GPU-based Cryptojacking, attacco e difesa

Anna Lisa Ferrara
Dep. of Biosciences and Territory
University of Molise
annalisa.ferrara@unimol.it

Federico Zappone
Dep. of Biosciences and Territory
University of Molise
f.zappone1@studenti.unimol.it

***Sommario*—In quest'ultimo decennio si è molto discusso di monete digitali e delle così dette criptovalute in seguito all'avvento delle blockchain. Il 3 gennaio 2009 veniva scritto il primo blocco di *Bitcoin*dal valore complessivo inferiore a un dollaro statunitense, in seguito raggiunse il suo massimo storico di quasi $65.000. Divenuto un caso più unico che raro, *Bitcoin* si è posto da apripista a più di 5000 altre criptovalute fino ad essere definito come l'oro del XXI secolo, questo grazie anche al processo di generazione caratteristico delle criptovalute detto *mining*. Grazie alla privacy che queste criptovalute sono in grado di offrire si sono sviluppati diversi tipi di attacchi informatici tra cui il *cryptojacking* in grado di utilizzare potenza computazionale del computer vittima al fine di effettuare *mining* di criptovalute. Questi tipi di attacchi si sono molto diffusi ma, ciononostante, lo sviluppo si è limitato ad attacchi che utilizzassero solo il processore delle vittime. In questo articolo viene mostrato come è concettualmente possibile effettuare un attacco di *criptojacking* in combinazione con il *Cross Site Scripting (XSS)* sfruttando però la potenza computazionale della *Graphics processing unit (GPU)* della vittima.**

***Index Terms*—Cryptojacking GPU XSS Injection**

I. Introduzione

In quest'ultimo decennio si è molto discusso di monete digitali e delle così dette criptovalute in seguito all'avvento delle blockchain. Quest'ultima è una tecnologia basata sul concetto di database distribuito, ovvero un sistema che utilizza un registro condiviso per salvare informazioni, esso è modificabile solo dai nodi della stessa rete ed è rappresentabile come una successione di blocchi contenenti delle informazioni. Contemporaneamente alle blockchain è nata anche la prima criptovaluta, più precisamente, il 3 gennaio 2009 veniva scritto il primo blocco di *Bitcoin* [18] dal valore complessivo inferiore a un dollaro statunitense [12]. In seguito il 6 novembre 2010 il valore di *Bitcoin* era di $0,50, in meno di due anni il prezzo era aumentato di circa 625 volte, e, il 14 aprile 2021, raggiunge il suo massimo storico di circa $64.800 [2] [30] [3]. Divenuto un caso più unico che raro, *Bitcoin* si è posto da apripista a più di 5000 altre criptovalute [4] fino ad essere definito come l'oro del XXI secolo. Questa definizione non è dovuta solo all'incredibile aumento del prezzo negli ultimi anni ma anche ad una delle caratteristiche chiave che sia l'oro che la maggior parte delle criptovalute condivide, il processo di "estrazione" definito appunto come *mining* nel campo delle monete digitali. Il *mining* di criptovalute consiste nel creare monete virtuali attraverso la risoluzione di alcune funzioni crittografiche necessarie per la validazione delle transazioni e dei blocchi che compongono una blockchain. Questa operazione termina con un compenso da parte della blockchain al *miner*, il quale, avendo offerto la sua potenza di calcolo viene premiato con un compenso monetario. Tali calcoli sono in generale eseguiti da sistemi informatici dedicati a questo specifico processo, essi sono divisi in due macro gruppi in base all'hardware che utilizzano: quelli che sfruttano la Central Processing Unit (CPU) e quelli che sfruttano la *Graphics processing unit (GPU)*. I primi prendono il nome di *ASIC*, acronimo di *Application Specific Integrated Circuit*, ovvero circuiti costruiti per la risoluzione di un calcolo ben specifico che risultano però molto inefficienti su altri tipi di algoritmi. I sistemi basati su GPU sono invece spesso molto più prestanti, le schede video riescono infatti a effettuare più calcoli al secondo rispetto alle CPU che risultano quindi meno redditizie nella maggior parte dei casi di *mining*. D'altro canto per le GPU risulta essere molto più difficile la gestione delle temperature e inoltre comportano costi maggiori sia in termini di assemblaggio che di costi energetici per il mantenimento. Questi ultimi aspetti hanno un notevole impatto per quanto riguarda il *mining*: il costo di acquisto delle componenti è nettamente aumentato negli ultimi anni, sia a causa dell'avanzamento delle tecnologiche più efficienti, sia a causa della grande domanda da parte degli interessati al *mining*.

Con l'aumentare dei costi, e allo stesso tempo delle opportunità di guadagno offerte dal mondo in crescita delle blockchain, le criptovalute hanno iniziato a risaltare agli occhi del crimine informatico. Più precisamente la tecnologia blockchain, e di conseguenza una buona parte delle criptovalute, posseggono una caratteristica molto importante per i cyber criminali, ovvero garantiscono una certa forma di anonimato. Le criptovalute infatti sono difficilmente riconducibili ad una persona fisica diversamente da un conto bancario classico. Le criptovalute sono infatti collegare ad un portafoglio digitale, detto appunto *wallet*. L'accesso ai *wallets* non è però regolamentato, e quindi non collegato fisicamente a qualcuno. Questa caratteristica ha portato a un enorme utilizzo delle criptovalute per lo svolgimento di azioni illecite, nel 2017 infatti si è riscontrato un netto aumento dei così detti *ransomware* [25], ovvero virus che una volta bloccato il sistema sul quale sono eseguiti chiedono un riscatto per il suo sblocco. Tale pagamento di tale riscatto è richiesto sotto forma di criptovalute per mantenere l'anonimato. Successivamente si è arrivati a una forma di attacco più "intelligente" e meno invasiva, ovvero i *cryptominers*. Definito appunto come "l'anno dei cryptomi-

ners", il 2018 vede la stessa impennata di casi di *ransomware* dell'anno precedente sotto una nuova forma di attacco che consiste nell'utilizzare potenza computazionale della vittima per generare criptovalute 1. Il tutto avviene seguendo un basso profilo, tramite programmi infetti che vengono installati sul computer o, come più recentemente si è affermato, attraverso script malevoli presenti all'interno di pagine web comuni. La differenza tra l'utilizzo di programmi eseguibili e gli script presenti online sta principalmente nella loro rintracciabilità e facilità di utilizzo, infatti quelli che operano attraverso il web sono più difficili da individuare e analizzare rispetto a programmi che devono necessariamente essere installati dall'utente e che, quindi, sono sotto l'occhio diretto di antivirus e delle politiche dettate dal sistema operativo.
L'utilizzo della GPU in all'interno di browser è ad oggi possibile, anche se non largamente diffuso, tramite l'utilizzo di alcune librerie e specifiche web. *OpenGL* (*Open Graphics Library*) è considerata ormai lo standard per quanto riguarda la grafica tridimensionale in sistemi operativi *Unix-like*, in particolare, essa è stata pensata per architetture parallelizzabili come le GPU e definisce delle *API* per applicazioni che operano in ambienti 3D. Su questo standard sono basate altre librerie che permettono l'utilizzo della GPU in ambito web come *WebGL*. Quest'ultima è una *Web-based Graphics Library* ovvero una libreria pensata per la gestione di elementi grafici all'interno del web, essa fornisce delle *API* per grafica 3D in un contesto *HTML5* tramite l'utilizzo del *Document Object Model (DOM)*. Da *WebGL* si arriva poi a *GPU.js*, ovvero una libreria implementata interamente attraverso il linguaggio *JavaScript* e pensata per sfruttare l'accelerazione hardware delle GPU attraverso l'utilizzo del medesimo linguaggio.
L'utilizzo di un linguaggio di scripting web come *Javascript* è a sua volta rilevante in quanto esso può essere utilizzato per portare a termine attacchi che sfruttano vulnerabilità di tipo *Cross Site Scripting (XSS)*. Gli attacchi di tipo *XSS* sono tra i più diffusi all'interno del web, infatti, secondo il report di Positive Technologies Security [20], la percentuale di questi attacchi è salita dall'occupare il 77,9% nel 2017 all'88,5% di tutte le tipologie di attacchi web registrate nel 2018. Secondo Precise Security [21] invece, in rapporto a tutti i tipi di attacchi sferrati verso le grandi compagnie di Europa e Nord America nel 2019, la percentuale di attacchi di *Cross Site Scripting* risulta essere del 39%, superando più del doppio la percentuale degli attacchi di *SQL Injection*.
L'*XSS* consiste nell'introdurre del codice arbitrario lato client all'interno dei siti web con il fine di eseguire una serie di attacchi rivolti ai visitatori. Questa vulnerabilità affligge i siti dinamici che non effettuano un controllo sugli input lato client, il che porta ad una "fusione" tra il payload, ovvero il codice inserito dall'attaccante, e il codice reale della pagina. Questa tipologia di attacchi si suddivide a sua volta in tre gruppi:

- XSS reflected: il payload viene eseguito solo nel momento dell'iniezione;
- XSS stored: il payload viene salvato all'interno della struttura dell'applicativo ed eseguito ad ogni accesso del contenuto;
- XSS DOM-based: il codice sorgente e la risposta del server non vengono modificate, il payload viene eseguito a runtime senza inoltrare richieste al server ma utilizzando il codice già presente nella pagina.

Esistono vari strumenti in grado di analizzare la struttura delle pagine web e di identificare vulnerabilità *XSS*, come ad esempio *XSStrike* [23], *Traxss* [14] e *XSSer* [7]. Questi sono tutti tool open-source di analisi sviluppati in *Python*, il più interessante è però *XSSer*, uno dei tool preinstallati presenti nelle distribuzioni *Linux* atte al *penetration testing*. Si è quindi pensato di utilizzare uno di questi tool per individuare in modo semi-automatico o del tutto automatico le vulnerabilità *XSS* all'interno di siti web.

A. Obiettivi e struttura dell'articolo

L'obiettivo principale di questo articolo è quindi quello di mostrare come sia possibile effettuare un attacco di tipo *cryptojacking* basato sfruttando la potenza computazionale della GPU della vittima, il tutto applicato in un contesto web. Inoltre una volta individuato il vettore di attacco si descriverà l'implementazione di una possibile forma di difesa.
La restante parte dell'articolo è strutturato nel modo seguente: sezione II introduce quello che è lo stato dell'arte per quanto riguarda gli attacchi *cryptojacking*, sezione III-A descrive come è possibile utilizzare la GPU in un contesto web, sezione III-B mostra l'injection automatizzata del payload, sezione III-C descrive le caratteristiche dell'ambiente di test utilizzato, sezione III-D individua una soluzione difensiva user-friendly, sezione IV-A analizza i risultati ottenuti, sezione IV-B conclude e riassume l'articolo ed infine sezione IV-C identifica quelli che sono le azioni future che saranno effettuate.

II. Stato dell'arte

Le GPU grazie alla loro potenza offrono, nella maggior parte dei casi, dei ricavi maggiori nel campo del *mining* rispetto alle CPU, di contro però non sono facilmente utilizzabili in ambito web. Ciò ha portato infatti ad uno sviluppo maggiore dei *cryptominers* basati su CPU e, di conseguenza, a sistemi di difesa che si concentrano su di essi. Musch et al. [17] e Saad, Khormali e Mohaisen [24] mostrano la diffusione dei *cryptominers* basati su CPU all'interno del web, inoltre analizzano l'efficacia di tecniche difensive di blacklisting che risultano però essere una protezione non sempre efficiente sebbene pratica. Wang et al. [29] mostra in seguito un metodo di analisi di firme basate su istruzioni della CPU durante l'esecuzione dei moduli *WebAssembly*. Konoth et al. [13] introduce invece *MineSweeper* basato anch'esso sul principio di firme ma con l'aggiunta del rilevamento di eccessive chiamate di sistema di natura crittografica legate agli algoritmi di mining durante l'esecuzione di un programma. Kharraz et al. [12] dimostra invece che come l'analisi della CPU generi una grande quantità di falsi positivi all'interno della navigazione web. Un differente approccio viene fornito da Tahir et al. [26] che presenta *MineGuard*, un sistema che tramite l'analisi delle prestazioni hardware rileva l'esecuzione di alcuni algoritmi di *mining*.

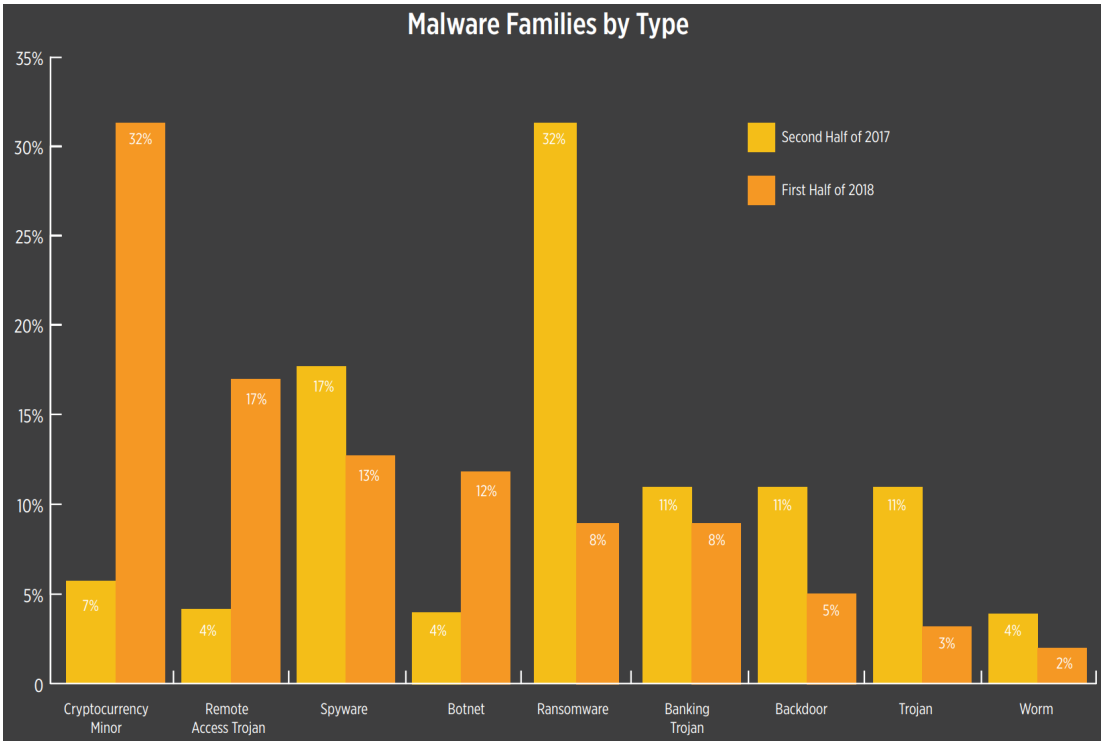

Figura 1. Top Malware Families by type, 2018, *Skybox Vulnerability Report Trends* [25]

MineGuard monitora costantemente l'hardware del computer e inoltre offre un'ottima soluzione sia in termini di efficienza che di utilizzo di risorse per la sua esecuzione. Belkin, Gelernter e Cidon [1] offre infine una soluzione dedicata esclusivamente a *cryptominers* di GPU che risulta essere decisamente più pratica in ambito web rispetto alle altre, andando a limitare le pagine web che utilizzano *WebGL* [11].

III. Metodologie utilizzate

A. GPU-WEB based Cryptominer

L'utilizzo di *Graphics processing units* all'interno di pagine web non è molto comune nonostante negli ultimi anni sia nettamente aumentata la complessità degli elementi grafici presenti all'interno dei browser. Ciononostante sono state sviluppate delle *Application Programming Interface* per garantire l'operabilità di processori grafici in un contesto web. In particolare ciò è reso possibile da *OpenGL* che a sua volta è utilizzato da librerie come *WebGL* e *GPU.js*. Quest'ultima è pensata per computare fino a quindici volte più rapidamente della sola CPU operazioni parallelizzabili come il caricamento di elementi grafici complessi o la moltiplicazione di matrici di grandi dimensioni [5]. Operando in un contesto di grafica tridimensionale, queste implementazioni permettono di effettuare operazioni anche complesse attraverso la GPU dell'utente che visualizza la pagina web dove sono utilizzate. Si è scelto di optare per l'utilizzo di *GPU.js* per lo sviluppo del payload malevolo in quanto, dato il fatto che l'implementazione avviene attraverso il linguaggio *Javascript*, si presta bene all'injection di codice malevolo. Un enorme svantaggio riscontrato durante lo sviluppo è stato il fatto che le librerie grafiche non permettono di eseguire qualsiasi tipo di operazione, infatti, per quanto riguarda *GPU.js* non è prevista la possibilità di effettuare operazioni di rete come le *HTTP requests*. Ciò costituisce una grande limitazione quando si tratta di operazioni che necessitano costantemente di dati esterni alla pagina web per l'esecuzione. Questa limitazione rende infatti problematico il collegamento con le blockchain e di conseguenza il mining di criptovalute. Sarebbe comunque possibile effettuare mining attraverso *GPU.js* nel caso in cui vi sia una re-implementazione dello *Stratum protocol* [22] utilizzato per il dialogo con le pool di mining e le blockchain. Pertanto nel corso di questo articolo è stato utilizzato come payload "malevolo" uno script che permette l'utilizzo della GPU per il calcolo di due matrici quadrate di dimensione 8096 righe e colonne composte da valori generati in modo casuale con lo scopo di dimostrare la fattibilità dell'attacco ma non l'effettiva esecuzione. Lo script prevede inoltre l'importazione dinamica della libreria GPU.js e l'avvio automatico delle operazioni di calcolo appendice A.

B. Cryptominer injection automatizzata

Come precedentemente si è detto nella sezione III-A, l'utilizzo della libreria *GPU.js*, e in particolare del linguaggio *JavaScript*, non è dovuta solo al fatto che è uno dei più utilizzati e supportati all'interno del web [28], ma soprattutto al fatto che è tra i principali utilizzati per l'iniezione di codice malevolo. Spesso infatti gli attacchi di tipo *XSS* sono portati a termine tramite iniezione di codice *Javascript* [19].

Per l'effettiva esecuzione dell'attacco si è sviluppato un tool di injection automatica in linguaggio *BASH* e basato sul software *XSSer*. Il tool sviluppato è basato su un file descrittore (appendice A) nel quale sono presenti le informazioni relative alle modalità di scansione e di attacco. Il tool prevede cinque modalità principali:

- GET, utilizzata per *HTTP requests* di tipo "GET";
- POST, utilizzata per *HTTP requests* di tipo "POST";
- ALL, utilizzata per vettori di attacchi su tutto il sito vittima;
- AUTO, automatica che utilizza delle euristiche;
- DEFAULT, standard molto simile alla modalità ALL.

Oltre alla definizione della modalità di scanning e di attacco è necessario definire *endpoint* e parametri necessari per le chiamate di tipo "GET", analogamente per le chiamate "POST", ponendo attenzione nell'inserire in entrambi i casi il placeholder "XSS" al posto di almeno un valore. Il placeholder sarà poi riconosciuto dal tool e, durante l'esecuzione, sarà rimpiazzato con i payload di scan e infine il payload di attacco. Vi è inoltre la possibilità di definire i *cookies* di sessione necessari per la corretta esecuzione su alcune applicazioni web. I *cookie* possono essere inseriti nella colonna finale del file descrittore e saranno poi automaticamente gestiti dal tool durante le varie fasi. L'esecuzione del tool può essere inoltre lanciata secondo due sotto-modalità ovvero scanning e injection. Le sotto-modalità sono indipendenti tra loro, infatti esse sono utilizzabili sia singolarmente che in combinazione. Infine, in entrambi i casi è possibile definire un payload personalizzato, in caso di scansione sarà utilizzato per ricercare vulnerabilità mentre in caso di injection sarà invece iniettato seguendo i vettori di attacco precedentemente individuati. Le modalità precedentemente descritte sono effettuate operando in *multithread*, per velocizzare l'esecuzione e utilizzando un *fake user-agent* con l'intento di oltrepassare i controlli e le limitazioni di accesso ripetuto per la maggior parte dei siti web.
Il tool in seguito all'esecuzione dello scanning di vulnerabilità genera un file di report al quale viene associato un controllo di integrità. Il report contiene tutti i test e i vettori di attacco analizzati e le informazioni riguardanti quali il risultato dell'operazione e una lista di attacchi andati a buon fine qualora ce ne siano. Il file di report viene infine consultato in fase di attacco per determinare il vettore di attacco con accuratezza più elevata e sfruttarlo per iniettare il payload malevolo.
Il payload finale non coincide esattamente con il payload contenente le operazioni computazionali complesse, bensì consiste nell'iniezione di un tag "¡script¿" avente come sorgente un *endpoint* che fornirà integralmente lo script (appendice A). Questo processo è essenziale in quanto si vogliono evitare limitazioni sul numero di caratteri inserendo così un payload di dimensioni contenute che però sortirà lo stesso effetto rispetto al payload malevolo basato su *GPU.js*.

C. Testing environment

L'ambiente sul quale si sono effettuati i test di validazione dell'attacco è stato creato appositamente utilizzando *Damn Vulnerable Web App (DVWA)*[6]. Quest'ultima è un'applicazione web codificata utilizzando *PHP* e *MySQL* il quale obiettivo principale è quello di creare un ambiente appositamente vulnerabile ma controllato che permetta lo studio delle problematiche di sicurezza più frequenti in ambito web e che divide le vulnerabilità in tre diverse difficoltà.
Al fine di ridurre al minimo le possibilità di danneggiamento del sistema in uso, e soprattutto per permettere la riproducibilità dell'attacco, l'applicativo *DVWA* è stato eseguito all'interno di un sistema di "containerizzazione" quale *Docker*[15]. Il container è stato predisposto per essere facilmente ricreato ed è inoltre stato aggiunto il supporto all'utilizzo della utility *docker-compose* per una ancor più facile interazione con il container di test. Infine è stato sviluppato un applicativo server con lo scopo di fungere da *dispatcher* per quanto riguarda il payload malevolo. Il server è stato realizzato in *Python* nella sua terza versione[27] e in particolare con il framework *Flask*[9]. Il *dispatcher* sviluppato ha il solo scopo di simulare il server dell'attaccante o un qualsivoglia fornitore di un servizio capace di fornire su richiesta il payload malevolo con *MIME Type: text/javascript*[16].

D. Soluzione difensiva non invasiva

Analizzando i risultati ottenuti da Wang et al. [29] una tecnica difensiva molto efficace consiste nel monitorare costantemente l'utilizzo dell'hardware del computer vittima per individuare l'attacco e interrompere il processo di esecuzione. Nonostante il *monitoring* della componentistica possa essere ridotto al riconoscimento delle sole funzioni crittografiche, come mostrato da Konoth et al. [13], un approccio di questo tipo risulta essere sì efficace ma anche invasivo e richiede la continua esecuzione del processo di controllo in background con conseguente utilizzo di risorse.
Un approccio più *user-friendly* è invece quello individuato da Belkin, Gelernter e Cidon [1], ovvero analizzare durante la navigazione web l'utilizzo di funzioni e librerie grafiche pensate per GPU e bloccarne l'esecuzione. Questa tecnica difensiva seppur "semplice" offre molti vantaggi, primo tra tutti risolve il problema della *code obfuscation* in quanto non analizza il codice eseguito direttamente, che potrebbe essere offuscato, ma solo l'accesso alle librerie web che sono richieste dalla pagina. Pertanto è stata impostata una tecnica difensiva basata sul blocco di quelle librerie web che permettono l'utilizzo di processori grafici. Il risultato di tale operazione è stato lo sviluppo di un'estensione web per il browser *Google Chrome* che neutralizza l'utilizzo della libreria *GPU.js* bloccando la richiesta di download del codice malevolo sul nascere. È importante far notare che la scelta di un'estensione è stata presa considerando anche la facilità di utilizzo da parte di un utente inesperto, infatti, le estensioni dei browser permettono lo sviluppo di interfacce semplici da utilizzare per l'utente e che permettono la personalizzazione di operazioni come il blocco dell'estensione solo su determinati siti o l'aggiunta di ulteriori librerie considerate non affidabili, così da garantire una maggiore personalizzazione e scalabilità.

IV. Analisi dei risultati e implementazioni future

A. Risultati

I test effettuati si sono svolti andando ad utilizzare il tool di analisi e di attacco sviluppato impostando come target le pagine vulnerabili esposte da *DVWA*. Nel dettaglio ci si è concentrati sulle vulnerabilità di tipo *XSS stored* in quanto esse erano quelle più interessanti per questo tipo di attacco in un contesto reale. Il tool ha portato al corretto sfruttamento delle vulnerabilità in tutti e tre i livelli di difficoltà proposti da *DVWA*.
Una volta provata l'efficacia dell'attacco si è testata anche l'efficacia del sistema di difesa che è correttamente riuscito a bloccare l'utilizzo della libreria *GPU.js*. Il blocco della libreria è stato prontamente rilevato ad ogni caricamento della pagina ed è stata correttamente bloccata l'esecuzione dello script precedentemente iniettato. Si può quindi affermare che i risultati ottenuti siano stati soddisfacenti per quanto riguarda l'attacco dal punto di vista concettuale, si è dimostrato che questo approccio è effettivamente utilizzabile e che è inoltre anche neutralizzabile. I risultati e il codice sviluppato sono *open-source* e disponibili sulla piattaforma *GitHub* sotto il nome di *Hi-Jacket*[1]. Quest'ultimo è diviso principalmente in tre sezioni:

- Bomb, il payload di test facente utilizzo della GPU;
- Remote control, il tool di injection automatica basato su *XSS*;
- Defuser, l'estensione difensiva del browser *Google Chrome*.

B. Conclusione

In quest'ultimo decennio si è molto discusso di monete digitali e delle così dette criptovalute in seguito all'avvento delle blockchain. In seguito l'interesse dalle criptovalute si è espanso fino a generare un grande interesse per le tecniche di *mining* con le quali le *cryptocurrencies* vengono generate. Di conseguenza il *mining* è diventato di grande interesse anche per i malintenzionati portando allo sviluppo di attacchi di basso profilo come il *cryptojacking*.
In questo articolo è stato mostrato come sia possibile effettuare un attacco di *cryptojacking* che utilizzi la potenza della *Graphics processing unit* all'interno del browser web. Nonostante il *cryptominer* ideato non sia stato completato a causa delle limitazioni delle tecniche utilizzate, il payload di test dimostra comunque la fattibilità di esecuzione dell'attacco basato su GPU. Si è inoltre sviluppato un tool di injection automatica del payload malevolo tramite attacchi di tipo *Cross Site Scripting*, la cui bontà è stata testata all'interno di un ambiente *Docker* contenente un'istanza di *Damn Vulnerable Web App*. Il testing ha prodotto risultati soddisfacenti riuscendo ad analizzare, individuare e sfruttare le vulnerabilità presenti nelle tre diverse difficoltà. In seguito si è sviluppato un sistema di difesa *user-friendly* pensato per l'utilizzo comune di utenti inesperti ma comunque attento alle loro esigenze di personalizzazione.

[1]https://github.com/ZappaBoy/hi-jacket

C. Sviluppi futuri

Il prossimo passo per il completamento effettivo dell'attacco sarà quello di integrare lo *Stratum Protocol* all'interno del payload malevolo da iniettare all'interno dei siti target. In questo modo si potrà inoltre confrontare il *mining* tramite browser e quello classico così da studiarne i pro e i contro. Ciò oltre ad offrire un metodo di paragone tra le due modalità potrebbe portare ad un nuovo tipo di mining nel caso in cui le prestazioni siano simili. Se le due modalità si rivelassero equivalenti si potrebbe infatti semplificare l'accesso al mondo del *mining* che potrebbe essere effettuato semplicemente navigando ad un indirizzo web.
Infine una ulteriore miglioria potrebbe essere quella di permettere una maggiore personalizzazione e l'implementazione di una grafica minimalista per quanto riguarda l'estensione di difesa contro questo tipo di attacchi. In questo modo si avrebbe un prodotto finito e accessibile agli utenti inesperti che potrebbe essere integrato come controllo durante la navigazione.

RIFERIMENTI BIBLIOGRAFICI

[1] Alex Belkin, Nethanel Gelernter e Israel Cidon. "The Risks of WebGL: Analysis, Evaluation and Detection". In: *European Symposium on Research in Computer Security*. Springer. 2019, pp. 545–564.

[2] *Bitcoin Wiki*. https://en.bitcoinwiki.org/wiki/Bitcoin_history.

[3] CoinGecko. *All Time High list*. https://www.coingecko.com/it/monete/ath.

[4] *Coinlore all Coins*. https://www.coinlore.com/all_coins.

[5] GPU.js community. *GPU.js benchmark*. https://gpu.rocks/#/benchmark.

[6] digininja. *Damn Vulnerable Web App*. https://github.com/digininja/DVWA.

[7] epsylon. *XSSer*. https://github.com/epsylon/xsser.

[8] gpujs. *GPU.js*. https://github.com/gpujs/gpu.js.

[9] Miguel Grinberg. *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2018.

[10] The Khronos Group Inc. *OpenGL*. https://www.opengl.org/.

[11] The Khronos Group Inc. *WebGL*. https://github.com/KhronosGroup/WebGL.

[12] Amin Kharraz et al. "Outguard: Detecting in-browser covert cryptocurrency mining in the wild". In: *The World Wide Web Conference*. 2019, pp. 840–852.

[13] Radhesh Krishnan Konoth et al. "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*. 2018, pp. 1714–1730.

[14] M4cs. *Traxss*. https://github.com/M4cs/traxss.

[15] Dirk Merkel. "Docker: lightweight linux containers for consistent development and deployment". In: *Linux journal* 2014.239 (2014), p. 2.

[16] Developer Mozilla. *MIME types (IANA media types)*. https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types.

[17] Marius Musch et al. "Web-based Cryptojacking in the Wild". In: *arXiv preprint arXiv:1808.09474* (2018).

[18] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. Rapp. tecn. Manubot, 2019.

[19] OWASP. *Cross Site Scripting (XSS)*. https://owasp.org/www-community/attacks/xss/.

[20] Positive Technologies Security. *Penetration testing of corporate information systems: statistics and findings, 2019*. https://www.ptsecurity.com/ww-en/analytics/corp-vulnerabilities-2019/.

[21] Precise Security. *Cross-Site Scripting (XSS) Makes Nearly 40% of All Cyber Attacks in 2019*. https://www.precisesecurity.com/articles/cross-site-scripting-xss-makes-nearly-40-of-all-cyber-attacks-in-2019/.

[22] Ruben Recabarren e Bogdan Carbunar. "Hardening stratum, the bitcoin pool mining protocol". In: *arXiv preprint arXiv:1703.06545* (2017).

[23] s0md3v. *XSStrike*. https://github.com/s0md3v/XSStrike.

[24] Muhammad Saad, Aminollah Khormali e Aziz Mohaisen. "End-to-end analysis of in-browser cryptojacking". In: *arXiv preprint arXiv:1809.02152* (2018).

[25] *Skybox Vulnerability Report Trends*. https://lp.skyboxsecurity.com/rs/440-MPQ-510/images/Skybox_Report_Vulnerability_Threat_Trends_2018_Mid-Year_Update.pdf.

[26] Rashid Tahir et al. "Mining on someone else's dime: Mitigating covert mining operations in clouds and enterprises". In: *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer. 2017, pp. 287–310.

[27] Guido Van Rossum e Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[28] W3Techs. *Usage statistics of JavaScript as client-side programming language on websites*. https://w3techs.com/technologies/details/cp-javascript.

[29] Wenhao Wang et al. "Seismic: Secure in-lined script monitors for interrupting cryptojacks". In: *European Symposium on Research in Computer Security*. Springer. 2018, pp. 122–142.

[30] *Wired - Trasformazione di Bitcoin*. https://www.wired.it / economia / finanza / 2019 / 01 / 03 / bitcoin - 2009 - trasformazione-storia/.

APPENDICE

GPU.JS PAYLOAD

```
let li = document.createElement('script');
li.src = 'https://unpkg.com/gpu.js@latest/dist/gpu-browser.min.js';
li.async = false;
let s = document.getElementsByTagName('script')[0];
s.parentNode.insertBefore(li, s);

setTimeout(() => {
   const gpu = new GPU()

   const DIMENSION = 8096
   const generateMatrices = () => {
      const matrices = [[], []]
      for (let y = 0; y < DIMENSION; y++) {
         matrices[0].push([])

         matrices[1].push([])
         for (let x = 0; x < DIMENSION; x++) {
            matrices[0][y].push(Math.random())
            matrices[1][y].push(Math.random())
         }
      }
      return matrices
   }

   const multiplyMatrix = gpu.createKernel(function (a, b, dim) {
      let sum = 0;
      for (let i = 0; i < dim; i++) {
         sum += a[this.thread.y][i] * b[i][this.thread.x];
      }
      return sum;
   }).setOutput([DIMENSION, DIMENSION])
   const matrices = generateMatrices()
   const out = multiplyMatrix(matrices[0], matrices[1], DIMENSION)
   console.log(out)
}, 3000);
```

INJECTED PAYLOAD

```
<script type="text/javascript" src="http://127.0.0.1:5000/get_payload"></script>
```

ESEMPIO FILE DESCRITTORE

```
GET      http://xss-game.appspot.com                    /level1/frame?query=XSS
POST     http://localhost:8060/vulnerabilities/xss_s/index.php
    txtName=hijacked&mtxMessage=XSS&btnSign=Sign+Guestbook
    PHPSESSID="ef31f7c8e8c5b5dac8caf4d2a26b9e0f;;security=low"
ALL      http://localhost:8060/vulnerabilities/xss_s/index.php
AUTO     http://xss-game.appspot.com                    /level1/frame?query=XSS
# Commented row DEFAULT http://xss-game.appspot.com                 /level1/frame?query=XSS
```