underlying similar Web APIs whose behaviors are less likely to change [33]. Secondly, it is known that the PoW algorithms used by Web miners *must* be memory-bound to be profitable to mine on commodity devices [10], [14]. This limitation means we do not expect their behavioral footprints to drastically change in the future.

E. CoinSpy CNN model

We have shown cryptojacking consists of activities that greatly affect a combination of computation, memory, and network. A key challenge that remains is separating out these resource behaviors on normal Web pages from those that contain cryptojacking. CoinSpy learns a deep learning CNN model that combines all three signals to perform this task. First we describe why we require CNN for classification and then describe its architecture.

Problems with shallow learning As a first step towards building a classifier, we applied shallow learning techniques to understand the signal obtained from the time series. Table I shows classification performance of a shallow Logistic Ridge Regression model. In this experiment, we collect the compute, memory, and network signatures for 10 seconds from 2,000 Websites half of which are ground truth benign and half of which contain cryptomining activity (details about the Websites and ground truth are in §IV). We measure our data sources on millisecond granularity thus, we have 10,000 samples for each data source that make up our time series.

Learning Method	CPU	Network	Heap	All
Logistic Regression	0.800	0.610	0.928	0.853
Deep Learning	0.875	0.714	0.900	**0.973**

Table I: A comparison of balanced classification rate (BCR) for 2,000 labeled Web pages. We can see that while a combination of features for the shallow learning does not increase the accuracy, our CNN model is able to learn more complex features form the raw data and can take advantage of this higher granularity to use the information from all features.

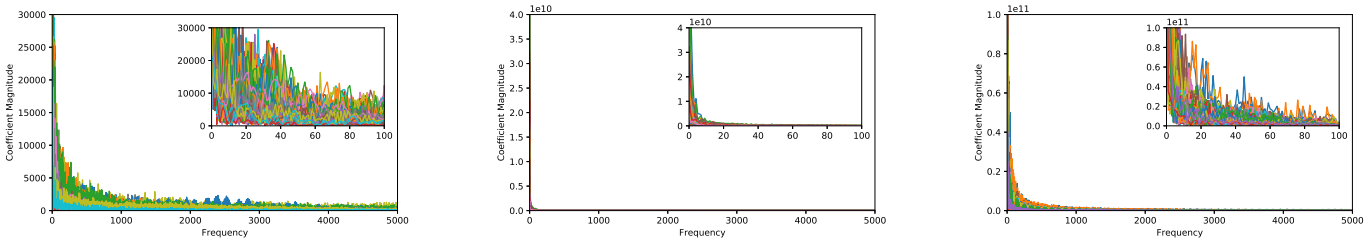
To reduce the dimensionality, we use a Fast Fourier Transform [41] and take the top-K Fourier Coefficients of highest magnitude. As creating one feature vector from the multiple time series is not straight forward, our approach was to create a vector containing the top-k frequency coefficients of each feature as well as frequency coefficients corresponding to the other features at those top-k, thus preventing as much information loss as possible. We used a 10-fold cross-validation to choose the hyper-parameters and normalization constant with the highest accuracy.

We find two inherent problems in shallow models. First, Table I shows that the shallow models have individual classification accuracy of 80.0%, 61.0%, 92.8% when trained individually on the compute, memory, and network timeseries respectively. But, combining them does not improve the signal. This is well known problem with shallow models when combining multivariate time series data [26]. Second is the curse of the dimensionality that shallow learning techniques are known to suffer from. With our sampling rate we will essentially have 10,000 features, which which our number of samples does not far exceed. Using the hand crafted features via the FFT to reduce the dimensionality helps by encoding information from these samples into a smaller amount of features, with the trade off of losing some information.

Why CNNs? In CoinSpy, we use Convolutional Neural Networks (CNNs) to combine the multi-variate time series and build a cryptojacking classifier. CNNs utilize convolutions to learn features in an unsupervised manner and are more resistant to losing signal when modeling. Convolutions are known to be especially effective when the features can be found in multiple locations from within the data; we expect this to be the case given the repeated execution of a PoW algorithm during mining.

As cryptojacking can initiate at varying times during the load process on a host Web page. CNNs accomplish analyzing data with this type of signal very well by using pooling to extract the features obtained from the convolutions across various parts of the data. CNNs also naturally incorporate multidimensional data, such as color channels of the same image or timeseries aligned over the same period, without having to specifically encode this combination into the model's features. These points give us high confidence in the relevance of our choice of model.

CoinSpy's CNN architecture: The CoinSpy choice of deep learning model is also aided by the shallow learning analysis. Figure 5 shows the real Fourier Coefficients for



(a) CPU (b) Network (c) Memory

Figure 5: The plotted magnitudes of the real Fourier coefficients for points our model was highly confident in classifying. From the inner plots, we most of the signal in the time series comes from the lowest 1-2% of frequencies.

the examples that our shallow model was correct and most confident in classifying. From this we can observe that only the lowest 1-2% of frequencies produce a signal across all features. Though not shown, this signal was present in much higher magnitudes for sites that were mining over those that were not, attributing to the results in Table I. It is likely these frequencies are encoding the signal caused by the PoW algorithms.

As the lower frequencies are the most important, the important signal is spread out throughout the timeseries points. In their deeper layers, CNN's accomplish learning from long persistent signals by convolving over highly pooled data. Thus, intuitively it should be these deeper layers of the CNN that can extract information we know to be encoded in these lowest frequencies.

Figure 6 shows the CoinSpy CNN architecture. We treat our align our three CPU, Memory, and Network timeseries to input them as channels and perform six layers of one dimensional convolutions and max pooling before running the resultant signal through a fully connected layer. The fully connected layer is used to combine the signals gathered into the deepest layers in preparations for classification. A softmax layer allows for classification of all three series into a binary label, representing the presence of cryptomining. A standard cross-entropy loss was used. Table I shows that the combining the three time-series using deep learning does indeed improve the classification performance.

F. *Incremental learning*

CoinSpy's CNN architecture naturally supports incremental learning. Deep neural nets learn by training on batches of samples, as opposed to operating on a full data set at once, and iteratively train the model weights by repeatedly performing weight updates over the batches. To incrementally update the model in the presence of a miner that may produce slightly different behaviors, we only need to obtain samples of this miner and continue training from where we left off. As our solution runs entirely in-browser, we expect users of the CoinSpy extension can help provide additional labels to help update the model weights after deployment.

IV. CoinSpy Implementation

We discuss the implementation details in terms of (a) the data collection and labeling approach for training, (b) training for generalizability, and (c) the implementation of an end-to-end CoinSpy browser extension.

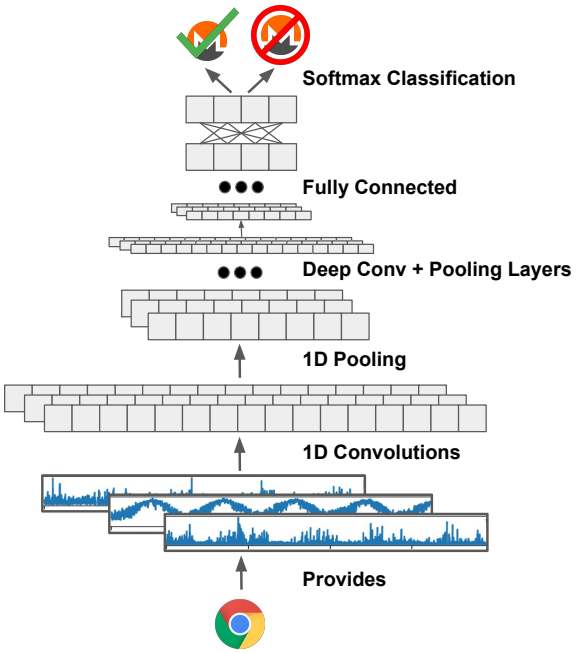


Figure 6: The CoinSpy CNN architecture. Data from taken directly from the browser is fed through a series of one dimensional convolution and pooling layers. There information is combined using a fully connected layer, and a softmax layer to classify a page as mining.

A. *CoinSpy model training and implementation*

The first step towards training the model is to collect training data. We describe our training set below.

At the time of writing, there does not exist a standard set of cryptomining pages on which to train a deep learning model such as CoinSpy. One could go about this by using existing black lists, e.g. from NoCoin [20]. However the size of this dataset is small, only ∼150 domains many of which are simply WebSocket proxy servers for PoW communications that can also be used by benign pages [38]. While the recent work of Outguard [21] does provide a list of domains they found to be participating in cryptojacking, we found that many of these are no longer participating in mining, or simply no longer exist. Thus, we also could not obtain our features from the mining pages listed by Outguard to train CoinSpy.

Instead, we use our injection method to create cryptomining versions of the site. The script we inject is coinhive.min.js

Parameter	Selection Range
Throttle Rate	$\{0, .1, .2, .3, .4, .5, .6, .7, .8, .9\}$
Number of Threads	$\{1, 2, 3, 4, 5, 6, 7, 8\}$
Injection Delay (seconds)	$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Table II: A set of parameters that we vary when injecting a cryptomining script to obtain our positive labels. We randomly pull a value from the set of values listed, running the script according to the parameters chosen.

taken from coinhive.com, which was, at the time of our training, the dominant player in the browser based cryptomining ecosystem [37]. We show in our evaluation of CoinSpy (§V) that our detection extends to other cryptomining families, without explicitly training on them.

We use the Chrome Remote Debugging Protocol [8] to obtain our data sources (Section III). Our data set consists of 5,000 sites from Alexa Top 5k [2]. To these Websites we inject cryptomining activity to create our ground truth dataset (described below). To inject the mining script, we use the functionality of the Debugging Protocol to have the script execute asynchronously as soon as the DOM (Document Object Model) of the page is created. Cryptomining scripts, and hence our simulated attack, all come from client side manipulations, so there is no effect to any live Web servers.

As has been done in the past, we implicitly trust the top 5,000 [11] Web pages to be free of cryptojacking scripts or any malware, and assign them negative labels; the sites with manually injected cryptomining scripts have positive labels. While it is possible that a few of these websites may be running cryptomining scripts and as such are not be true negatives, we believe this number is too small (868 among top 100K in Alexa list [16]) to add any significant noise to our training set. Of the 10,000, we randomly choose 8,000 Websites to train, and test on the remaining sites. We performed all model training and testing via Tensorflow [1] on an Nvidia Tesla K80 GPU.

For each of the 10,000 Websites, we use Google Chrome Version 68.0 to navigate to the base domain of the page, i.e. the landing page. While navigating, we collect samples of the CPU, memory, and network activity sources at the granularity of 1 millisecond, giving us 10,000 time samples of each over the time the page is live in Chrome. Sampling for a longer time period can potentially improve the model, but will result in an increase in model size. Additionally, browser extensions have a limit on the model size they can load, and thus we limit the sampling time. A shorter sampling period also implies a smaller window for real-time detection. We start to profile at the start of the page load to avoid missing out on any cryptojacking signal.

B. Training for generalizability in-the-wild

In order to obtain a representative data set that captures 'in-the-wild' cryptojacking behavior, we vary the parameters of the PoW algorithm and train across these parameters. Table II shows the parameters that we vary when creating our data set. We choose these parameters based on how cryptomining scripts vary in the wild. Many cryptomining scripts come the ability to be *throttled*. A throttling of x% means the cryptomining script will attempt to reduce its number of CPU cycles by x%. This is so that they can attempt to reduce their detection or not burden their client's resources. Cryptominers also vary the number of threads on which they run the PoW computation. Finally, miners often inject a delay after which they start the mining script, to keep in tact the user experience for the initial vital seconds of the page load. To capture these properties, we choose to vary the scripts throttle rate and number of mining threads and add a random delay for when we actually start mining on a page. We then learn the cryptojacking classifier by training over these varying PoW behaviors.

C. End-to-end CoinSpy Browser Extension

We deploy the trained model of CoinSpy as a browser extension, a pop-up interface from which shown from in Figure 4. The extension collects signals about the CPU, network, and memory using browser extension APIs. While we implement CoinSpy directly on top of Chrome, most modern browsers provide means to collect the same data [7]. The extension attaches to the current tab and monitors our datasources in increments of 10 seconds.

The CoinSpy model is implemented in the extension through Tensorflow.js. Tensorflow.js provides optimizations to reduce the model size down from 950 MB to ∼300 MB, and to break it up into 4MB chunks able to be cahced by the browser so that the overhead from the model only applies during the first run of the extension. We expect the extension to help incrementally improve CoinSpy by having users provide feedback on the extensions' labeling process while browsing.

V. Evaluation

In our evaluation of CoinSpy, we:

- Show CoinSpy has high detection accuracy of over 95% across different datasets consisting of live and synthetic mining activity.
- Show CoinSpy model can detect mining by other cryptomining families even when not trained on them, with an accuracy of over 98%. Further we show the CoinSpy model, despite being behavioral, is not device-specific.
- Compare CoinSpy to the relevant state-of-the-art detectors CMTracker [16] and Outguard [21]. We show CoinSpy performs better on the current cryptomining landscape, both through synthetic and live data, than CMTracker and comparably to Outguard. However we provide evidence as to why the design of CoinSpy should scale better to future detection than both these alternatives.
- Provide evidence to show that the CoinSpy browser extension does not significantly affect the user experience on client browsers.

A. Evaluation Methodology

Datasets: For the evaluation of CoinSpy, we collect behavioral data during page loads from four different Web page sets (described below). To run all experiments, we instrument the Chrome browser using remote debugging [7] to load each page in our dataset for 10 seconds, and under a cold cache. We describe the page data sets below:

Test dataset: This dataset consists of 10K total Websites: 5k benign Websites and 5k cryptomining websites where we inject cryptojacking script from CoinHive [24] these pages. We use this large data set for training our model (§IV). We use an 80-20 train-test split.

Curated in-the-wild dataset: To evaluate CoinSpy with real cryptomining sites in-the-wild, we create a 100-page data set. We query PublicWWW [36] database for domains that contain static cryptomining signatures [37], [23]. Our query returned 857 Web pages with cryptomining signatures. However, just

because a mining signature is present, does not mean the site will actually mine. Examples of this include if the delivery of the mining script does not occur on every load of the page [16], or if the mining is subject to a user action. We thus performed a manual sampling of 100 pages, which showed 80 of which were actually executing cryptomining signal, and 20 that were not despite the signatures present.

Benchmark dataset: We collect 50 sites randomly from Alexa Top 5K. We manipulate these in various ways, such as injecting cryptojacking scripts from different families, which are discussed further where relevant.

Alexa 100K: To evaluate on an exclusively in-the-wild dataset, we randomly sample a set of 100k sites from the Alexa Top 1 Million pages and run CoinSpy over the resulting data.

Devices: We experiment across two devices. The first is a cloud device with a 12 Core Intel Xenon Processor, 16 GB of memory, and running Ubuntu 18.04. The second is an Ubuntu 18.04 Desktop PC with an 8 Core i5 processor, 8 GB of RAM. We perform cross device testing to show our learned behavior is not machine dependent. If not otherwise specified, we present the results with data collected from the Desktop device.

Evaluation Metrics: We evaluate CoinSpy using four metrics. The first is the overall accuracy. The second is Sensitivity (True Positive Rate), i.e., how many true mining sites we correctly identify as cryptojacking. The third is Specificity (True Negative Rate), i.e, how many benign sites we do not flag as cryptojacking, and Balanced Classification Rate, the average of these two given some data sets contained an unequal number of positives/negatives.

Compared Methods: We compare CoinSpy with the state-of-the-art in-browser alternatives, *CMTracker* [16] and *Outguard* [21]. CMTracker detects cryptomining using a simple heuristic: looking for a JavaScript function that executes on the CPU for more than a threshold time.

Outguard combines static and behavioral features to train a linear SVM model. Specifically, their static features include a binary value dictating presence of known cryptomining function identifiers on the page and a binary value indicating that a named JavaScript function repeated multiple times. Their behavioral features include the number of WebWorkers being executed, the binary presence of WebAssembly, and signal that WebWorkers were engaging in heavy interprocess communication via the PostMessage API [3]. This high amount of ipc comes from the workers beign given PoW tasks and returning their PoW guesses (§II). Other alternative cryptomining detectors (discussed §VI) either are not confined to the browser sandbox [23] or use exclusively static signatures to detect cryptomining [37].

B. Evaluating CoinSpy accuracy

Table III shows that accuracy of running CoinSpy over the test data set. The first column shows the accuracy of running CoinSpy on the cloud device, the device on which the model was trained on. CoinSpy has over 97% accuracy on the test data set. As the CoinSpy extension will be operating on behavioral footprints different from those of the original training machine, we apply a simple normalization technique to adapt our features to new machines.We learn a Z-score for the means and variances for each of our 10,000 time series points across all samples from our cloud data set. When testing on data from a new machine, we first normalize the time series points according to these Z-scores. Table III shows a $<1\%$ drop in accuracy when testing CoinSpy on features from a new machine.

Metric	Cloud device	Desktop device
Accuracy	.973	.964
Sensitivity	.970	.982
Specificity	.977	.947
BCR	.973	.964

Table III: Results on CoinSpy on the 2K test dataset. We first run CoinSpy on the same device that the model is trained on (the Cloud device). We then test CoinSpy on an alternate Desktop device by applying CoinSpy without retraining. In both cases, CoinSpy accuracy is high, over 96%.

C. Evaluating the CoinSpy browser extension

Our evaluation of the CoinSpy extension considers whether the overhead of collecting the model features and running testing the model live will affect the user's Web quality of experience. To do so, for each page load we measured the extension's performance impact through three common page load time metrics, onload [30], first contentful paint (fcp) [32], and time-to-interactivity [31]. These measure network load time, rendering time, and when the user can interact with the page respectively [19], [29].

We consider the model to have already been loaded and parsed from the cache/network when running this experiment, and do not include this initial overhead. We ran each page 10 times with a cold browser cache, both with and without the model executing, and represented our final numbers for each page as the mean of the metrics across the runs in milliseconds.

We found that, in the median case, the onload, fcp, and tti, only inflated by 137, 15, and 124 milliseconds respectively, which all fall into natural variances in page load times [43]. Further, a *paired t-test* for each metric showed CoinSpy made no significant difference in the distributions of any metric across pages (all $p > .05$). This shows that CoinSpy does not provide significant overhead for the user's Web quality of experience.

D. Extending CoinSpy to other mining families

Our goal is to evaluate CoinSpy over other players in the cryptomining landscape. Thus, we evaluate CoinSpy against cryptomining players identified by existing works [23], [37], [33]. These miners primarily use CryptoNight [10] and ETHash PoWs [44], but span many cryptocurrency blockchains (Monero, JSECoin, UPlexa). JSEMiner uses the ETHash PoW [44] (see Figure 1). CoinImp, CryptoLoot, and WebMine use the CryptoNight PoW [10] to mine Monero, but CoinImp and CryptoLoot also offer scripts with modified versions of CryptoNight to mine the WebChain and uPlexa blockchains respectively.

Table IV shows the accuracy of CoinSpy in detecting the five CryptoNight families of miners. To perform the evaluation, we inject the scripts from each of these miners into the pages from our *benchmark* data set for evaluation, hence we only evaluate with accuracy. Although researchers have identified 13 popular miners [23], [37], [33] that use CryptoNight, only the five in Table IV are active at the time of performing our experiments. In fact, CoinHive, the largest Web mining player of 2018 [21], [37] has shut down their service as of March 2019, showing the volatility of the environment.

Importantly, CoinSpy is able to detect the presence of these other miners, without explicitly training for them, with an

Miner	CoinSpy	CMTracker	Outguard
CoinImp (Monero)	1.00	.82	1.00
CoinImp (WebChain)	.98	.84	1.00
Crypto-loot (Monero)	1.00	.82	1.00
Crypto-loot (uPlexa)	.98	.84	1.00
WebMine (Monero)	1.00	.82	1.00

Table IV: Evaluating CoinSpy and other relevant approaches on detecting five other cryptomining families that employ the CryptoNight PoW [10]

accuracy of over 98%, showing the power of our features and model. This result also shows that, despite mining different blockchains/currencies, these scripts do not produce highly variable behaviors outside those accounted for in our training. However, while Outguard also performs comparably or better on these new families, we analyze its weaknesses in being used as future-proof detector below.

For the ETHash PoW [44] based miner, JSEMiner [17], our initial tests were not able to perform accurate detection. We found this to be the case for Outguard and CMTracker as well. On further inspection, we find that this is because JSEMiner prioritizes the user experience over resource usages and severely throttles CPU usage, is implemented using native browser JavaScript libraries without WebAssembly, and executes only in the main thread of the page. We consider this result a positive; CoinSpy does not overreach in detection, only detecting true abusive cryptominers.

E. Detection over the in-the-wild dataset

We move to analyze results from a small sample of in-the-wild miners using our *curated in-the-wild* dataset. Table V shows the accuracy of CoinSpy and closely related techniques. Recall from above this dataset contained 80 ground truth mining pages and 20 pages which contained mining signatures but did not actually have the mining execute, which we labeled as ground truth negatives via manual inspection. We hence analyze in terms of both Sensitivity and Specificity.

CoinSpy was able to correctly classify 75 out of the 80 (93.8%) sites that were participating in mining. CoinSpy correctly classified as non-cryptojacking all of the 20 Websites that contained cryptomining code but did not perform cryptojacking. In contrast, CMTracker [16] has a low Sensitivity of 66%. We explore this aspect of CMTracker further in our exploration of what makes CoinSpy future-proof.

We note that Outguard actually performed very slightly better in detecting the miners than CoinSpyin this experiment, detecting 79 of 80 ground truth miners (98.8%). This suggests that Outguard actually outperforms CoinSpy for in-the-wild mining detection in the current landscape. However we explore below why Outguard may be less future-proof as a detector.

Finally, we analyzed the five false negatives where CoinSpy marked the sites as not cryptomining. Four out of the five sites throttled their CPU utilization to over 80%. Such a high throttling rate means that the CPU is not utilized much, weakening the PoW signal and making the classification hard. However, this also restricts mining payouts. One site did not throttle but only mined at the last second of our profile; a longer profiling period for this page would likely allow us to detect it, such as if the CoinSpy extension were to collect the next 10 seconds of data while in use during live browsing.

F. A case for future-proof detection

While we have shown that CoinSpy is a performant detector under the current mining landscape, it admittedly performs

Metric	CoinSpy	CMTracker	Outguard
Accuracy	.95	.73	.98
Sensitivity	.938	.666	.988
Specificity	1.0	1.0	.95
BCR	.969	.799	.969

Table V: Accuracy of CoinSpy and comparisons with CMTracker and Outguard on the in-the-wild dataset.

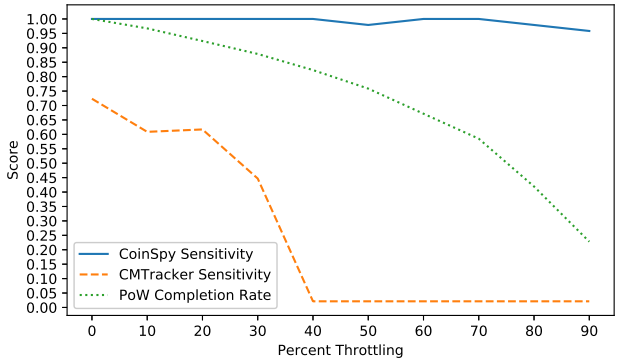


Figure 7: A comparison of the false negative rate of CoinSpy and CMTracker for different CPU throttling. CMTracker does not detect miners if the CPU is throttled over 40%. In contrast, CoinSpy is trained to be robust to throttling, only missing 5% of miners in our set for throttle rates up to 90%. The third line is the PoW completion rate relative to that of 0% throttling.

comparably to the Outguard [21] detector. To differentiate, we provide several arguments for why the CoinSpy features and design provide a more solid basis over for detection going forward. We do so by analyzing the robustness of its features, and its ability to incrementally update to small behavioral changes in PoW behaviors.

Throttling CPU: Cryptominers can throttle the rate at which they use the CPU to go undetected. Because CoinSpy and CMTracker both operate on the behavioral effects of mining on the CPU, we study how these detectors work under different throttling rates. To this end, on the benchmark dataset, we inject cryptomining activity but vary the CPU throttling from $[0, .9]$ in increments of .1. Correspondingly, we compute the PoW completion rate relative to the maximum completion rate at 0% throttling, to understand the profit trade-offs in throttling.

Figure 7 shows the accuracy and hash rate of CoinSpy and CMTracker across all throttling rates. CMTracker stops detecting miners that have their rates throttled by 40% while the PoW completion rates only decreased by $\sim$15% from that of no throttling. We saw the effect this had on the performance of CMTracker for our *curated in-the-wild* dataset, showing that throttling is applied in real settings. Outguard, in fact, cites throttling as a main reason to avoid using a simple CPU utilization as a feature in its model [21]. Since it does not use explicit CPU behavior, it is not affected by CPU throttling.

In comparison, CoinSpy keeps the same relative performance throughout throttle rates, failing to detect only 5% of the miners for throttle rates up to 90%. This result comes form the fact that CoinSpy does not have a single static parameter, trains across throttling rates, and includes other aspects of the cryptomining behavior (memory, network activity).

Version Specific Implementation: Outguard obtains its features from properties parsed from low-level browser traces.

While these can be accessible from the browser's runtime, this does not mean that Outguard is guaranteed to function following each browser update. Outguard built for the original version of Chrome as used by the authors [21] produced *all negatives*, for all of our previous evaluations, when tested on Chrome 73 and 76. We thus modified Outguard's feature extractor to allow it to function on new data collected for each of these new versions of Chrome. We verified, using ground truth injected scripts on our benchmark dataset, that these rebuilt Outguard versions performed to expectation [21]. The CoinSpy behavioral features, in comparison, were invariant with respect to the browser versions tested.

Modifying Mining Behavior: Here we aim to show how small changes to the implementation of miners can affect the state-of-the-art while leaving CoinSpy unaffected. For this experiment we modified JSEMiner. Recall that JSEMiner uses a variety of techniques to make mining less abusive for clients, at the cost of lower profit rates. Under the motivation that certain, more malicious, Web servers will want to maximize cryptocurrency profits, we created a new JSEMiner that actually reduces PoW throttling and performs PoW execution in parallel from within WebWorkers.

CoinSpy detects this aggressive JSEMiner with 97.8% accuracy. Outguard, on the other hand, was only able to detect this JSEMiner with an accuracy of 8.0%. This is because of Outguard's reliance on the inter-process communication activities used by conventional miners to relay PoW tasks to and from the main and WebWorker threads. However, as JSEMiner is designed for running on a single thread, each WebWorker in our modified version actually acts as both a controller and a worker; no communication is done between them via PostMessage. CoinSpy makes no such implementation assumptions, directly picking up the behavioral signals generated by the more aggressive ETHash.

Potential to be conservative: We analyze how CoinSpy and alternatives compare in how conservative they are in labeling benign sites as infected. As we have shown in our analysis of our *curated in-the-wild* dataset, Table V, CoinSpy and CMTracker did not show false positives, whereas Outguard showed 5% false positives. While quantitatively similar, there are some qualitative differences that make CoinSpy stand out.

In the case of CMTracker, its thresholding approach represents a direct trade-off of false positives for false negatives; choosing too low a threshold for CPU will start flagging benign sites as malicious. The recommended threshold of CMTracker, 30%, works well to avoid false positives. However, as we have showed, this threshold increases false negatives in the presence of throttling.

For Outguard, we manually inspected the 5% false positives, and found that these were labeled because the amount of inter-process communication between the main thread and WebWorker threads was very high. Similar to how the stress on this implementation detail hurts the robustness Outguard to detect a modified JSECoin, it also causes it to flag benign sites. Such inter-process communication is actually a common use case for Web pages [46], especially as use cases for WebWorkers inevitably expand in the future. CoinSpy however, operates on the exact behavioral CPU signals that PoWs make when executing within these WebWorkers, not just on this specific implementation detail of current miners. While it is unclear that these biases in Outguard and CMTracker can be easily remedied, we will show that CoinSpy can improve its false positive rate (as observed in our *Test Dataset*) incrementally by retraining with up to 10% improvement (shown in Table VI).

Method	BCR	BCR (Without incremental training)
Shifting	.880	.835
Scaling	.987	.970
Warping	.968	.915
All	.943	.842

Table VI: CoinSpy performance in the presence of perturbations, both with and without incremental training. The table shows that CoinSpy's incremental training is critical to its performance.

Future changes and Incremental learning: We have argued the features selected by CoinSpy are more future proof than those used for detection by CMTracker and Outguard. While we have argued our behavioral features are not likely to change in the future (§III), we still evaluate whether CoinSpy can be thwarted by small changes made to implementations of cryptomining algorithms. As finding ground truth in-the-wild to quantify these new changes is hard, we simulate such changes in miners by adding a set of common timeseries perturbations [26] to our original data set.

- Shifting, where each time sample is replaced by a sample some constant seconds in the future. We shift our 10 seconds of data from the set of $\{1, 2, 3\}$ seconds.
- Scaling, where each sample is amplified by a constant factor. We scale by factors from the set of $\{2, 3, 4\}$.
- Warping, where the speed of samples are changed by replacing each sample with a sample in the future at an increasing, e.g. sample i is replaced by sample $2i$ for all i. We warp by factors from the set of $\{2, 3, 4\}$.
- A random selection from *all* of the above for each sample.

Table VI shows the accuracy of CoinSpy when the cryptomining algorithm is perturbed by shifting, scaling, and warping. CoinSpy is able to perform cryptojacking with a 94.3% BCR across perturbations (the corresponding accuracy was also 94.3%). Further, the table shows the importance of incremental training, a design point of CoinSpy. Without incremental training though, the performance is much lower, at 84.2% BCR across the perturbations, highlighting the benefits of this design choice for CoinSpy.

G. Evaluating CoinSpy on Alexa Top 1M

We finally provide a small measurement experiment to analyze the state of mining on the Alexa Top Sites [2]. The Outguard project reported from experiments in mid-2018 that 0.06% of the Top 1M were engaging in cryptojacking, and earlier works using static analysis quoted similar figures [37]. To check these results, we ran CoinSpy over a random sample of 100k sites from the Alexa Top 1M in early 2019 to see the state of these claims on the landscape.

It is hard obtain an explicit ground truth for verification, due in part to the known volatility of sites to actually include cryptojacking on their pages, even within a small time frame [16]. We thus explore a small sample of sites, of size 100, taken from the pool of sites in which CoinSpy labeled positive over the 100k, and examine them manually. Our manual inspection revealed 3 of these pages to be ground truth miners. Given the sample sizes involved, we consider the figure of 0.06% miners on the Top 1M to be plausible.

VI. Related Work

Our work is at the intersection of cryptojacking, behavioral analysis, and Web security. We discuss related work in each of these areas.

A. Cryptomining on the Web

Although Web cryptomining is a relatively recent phenomenon there has been numerous research efforts to perform its exploration and detection. Existing works have looked at which Websites likely have cryptomining scripts, how much profit one can make with cryptojacking, and the percentage of top ranked pages that run cryptomining [37], [14], including recent studies which take these analyses outside the scope of the browser [35].

In terms of detection specifically, several approaches use existing public blacklists [14] or use static signatures [37], [20] to detect Web cryptomining. Antivirus software such as Symmantec also detect cryptomining using a static signature. The problem is that static code signatures and pattern matching can be evaded, and such evasion techniques are already being deployed by cryptominers [16], [38].

Recent works have begun to develop more general detection methods, studying the unique behavior of cryptojacking scripts [23], [16]. One such approach is utilized by Minesweeper [23] monitors CPU L1 caching reads and writes as a signal for mining detection, given the known frequent access rates to these caches by memory hard PoW algorithms. However this low level cache information cannot be accessed from within the scope of the browser and thus cannot be utilized by CoinSpy.

Another closely related work [38], supplements their static analyses, based on code length and complexity of cryptomining scripts, by providing a first look at how behavioral features change for sites under cryptomining. While they do not provide an explicit behavioral detector and accuracy metrics, they observe the changes in CPU Usage, Web Socket packet sizes, and browser power consumption. We have argued that raw CPU usage is not robust (See §V). Further, power consumption cannot be accurately monitored accurately using resources only available within Web browsers [42]. Finally, while we have also shown the effectiveness of WebSocket activity in detecting miners, CoinSpy shows how all three behavioral resources of CPU, Network, and Memory can actually be combined to perform more robust detection.

Finally, CMTracker [16] and Outguard [21], are closest to our work and are in-browser detectors that also uses behavioral analysis. We showed in our evaluation that CoinSpy outperforms CMTracker both in terms of accuracy and reducing false positives, and is more future-proof to detect miners than Outguard.

There has been recent work on solving PoW algorithms to accomplish social good, rather than waste resources. WebCoin [22] designs a PoW for a new blockchain that can be used to create a global distributed search indexing tool for the Web while securing the blockchain itself. As this approach is still its early theoretical stages, we do not evaluate CoinSpy on this PoW algorithm.

B. Web security

Similar to cryptomining detection, several (Web) security defenses focus on detecting malware with static signatures such as presence of known IP/domain blacklists, or number of DOM read/write operations [6], [5]. While these works have been shown to be computationally efficient they are also vulnerable to evasive techniques such as code obfuscation [27], [18].

In response to such evasions, dynamic, behavioral based defenses have come into favor. Examples range from using access patterns to detect code injection attacks [34], using TCP traffic patterns to detect botnets [41], and using performance counting behavior to detect Linux rootkits in Android phones [12]. However, these defenses are specific to the application area and cannot be applied to detecting cryptomining. As such, the intuition of a given application area is key to designing such robust behavioral defenses.

C. Behavioral analysis using learning

Machine learning is a popular method to learn behavioral patterns given historical data. Many approaches use shallow learning models, e.g. Logistic Regression or Naive Bayes using feature engineering processes [6]. Recently, there has been a push towards deep learning based approaches that does not require hand-crafted feature engineering. Some of these techniques, such as classifying Android malware [45], make use of low level data sources that are not accessible from browsers. Other security tools deploy directly on the browser [11], [28] and also highlight the benefits of an in-browser defense. CoinSpy also heavily leverages deep learning as a means to perform behavioral detection of cryptojacking, in-browser.

VII. Conclusion And Future Work

Browser-based cryptomining is a recent ethical and security concern on the Web, with cryptomining scripts being delivered by ranked domains and even through advertisement networks [23]. Existing defenses have several drawbacks including what actions they can take when mining activity is detected, their ability to be adopted by users, and how easily they can be thwarted. To this end, we design and build CoinSpy an entirely in-browser based framework for detecting cryptomining activity within Web pages. CoinSpy is built using a deep-learning pipeline that combines a variety of signals generated when a cryptomining script executes on a Web page. We perform in-depth analysis of both injected sites as well as websites in the wild and show that our model is accurate should provide better detection in future over existing alternatives.

While we have shown CoinSpy can detect cryptomining with high accuracy, there are several future directions in which this work can continue. In the near term, we would like to evaluate incremental learning on PoW algorithms that are not currently mined en masse on the Web [22]. We would also like to evaluate different neural network architecture based on capabilities of client device types. For example, based on the client capabilities we may decide to use a 3-layer CNN instead of 6 to evaluate the performance vs. accuracy trade-off. To supplement performance, we can also explore using automated frameworks such as Tensorflow-lite, to further optimize the model and its size.

We also plan to evaluate user statistics of the CoinSpy browser extension, such as how many users are actually able to provide us with new labeled pages for in-the-wild miners that we may currently miss. In the longer term, we would like to answer the open question as to what actions can, and should, be taken from the browser once cryptomining is detected; for now we focus on detection.

REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[2] Alexa, "Alexa top sites," https://aws.amazon.com/alexa-top-sites/.

[3] M. Avellar, "Window.postmessage," https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage, September 2019.

[4] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario, "Automated classification and analysis of internet malware," in *Proceedings of the 10th International Conference on Recent Advances in Intrusion Detection*, ser. RAID'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 178–197. [Online]. Available: http://dl.acm.org/citation.cfm?id=1776434.1776449

[5] K. Borgolte, C. Kruegel, and G. Vigna, "Delta: Automatic identification of unknown web-based infection campaigns," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. New York, NY, USA: ACM, 2013, pp. 109–120. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516725

[6] D. Canali, M. Cova, G. Vigna, and C. Kruegel, "Prophiler: A fast filter for the large-scale detection of malicious web pages," in *Proceedings of the 20th International Conference on World Wide Web*, ser. WWW '11. New York, NY, USA: ACM, 2011, pp. 197–206. [Online]. Available: http://doi.acm.org/10.1145/1963405.1963436

[7] A. Cardaci, "Chrome remote interface," https://github.com/cyrus-and/chrome-remote-interface.

[8] Chrome Debug Team, "Chrome remote debugging," http://bit.ly/2rnmsZx.

[9] Crypto-Loot, "Crypto-loot becomes the number one web miner," https://cryptolootminer.com/news, February 2019.

[10] CryptoNote, "The cryptonight proof of work algorithm," https://cryptonote.org/inside.php.

[11] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert, "Zozzle: Fast and precise in-browser javascript malware detection," in *Proceedings of the 20th USENIX Conference on Security*, ser. SEC'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 3–3. [Online]. Available: http://dl.acm.org/citation.cfm?id=2028067.2028070

[12] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo, "On the feasibility of online malware detection with performance counters," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: ACM, 2013, pp. 559–570. [Online]. Available: http://doi.acm.org/10.1145/2485922.2485970

[13] E. Elliot, "What is web assembly?" June 2015.

[14] S. Eskandari, A. Leoutsarakos, T. Mursch, and J. Clark, "A first look at browser-based cryptojacking," *CoRR*, vol. abs/1803.02887, 2018. [Online]. Available: http://arxiv.org/abs/1803.02887

[15] C. Extensions, "Stay secure," https://developer.chrome.com/extensions/security.

[16] G. Hong, Z. Yang, S. Yang, L. Zhang, Y. Nan, Z. Zhang, M. Yang, Y. Zhang, Z. Qian, and H. 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*, ser. CCS '18. New York, NY, USA: ACM, 2018, pp. 1701–1713. [Online]. Available: http://doi.acm.org/10.1145/3243734.3243840

[17] JSEcoin, "Digital currency - designed for the web," https://jsecoin.com/en/home/.

[18] A. Kapravelos, Y. Shoshitaishvili, M. Cova, C. Kruegel, and G. Vigna, "Revolver: An automated approach to the detection of evasive web-based malware," in *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*. Washington, D.C.: USENIX, 2013, pp. 637–652. [Online]. Available: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/kapravelos

[19] C. Kelton, J. Ryoo, A. Balasubramanian, and S. R. Das, "Improving user perceived page load times using gaze," ser. NSDI '17. USENIX Association, 2017, pp. 545–559. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/kelton

[20] R. Keramidas, "Nocoin: a tiny browser extension aiming to block coin miners," https://github.com/keraf/NoCoin, September 2017.

[21] A. Kharraz, Z. Ma, P. Murley, C. Lever, J. Mason, A. Miller, N. Borisov, M. Antonakakis, and M. Bailey, "Outguard: Detecting in-browser covert cryptocurrency mining in the wild," in *The World Wide Web Conference*, ser. WWW '19. New York, NY, USA: ACM, 2019, pp. 840–852. [Online]. Available: http://doi.acm.org/10.1145/3308558.3313665

[22] U. Klarman, M. Flores, and A. Kuzmanovic, "Mining the web with webcoin," in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '18. New York, NY, USA: ACM, 2018, pp. 165–177. [Online]. Available: http://doi.acm.org/10.1145/3281411.3281415

[23] R. K. Konoth, E. Vineti, V. Moonsamy, M. Lindorfer, C. Kruegel, H. Bos, and G. 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*, ser. CCS '18. New York, NY, USA: ACM, 2018, pp. 1714–1730. [Online]. Available: http://doi.acm.org/10.1145/3243734.3243858

[24] B. Krebs, "Who and what is coinhive?" https://krebsonsecurity.com/2018/03/who-and-what-is-coinhive/, march 2018.

[25] Monero, "Private digital currency," https://www.getmonero.org/.

[26] F. Morchen, "Time series feature extraction for data mining using dwt and dft," Philipps-University Marburg, Tech. Rep., 2003.

[27] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, Dec 2007, pp. 421–430.

[28] C. Neasbitt, B. Li, R. Perdisci, L. Lu, K. Singh, and K. Li, "Webcapsule: Towards a lightweight forensic engine for web browsers," in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: ACM, 2015, pp. 133–145. [Online]. Available: http://doi.acm.org/10.1145/2810103.2813656

[29] R. Netravali, V. Nathan, J. Mickens, and H. Balakrishnan, "Vesper: Measuring time-to-interactivity for web pages," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, Apr. 2018, pp. 217–231. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/netravali-vesper

[30] M. D. Network, "The onload property of the GlobalEventHandlers," https://developer.mozilla.org/en-US/docs/Web/API/GlobalEventHandlers/onload.

[31] D. Oksnevad, "Time to interact: A new metric for measuring user experience." http://bit.ly/2Fz0mvd.

[32] S. Panicker, "W3c paint timing working draft," http://bit.ly/2f2CGSk.

[33] P. Papadopoulos, P. Ilia, M. Polychronakis, E. P. Markatos, S. Ioannidis, and G. Vasiliadis, "Master of web puppets: Abusing web browsers for persistent and stealthy computation," *CoRR*, 2018.

[34] B. L. Paruj Ratanaworabhan and B. Zorn, "NOZZLE: A defense against heap-spraying code injection attacks," in *Presented as part of the 18th USENIX Security Symposium (USENIX Security 09)*. Montreal, Canada: USENIX, 2009. [Online]. Available: https://www.usenix.org/node/

[35] S. Pastrana and G. Suarez-Tangil, "A first look at the crypto-mining malware ecosystem: A decade of unrestricted wealth," in *Proceedings of the Internet Measurement Conference*, ser. IMC '19. New York, NY, USA: ACM, 2019, pp. 73–86. [Online]. Available: http://doi.acm.org/10.1145/3355369.3355576

[36] PublicWWW, "Search engine for source code," https://publicwww.com/.

[37] J. Rüth, T. Zimmermann, K. Wolsing, and O. Hohlfeld, "Digging into browser-based crypto mining," in *Proceedings of the Internet Measurement Conference 2018*, ser. IMC '18. New York, NY, USA: ACM, 2018, pp. 70–76. [Online]. Available: http://doi.acm.org/10.1145/3278532.3278539

[38] M. Saad, A. Khormali, and A. Mohaisen, "Dine and dash: Static, dynamic, and economic analysis of in-browser cryptojacking," *eCrime 2019*, 2019.

[39] D. Sillars, "The performance impact of cryptocurrency mining on the web," https://bit.ly/2SOPQmv, November 2017.

[40] S. S. R. Team, "Insights into the cyber security thread landscape," https://www.symantec.com/blogs/threat-intelligence/istr-23-cyber-security-threat-landscape, March 2018.

[41] F. Tegeler, X. Fu, G. Vigna, and C. Kruegel, "Botfinder: Finding bots in network traffic without deep packet inspection," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12. New York, NY, USA: ACM, 2012, pp. 349–360. [Online]. Available: http://doi.acm.org/10.1145/2413176.2413217

[42] M. Varvello, K. Katevas, M. Plesa, H. Haddadi, and B. Livshits, "Batterylab, a distributed power monitoring platform for mobile devices," *Proceedings of the 18th ACM Workshop on Hot Topics in Networks - HotNets '19*, 2019. [Online]. Available: http://dx.doi.org/10.1145/3365609.3365852

[43] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "How speedy is SPDY?" in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI '14. USENIX Association, 2014, pp. 387–399. [Online]. Available: http://dl.acm.org/citation.cfm?id=2616448.2616484

[44] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," https://ethereum.github.io/yellowpaper/paper.pdf, December 2018.

[45] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, "Droid-sec: Deep learning in android malware detection," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: ACM, 2014, pp. 371–372. [Online]. Available: http://doi.acm.org/10.1145/2619239.2631434

[46] A. Zlatkov, "How javascript works: the building blocks of web workers + 5 cases when you should use them," https://bit.ly/2Hs3eqP, January 2018.