Report progetto

Valentina Ceoletta VR407794
Mattia Zanotti VR411086
Nicolò Zenari VR412613

26 settembre 2018

Indice

Capitolo 1

Descrizione del progetto

Il progetto consiste nell'esplorare il web con l'obiettivo di raccogliere codice JavaScript considerato malevolo. In linea generale, il progetto inizia attraverso la collezione degli URL considerati malevoli dalla comunità *hpHosts*. In secondo luogo, tali indirizzi web sono passati a *heritrix*, un crawler in grado di effettuare il download delle pagine web e di tutte le risorse necessarie. Successivamente, i siti web scaricati con successo, vengono utilizzati come base per interrogare *Virtus Total* e *Google Safe Browsing*, due tool online in grado di fornire informazioni legate all'essere malevolo o meno. Il linguaggio scelto per completare il progetto è Python, un linguaggio ad alto livello semplice, flessibile e adatto alla realizzazione di script.

1.1 File di configurazione

All'interno della directory principale del progetto è presente il file *config.json*, un file in cui sono strutturate tutte le informazioni necessarie per il corretto funzionamento del progetto. Di seguito, per ogni configurazione segue la spiegazione dell'utilizzo:

- *out_dir_path*: il path della directory in cui salvare tutte le risorse ottenute;
- *hosts_dir_path*: directory in cui andranno salvate tutte le risorse di ogni host scaricato;
- *heritrix_dir_path*: il nome della directory in cui lo script *warcExtractor* crea le directory degli host;
- *javascript_dir_path*: il nome della directory in cui lo script *jsExtractor* salva i file javascrit estratti;
- *GSB_dir*: directory in cui vengono i creati i file *bodyX.josn* necessari per interrogare Google Safe Browsing;

- *hosts_json_filename*: nome del file JSON prodotto dallo script *getHosts* contenente tutti gli host ottenuti da HpHosts con le relative informazioni;
- *seeds_filename*: nome del file prodotto dallo script *getHosts* contenente, riga per riga, l'URL degli host ottenuti da HpHosts;
- *host_file_url*: URL di HpHosts (il parametro 0 indica la classe, mentre 1 indica la pagina);
- *download_from_date*: data di pubblicazione da cui iniziare il download;
- *download_to_date*: data di pubblicazione entro il quale terminare il download;
- *vt_requests_url*: URL di VirusTotal necessario per inviare la richiesta;
- *vt_report_url*: URL di VirusTotal necessario per ottenere il report;
- *vt_api_key*: chive API di VirusTotal;
- *class_to_download*: array di classi di HpHosts d'interesse;
- *json_GSB_filename*: nome del file JSON necessari allo script *gsb*;
- *GSB_url*: URL di Google Safe Browsing;
- *GSB_key*: chiave API di Google Safe Browsing;
- *GSB_clientId*: nome dell'azienda (obbligatorio) per la richiesta a GSB;
- *GSB_clientVersion*: versione del client Google Safe Browsing;
- *GSB_threatTypes*: array di minacce (parametro di interrogazione di GSB);
- *GSB_platformTypes*: array di piattaforme (parametro di interrogazione di GSB);
- *GSB_threatEntryTypes*: tipologie ti entry passate a GSB attraverso i file JSON definiti in *json_GSB_filename*;
- *last_check*: ultimo controllo eseguito dallo scirpt *getHosts*.

1.2 getHosts

Lo script *getHosts* si occupa di richiedere e parserizzare le pagine HTML fornite da HpHosts con l'obiettivo di ottenere gli host desiderati. HpHosts è una repository di URL ritenuti malevoli da una comunità che viene aggiornata quotidiamente. Tra tutti gli host pubblicati sul sito, si è deciso di scaricare solo quelli che rientrano nella categoria **EMD**. La tipologia d'interesse viene specificata nel file di configurazione e più precisamente nell'array *class_to_download*, mentre il periodo temporale necessario viene specificato agendo sulle variabili *download_from_date*, *download_to_date* e *last_check*.

Lo script scarica la prima pagina solamente per ricavare il numero totale di pagine. Una volta ottenuta l'informazione, sulla base dei core disponibili nella macchina, vengono create tante thread quante il numero di core. Ogni thread ha il compito di interrogare e parserizzare tutte le pagine il cui numero è multiplo della prima pagina indicata (ad esempio, se si utilizzano 2 thread, la prima dovrà lavorare con le pagine 1,3,5,7, etc mentre la seconda con le pagine 2,4,6,8, etc).

Per quanto riguarda la parserizzazione, lo script estrapola la tabella HTML in cui sono contenuti tutti gli hosts. Per ogni host la cui classe corrisponde a una tra quelle desiderate, estrapola hostname, IP, classe, data di pubblicazione su hpHosts e data di aggiunta. Tutti gli host creati sono aggiunti ad una lista utile per la scrittura su file.

Ogni thread, come accennato precedentemente, si occupa di richiedere la pagina HTML e di parserizzarla in modo da estrapolare tutte le informazioni utili. Esse terminano se finiscono le pagine di loro competenza o se in una pagina è presente almeno un host già trattato [1] oppure la cui data di pubblicazione è antecedente o seguente il periodo d'interesse.

Per ogni host che soddisfa le condizioni specificate nel file di configurazione, lo script *hostInfo* invocato dalle thread di *getHosts* si occupa di creare la seguente struttura ad albero delle directory con i relativi file di competenza:

```
out
└── hosts
    ├── hostname1
    │   └── info.json
    ├── hostname2
    │   └── info.json
    └── ...
```

[1] La data dell'ultimo controllo è specificata nel file di configurazione sotto la voce *last_check*

1.2.1 Output

Il risultato dello script è:

- la directory di output;
- la directory hosts;
- una directory per ogni host;
- il file *info.json* all'interno della rispettiva cartella di ogni host.

1.2.2 Tempi di esecuzione

I fattori che incidono sul tempo di esecuzione dello script sono **(1)** il numero di core messi a disposizione dal computer che si utilizza e **(2)** il numero di host che si vanno a scaricare in quanto il tempo impiegato da *hpHosts* per creare una pagina è all'incirca di 3/5 secondi.

1.3 Crawler

Il crawler è un software che analizza i contenuti di una rete in modo metodico e automatizzato, si potrebbe definire un *bot* che acquisisce una copia testuale di tutti i documenti visitati (HTML principalmente) e le salva in un file di output. Dalle recensioni web e dai giudizi degli utenti, utilizziamo Heritrix [**?**] nella versione 3.1.1 come crawler. Essendo altamente automatizzato è possibile impostare i parametri a piacimento nel file di configurazione *crawler-beans.cxml*. Per il nostro obiettivo sono stati modificati i seguenti campi per poter estrarre solo le pagine utili:

- **org.archive.modules.seeds.TextSeedModule**: impostato per estrarre gli host da visitare dal file *seeds.txt*, collocato nella directory principale del jobs;
- **org.archive.module.deciderules.DecideRuleSequence**: applicate una serie di regole in cascata tra cui:
  1. rifiuto di tutte le pagine
  2. accetazione delle sole pagine HTML e JS
  3. accettazione delle sole pagine a profondità 1 hops dal dominio principale;
- **org.archive.modules.Extractor**: applicazione dei soli estrattori HTTP, HTML e JS;
- **org.archive.modules.writer.WARCWriterProcessor**: salvataggi del crawling nel formato *.warc*;

La scelta della profondità è stata dettata dal fatto che outlink a potenziali pagine o a potenziali script malevoli non sono raggiungibili con un numero di hop prefissati. Abbiamo deciso quindi di estrarre tutte le pagine HTML e JavaScript del dominio e poi le relative pagine collegate. Non tutti i siti permettono il crawling del proprio dominio, infatti Heritrix è stato impostato per non violare le direttive del file *robots.txt* in cui vengono specificati dagli amministratori del sito, quali path sono accessibili e quali no. Alcuni vietano il completo crawling, perciò è possibile che alcuni host non vengano scaricati con successo ma ritornino un codice di errore relativo (per la lista completa visitare il seguente link: `https://github.com/internetarchive/heritrix3/wiki/Status%20Codes`). L'avvio del crawler avviene dapprima da linea di comando:

```
$HERITRIX_HOME/bin/heritrix -a utente:password
```

e successivamente recandosi tramite web browser alla seguente pagina https://localhost:8443. Si seleziona il job corrispondente [2], si compila il crawler e poi lo si avvia premendo il bottone relativo.L'output che ne deriva è un file di tipo warc.gz.

Tale file viene elaborato attraverso lo script Python *warcExtractor.py*. Utilizzando un tool per l'estrazione delle informazioni dal file *.warc* [**?**] , per ogni host scaricato con successo lo script crea una directory nominata *heritrix* all'interno della cartella dell'host (vedi 1.2) in cui salvare tutti i file associati. Per i siti in cui non è stato possibile il crawling, lo script provvederà a rimuoverne la directory creata in precedenza da *getHosts*. Come ultimo lavoro, tutti gli host la cui operazione di raccolta è stata conclusa con successo verranno inseriti nel file *seeds.txt* necessario a VirusTotal ed infine invocherà lo script *hostInfo* per creare i file *bodyN.json* necessari al modulo GSB.

Dopo la creazione delle directory, si avvia lo script Python *jsExtractor.py* che provvede a creare una sub-directory javascript se è presente del codice JavaScript all'interno delle pagine scaricate. Per poter estrarre il codice contenuto all'interno del tag ⟨script⟩ viene utilizzata la libreria *BeautifulSoup* che fornisce dei metodi per la ricerca del tag desiderato all'interno del documento e ritorna la lista del contenuto. Tutte le parti di script o referenze a script esterni alla pagina vengono salvati in un unico file dal nome progressivo *scriptN.js* all'interno della sub-directory creata precedentemente.

1.3.1 Tempi di esecuzione

I fattori che incidono sul tempo di esecuzione dello script sono **(1)** il numero di thread impostate nel crawler per poter scansionare la rete **(2)** il numero

[2] Si possono creare diversi job personalizzandoli a piacere

di host da visitare **(3)** il numero di host scaricati con successo dai quali estrarre i file associati **(4)** il quantitativo di codice javascript incluso nelle pagine scaricate

Al primo avvio il numero di host sarà certamente nell'ordine delle migliaia e il crawler richiede molte ore di processamento. Ai successivi avvii il tempo si accorcia.

1.4 Google Safe Browsing (GSB)

Uno script Python si occupa di interrogare Google Safe Browsing [**?**] utilizzando il file *bodyN.json*. Tali file si trovano seguendo il seguente albero:

```
out
├── hosts
└── gsb
    ├── body0.json
    ├── body1.json
    └── ...
```

L'implementazione della creazione di più file è stata necessaria in quanto Google impone l'interrogazione di al più 500 host per volta. All'interno dei Json sono contenute le seguenti informazioni:

- *ClientInfo*: le informazioni sul client che esegue la Safe Browsing API request;
- *threatTypes*: il tipo di minaccia che si vuole controllare;
- *platformTypes*: quali piattaforme controllare;
- *threatEntryTypes*: il formato delle minacce che si vogliono analizzare (url, executable);
- *threatEntries*: lista delle minacce da analizzare.

Il server di GSB esegue il join tra *threatTypes*, *platformTypes* e *threatEntryTypes* ritornando quindi un file Json con molte voci ripetute. L'output viene quindi semplificato, indicando per ogni URL, le piattaforme per il quale risulta essere malevolo (es. Linux, Windows, OSX, ...) e il tipo di tipo di minaccia rilevata (Social Engineering, Malware, ...). Infine, tutte le informazioni ottenute sono salvate nel file *gsb-tags.json*

1.4.1 Tempi di esecuzione

Il tempo di esecuzione di Google Safe Browsing non allunga i tempi totali dell'intero progetto. L'esecuzione è nell'ordine di qualche secondo.

1.5 Virus Total (VT)

Lo script Python interroga Virus Total [**?**] utilizzando le sue API pubbliche. Virus Total nella sua versione gratuita è un servizio che analizza file sospetti o URLs e ne permette la rapida identificazione ed etichettatura. Dapprima si analizza il file *seeds.txt* che contiene tutti gli hosts da verificare, poi si effettua la richiesta divisa nel seguente modo:

1. una richiesta **post http** al server contenente l'url da analizzare;
2. dopo l'analisi è necessaria una richiesta di **report** contenente un serie di informazioni; verrà applicato in secondo luogo un filtro per la selezione di solamente alcuni dati:
   - *vt_scan_date*: la data in cui è stata fatta la scansione su VT;
   - *positives*: il numero di antivirus che hanno riconosciuto l'URL analizzato come malevolo;
   - *total*: il numero complessivo di antivirus che hanno analizzato l'URL.

Viene salvato tutto nel file *vt-tags.json*.

1.5.1 Tempi di esecuzione

Utilizzando le API pubbliche e gratuite di Virus Total è possibile fare al più 4 richieste al minuto, quindi è possibile analizzare al massimo 2 siti al minuto dato che ogni sito necessita di due chiamate al server.

1.6 Statistiche

1.6.1 Google Safe Browsing

Per lanciare le statistiche in merito alla quantità di host che Google Safe Browsing è riuscito a taggare è sufficiente lanciare lo script *gsbStata*. Ogni host viene taggato solamente se sui server vengono trovate le informazioni necessarie.

Dal 13 al 31 Agosto 2018, su 288 host collezionati, Google Safe Browsing ha taggato con successo 11 host.

1.6.2 Virus Total

Per lanciare le statistiche in merito alla quantità di host che Virus Total ha classificato come non malevolo, è necessario lanciare lo script *vtStata*. Ogni hosts verrà fatto analizzare da un serie di antivirus, ottenendo alla fine una percentuale sul riconoscimento dell'host come malevolo, ovvero Virus

Total indicherà quanti antivirus hanno riconosciuto l'host come malevolo e il numero di antivirus che hanno analizzato l' host.

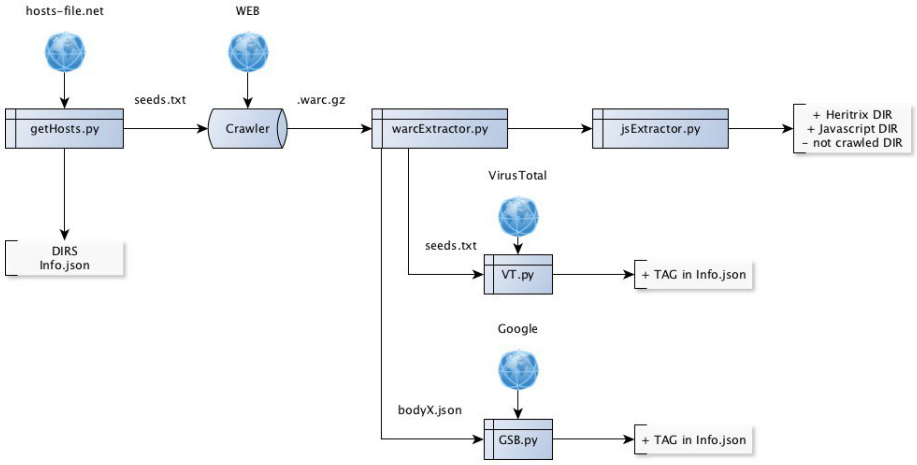
Dal 13 al 31 Agosto 2018, su 288 host collezionati, Virus Total ha contrassegnato come non malevoli 51 host.

1.6.3 eval

Per lanciare le statistiche relative agli eval, è necessario lanciare lo script *evalStata*.

Dal 13 al 31 Agosto 2018, con 288 host collezionati, sono stati prodotti 4494 file javascript. Di questi 5 presentano eval espliciti e 4489 contengono eval impliciti. La statistica degli eval per singoli host si può trovare in 2.1.

1.7 Schema del progetto



Capitolo 2

Appendice

2.1 Tabella eval per host

Host	Eval espliciti	Eval impliciti
www.j610033.myjino.ru	0	3
www.veryaffordableblinds.com	0	3
sigtap.datasus.gov.br	0	4
rahulacollegeoba.lk	0	98
xhcrab.com	0	54
www.buyisraeli.co.il	0	45
pharezconsulting.com	0	2
www.mgprogramming.net	0	15
j.kyryl.ru	0	2
www.pset.ltd	0	2
www.tabanway.com.tr	0	227
steamdesktopauthenticator.com	0	2
www.gruzolub.ru	0	202
www.valoneew.beget.tech	0	3
pureleisurefun.com	0	121
plus.google.com	3	0
www.youtube.com	0	102
omexturviajes.com	0	22
twitter.com	0	407
www.hugedomains.com	0	42
www.smithmillie21.000webhostapp.com	0	2
hsa.ddns.net:8181	0	2
drclaudiadiez.com	0	1
nailedglamour.com	0	2
kaymanlimited.com	0	8
prijzen-dakkapel.nl	0	1

ajax.googleapis.com	0	1
infrarotgrill-test.info	0	3
wxklsb.com	0	57
dev2.mywebproof.net	0	9
zionsifac.com	0	23
cointradingsoftware.com	0	1
demo.proteusthemes.com	0	20
koenig-kebap.de	0	6
www2.datasus.gov.br	1	0
cosmos.cmitik.ru	0	5
mail.thuoht.website	0	3
www.rntk-imperia.ru	0	31
sites.google.com	0	5
mister-clean.pro	0	2
www.unimedmissoes.com.br	0	1
www.000webhost.com	0	169
www.sscgd.win	0	1
yarbisalama.hopto.org	0	1
hospitalsantoangelo.com	0	198
mypointapp.com	0	1
mac10mincomercio.000webhostapp.com	0	76
jaju.ltd	0	3
www.xhcrab.com	0	2
yawzee.me	0	4
www.collateralproduccions.com	0	1
www.st.is	0	10
kiyanka.club	0	3
www.clinicasaoangelo.com.br	0	3
piezodoorphone.com	0	29
windows10portal.com	0	47
www.ysd63.com	0	1450
www.smtechsrnc.com	0	1
br.wordpress.org	0	14
rahulacollege.org	0	27
www.secured-production.000webhostapp.com	0	2
www.cmitik.ru	0	4
centralappdownloadtrials.com	0	5
www.oxbourn.com	0	7
www.wxklsb.com	0	2
octopuspackaging.com	0	125
www.taverna-stuttgart.de	0	28

ideaintl.net	0	8
kassconnect.ru	0	77
www.zionsifac.com	0	1
ip01reg.myjino.ru	0	3
logs.icu	0	1
www.ipe.rs.gov.br	0	3
www.tulsamassageboutique.com	0	1
www.initex.com	0	1
houswe.com	0	223
www.google.com	0	6
cartaonet.datasus.gov.br	0	1
www.milfcamsluts.com	0	1
istanbuliklimlendirme.net	0	1
imagic-box.com	0	220
www.alvaro.com.br	0	34
kd.mestving.co.uk	0	1
www.cjoint.com	0	7
www.brcsari.ir	0	1
pratimspizza.com	0	145
www.nathaliedodon.com	1	2

Bibliografia

[1] https://github.com/erroneousboat/warc3. Warc extractor.

[2] https://github.com/internetarchive/heritrix3. Heritrix.

[3] https://safebrowsing.google.com. Google safe browsing.

[4] https://www.virustotal.com. Virus total.