

UNIVERSITÀ DEGLI STUDI DEL MOLISE
DIPARTIMENTO DI BIOSCIENZE E TERRITORIO

Project work inerente al corso di
SICUREZZA DELLE RETI E DEI SISTEMI SOFTWARE

**Crawling and Vulnerability
Assessment**

Studente

Francesco di Rito
Matricola 167900

Studente

Emanuela Guglielmi
Matricola 167901

Anno Accademico 2019/2020

Indice

1	Introduzione	4
1.1	Contesto applicativo	4
1.2	Motivazioni e obiettivi	6
1.3	Organizzazione del lavoro	7
2	Stato dell'arte	8
2.1	Web Crawler	8
2.2	Funzionamento del Web Crawler	9
2.3	Policy di funzionamento	13
2.4	File robots.txt	14
2.5	Web crawler e gli errori di indicizzazione	15
2.6	Scraping dei dati	16
2.7	Differenza tra Scraping e Crawling dei dati	16
2.8	Crawling, Scraping e legalità	17
2.9	Cross-Site Scripting	18
2.9.1	DOM-based XSS	18
2.10	Analisi delle vulnerabilità	19
2.11	Burp Suite	22
2.12	Dynamic taint analysis	22
3	Metodologie	25
4	Realizzazione del progetto	28
4.1	Realizzazione del crawler	28
4.1.1	Robots.txt	30

4.1.2	Database	31
4.2	Analisi con Burp Suite	32
4.2.1	Configurazione di Burp Suite	32
4.2.2	Scansione vulnerabilità del sito Web	36
5	Manuale Utente	45
5.1	Splash Screen	45
5.2	Welcome Panel	46
5.3	Schermata <i>Crawling</i>	46
5.4	Status bar	49
5.5	Schermata <i>Seeds</i>	51
5.6	Schermata <i>History</i>	52
5.7	Console	53
6	Studio vulnerabilità	55
6.1	Alexa Rank	55
6.2	Crawling	56
6.2.1	Attività di analisi	66
7	Conclusione e sviluppi futuri	73

Capitolo 1

Introduzione

1.1 Contesto applicativo

Dalla nascita del *World Wide Web* (6 Agosto 1991) fino ad oggi, la stima delle pagine che compongono questo immenso mondo astratto risale a circa 10 miliardi. Non è difficile rendersi conto di quanto potrebbe risultare complesso effettuare una ricerca all'interno di questo numero smisurato di informazioni, senza disporre di strumenti adatti. La mente umana naturalmente non sarebbe in grado di gestire tali quantità, pertanto c'è bisogno di algoritmi che siano in grado di analizzare e archiviare i dati presenti nel web, mettendo in ordine gli stessi in modo da renderli velocemente accessibili e conoscibili. Generalmente, a svolgere questo lavoro sono i motori di ricerca, ovvero sistemi automatici che sulla base di parole o frasi prese in input (generalmente definite *query*), analizzano il web con l'obiettivo di restituire una lista di siti contenenti l'input specificato. L'attività di un motore di ricerca consiste principalmente in diversi processi [1]:

- **crawling**, in generale, riguarda la scansione del web al fine di rilevare informazioni dalle pagine analizzate;
- **indexing**, termine che indica l'indicizzazione dei siti. Dopo la fase di crawling, tutte le informazioni che sono state recuperate, devono in qualche modo essere ordinate. Pertanto, mediante l'utilizzo di determinati algoritmi, il motore di ricerca classifica le pagine analizzandone

parole chiave, categorie, tematiche e ulteriori parametri e sulla base di questi ultimi sarà attuato il processo di indicizzazione. Nel momento in cui un utente avvia una ricerca, il motore non andrà a consultare l'intero web, bensì andrà a leggere il proprio database contenente i dati già perfettamente ordinati. Grazie a questo meccanismo è possibile avere in tempi rapidi la pagina di risposta e i relativi contenuti ricercati;

- **ranking**, cioè la classificazione dei siti per importanza o rilevanza. Nel momento in cui un utente inserisce la query di ricerca, il motore attraverso evoluti e raffinati algoritmi preleva dagli indici i documenti più vicini semanticamente alla query, ordinandoli poi in una sorta di classifica. La maggior parte dei motori di ricerca, per decidere questo ranking, prendono in considerazione più di 200 fattori, tra questi ad esempio, le parole più frequentemente ricercate all'interno della pagina, oppure la presenza o meno delle parole che formano la query nell'URL, nel meta-tag title, nel titolo e nelle prime righe del testo o la qualità del sito e la sua importanza, stabilita dal motore di ricerca attraverso un'analisi semantica;
- **creazione della SERP (Search Engine Results Page)**, la pagina dei risultati del motore di ricerca o, in altre parole, l'elenco ordinato visualizzato da un utente a seguito di una ricerca. Questa è stilata in modo decrescente dal contenuto migliore a quello peggiore.

Grazie a tutta questa serie di attività, ogni giorno milioni di utenti sono in grado di effettuare le proprie ricerche sul web ed in una frazione di secondo ottengono risposte contenenti i risultati desiderati.

1.2 Motivazioni e obiettivi

L'importanza dei motori di ricerca e del loro funzionamento, ci ha indotti a prendere in considerazione il processo mediante il quale essi operano sul web ed in particolare una fase molto interessante, ossia il *crawling*. Difatti, nel presente lavoro ci concentreremo particolarmente su quest'ultima, definendo in dettaglio quali sono gli obiettivi dei cosiddetti crawlers e le modalità di operazione utilizzate da questi software. Ci caleremo in un contesto di sicurezza al fine di analizzare delle vulnerabilità dei siti web. Infatti, nonostante il web costituisca una risorsa fondamentale nel mondo odierno nella vita di molte persone, rappresenta anche un contesto di molteplici attacchi. I nostri dati e un po' tutte le informazioni che ci caratterizzano, non sono al sicuro sul web, in quanto la sicurezza delle applicazioni non è mai garantita, malgrado l'utilizzo di elevate misure di protezione. Pertanto, l'obiettivo principale di tale lavoro riguarderà l'analisi del web al fine di rilevare possibili vulnerabilità presenti nelle pagine che andremo a processare. Nel dettaglio, le vulnerabilità su cui ci concentreremo principalmente, sono quelle che rendono possibili attacchi di tipo *DOM-based XSS*. Di conseguenza, divideremo il nostro lavoro in due fasi. La prima fase riguarderà l'implementazione di un crawler, al quale daremo in input dei link da analizzare e sulla base di questi, l'obiettivo del nostro modulo sarà quello di percorrere i collegamenti ipertestuali che si trova davanti restituendo in output ulteriori link, i quali saranno strutturati ed immagazzinati in un database. I link da cui partiremo per effettuare questa analisi saranno i siti di Alexa (<https://www.alexa.com/topsites/countries/IT>), tra i più popolari sul web a livello globale. Pertanto, a partire da questi siti, attraverso il nostro crawler raccoglieremo tutti i collegamenti ad altre pagine web impostando un limite al numero di link da analizzare per limitare la nostra ricerca ad una dimensione gestibile degli stessi, dati i nostri vincoli di risorse. Successivamente, nella seconda fase, utilizzeremo gli url ottenuti dal crawler come input da dare in pasto ad un tool al fine di rilevare eventuali vulnerabilità *DOM-based XSS*. Ci sono diverse metodologie riguardanti il rilevamento di tali vulnerabilità su Internet. Alcune di queste sono statiche, altre dinamiche. Come vedremo nei

paragrafi successivi, esistono alcuni tool automatici in grado di rilevare tali vulnerabilità, come ad esempio *Burp Suite*, i quali svolgono le proprie attività basandosi su approcci statici o dinamici, a tal fine andremo a valutare l'efficacia dei diversi approcci sulla base dei risultati ottenuti. Inoltre, valuteremo di implementare un nostro tool di rilevamento di tali vulnerabilità, dopo aver preso atto della metodologia più adatta ad effettuare tali rilevamenti.

1.3 Organizzazione del lavoro

Nel capitolo in questione abbiamo introdotto brevemente alcuni concetti che andranno a caratterizzare lo svolgimento di questo lavoro definendo inoltre quali saranno gli obiettivi. Nei successivi capitoli vi è la descrizione delle metodologie da utilizzare ed infine, le varie fasi del lavoro svolto.

- **Capitolo 2:** dove vi è lo stato dell'arte contenente informazioni relative alla definizione e al funzionamento del *crawling*, alla variante dell'attacco *Cross-site Scripting* denominato *DOM-based XSS* e un'introduzione al software *Burp Suite*.
- **Capitolo 3:** all'interno del quale vengono illustrate le metodologie prese in considerazione per l'effettiva realizzazione del lavoro.
- **Capitolo 4:** tale sezione è dedicata alle tecniche di implementazione utilizzate per lo sviluppo del crawler e le procedure da utilizzare al fine di effettuare un'analisi mediante il tool *Burp Suite*.
- **Capitolo 5:** vi è riportato il Manuale Utente inerente al crawler realizzato, attraverso un'anteprima delle varie schermate del tool.
- **Capitolo 6:** sono riportate in dettaglio le varie fasi che caratterizzano il lavoro svolto illustrando inoltre uno studio di vulnerabilità. Il tutto mediante appositi *screenshot* i quali fungono da guida alle varie attività.
- **Capitolo 7:** tale sezione è dedicata alla conclusione del lavoro svolto.

Capitolo 2

Stato dell'arte

2.1 Web Crawler

La prima fase svolta da un motore di ricerca durante la sua attività è la scansione del web, meglio conosciuta come *crawling*. Con questo termine viene definita quell'attività mediante la quale si vanno ad analizzare, in maniera metodica e automatizzata, i contenuti di una rete, in genere per conto di un motore di ricerca. Tale attività è normalmente svolta da un cosiddetto crawler definito anche web crawler, spider o robot. I robot sono automatizzati, motivo per il quale funzionano seguendo le loro istruzioni senza che un utente umano debba avviarli. I robot di solito operano su una rete, più della metà del traffico Internet è costituito da bot che scansionano contenuti e interagiscono con pagine web. Un crawler è un tipo di bot (abbreviazione di robot), il cui obiettivo consiste nell'attraversare le pagine web “seguendo” link ipertestuali e al contempo raccolgono informazioni sui contenuti delle pagine con lo scopo di indicizzare queste ultime in modo opportuno nel database principale del motore di ricerca [2]. L’obiettivo di un bot di questo tipo è imparare di cosa tratta (quasi) ogni pagina sul web, in modo che le informazioni possano essere recuperate quando è necessario. I bot più diffusi utilizzati dai motori di ricerca più importanti sono:

- Googlebot, utilizzato da Google, il quale attualmente utilizza un crawler per le ricerche Desktop e uno per quelle Mobile;

- Bingbot, utilizzato da Bing;
- Slurp Bot, di Yahoo;
- Romilda, di Facebook;

Come si evince, i crawlers sono un elemento fondamentale nei motori di ricerca. Applicando un algoritmo di ricerca ai dati raccolti dai crawler web, i motori di ricerca possono fornire collegamenti pertinenti in risposta alle query dell'utente, generando l'elenco di pagine web che vengono visualizzate dopo che un utente digita del testo su Google o Bing (o un altro motore di ricerca). Un bot del web crawler è come qualcuno che sfoglia tutti i libri di una biblioteca disorganizzata e mette insieme un catalogo di carte in modo che chiunque visiti la biblioteca possa trovare rapidamente e facilmente le informazioni di cui ha bisogno. Per aiutare a classificare e ordinare i libri della biblioteca per argomento, l'organizzatore leggerà il titolo, il riassunto e parte del testo interno di ciascun libro per capire di cosa si tratta. Tuttavia, a differenza di una biblioteca, Internet non è composto da pile di libri fisici e ciò rende difficile dire se tutte le informazioni necessarie sono state indicizzate correttamente o se ne vengono trascurate grandi quantità. La maggior parte dei crawler web non esegue la scansione di tutte le risorse disponibili online ma decide quali scansionare in base al numero di altre pagine collegate, alla quantità di visitatori e ad altri fattori che indicano la probabilità che la pagina contenga informazioni importanti e pertinenti [3]. Non è noto quanta parte di Internet disponibile pubblicamente sia effettivamente sottoposta a scansione dai robot dei motori di ricerca. Alcune fonti stimano che solo il 40-70% di Internet sia indicizzato per la ricerca, ovvero miliardi di pagine web.

2.2 Funzionamento del Web Crawler

Come anticipato, l'obiettivo principale del crawler è quello di scansionare la rete. Al fine di eseguire tale attività di scansione, bisogna definire alcuni insiemi utili al funzionamento:

un primo insieme è composto dai cosiddetti *seeds*, ovvero url di partenza che verranno dati in input al programma. Dunque, in primis si esegue la

scansione delle pagine web in corrispondenza di tali url e successivamente, durante la ricerca per indicizzazione di tali pagine, si troveranno collegamenti ipertestuali ad altri url e si aggiungeranno all'elenco di pagine da sottoporre a scansione. La scelta dei *seeds* è molto importante poiché da questi dipende il risultato di tutto il *crawling*, ciò in quanto una scelta non opportuna potrebbe portare ad analizzare un numero di pagine molto ristretto, poiché non tutte contengono link ipertestuali. Oppure, considerato il vasto numero di pagine web su Internet che potrebbero essere indicizzate per la ricerca, questo processo potrebbe continuare quasi all'infinito. Tuttavia, un crawler web seguirà alcune politiche che lo rendono più selettivo su quali pagine eseguire la scansione, in quale ordine e con quale frequenza dovrebbe eseguire nuovamente la scansione per verificare la presenza di aggiornamenti del contenuto. A tal fine, un aspetto fondamentale è l'importanza relativa ad ogni pagina web. La maggior parte dei crawler non esegue la scansione dell'intera Internet disponibile al pubblico; decidono invece quali pagine scansionare per prime in base al numero di altre pagine che si collegano a quella pagina, alla quantità di visitatori che la pagina ottiene e ad altri fattori che indicano la probabilità della pagina di contenere informazioni importanti. Difatti, l'idea è che una pagina web citata da molte altre pagine e che attira molti visitatori, conterrà probabilmente informazioni autorevoli di alta qualità, quindi è particolarmente importante che un motore di ricerca le abbia indicizzate.

Un secondo insieme, molto importante, è quello definito *crawl frontier*, da cui verranno prelevati i link su cui effettuare l'analisi, inseriti i nuovi link trovati ed infine rimossi quelli su cui l'analisi è avvenuta. Da ciò si deduce l'esistenza di un ulteriore set di link che identifica tutti gli url già analizzati, i quali sono stati rimossi dalla *frontiera*. La gestione di questi insiemi avviene mediante un database il quale andrà ad immagazzinare tutti i link. L'operazione primaria svolta da un crawler è quella di recuperare un link dalla *frontiera*, il cosiddetto *seed*. Nel momento in cui quest'ultimo viene processato, verrà eliminato dalla *frontiera* ed inserito nel set riguardante i link *visitati*. Verrà scaricato il codice *HTML* della pagina del link in questione al fine di analizzarne i relativi contenuti, secondo gli scopi del crawler, ed estrarre tutti i link della pagina. Di questi ultimi, quelli non presenti né nella *frontiera* né nel

set dei link *visitati*, verranno aggiunti alla *frontiera* pronti per essere processati in seguito. Naturalmente, se sono presenti ancora link da analizzare, il crawler provvederà a processarli, altrimenti il processo termina [4]. Per tale motivo, l'esecuzione del programma può terminare in automatico dopo che sono stati analizzati tutti i link nella *frontiera*, altrimenti può essere concluso manualmente nel momento in cui non è stato definito un limite negli url da analizzare. Di seguito è riportato l'algoritmo di funzionamento di un crawler il quale rappresenta le principali attività eseguite.

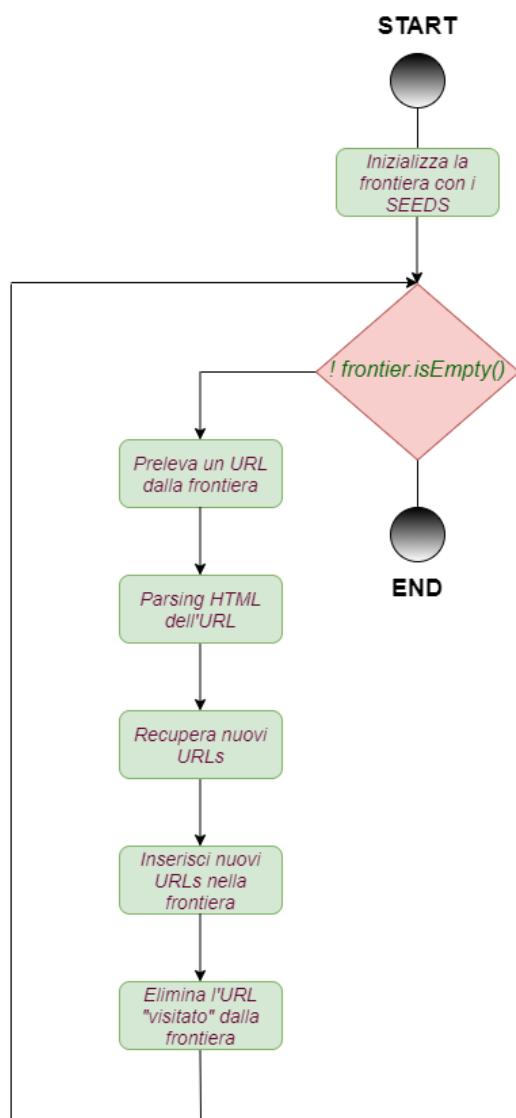


Figura 2.1: Crawling: algoritmo

2.3 Policy di funzionamento

Nel paragrafo in questione analizzeremo alcuni concetti che regolarizzano le operazioni eseguite da un crawler: le cosiddette *policy*. In particolare ne esistono 4 e prendono il nome di:

- **Selection policy (Selezione):** come è noto, il web raggiunge dimensioni enormi, pertanto è praticamente impossibile riuscire ad indicizzare tutti i siti presenti. La *policy di selezione* garantisce quindi una priorità per ogni pagina web, di conseguenza si andranno a scansionare solamente i siti di maggiore importanza e che quindi siano maggiormente aggiornati, maggiormente visitati e con contenuti qualitativamente più elevati.
- **Re-visit policy (Gestione delle visite):** i tempi per effettuare una mappatura completa di una data porzione del web da parte di un crawler, potrebbero essere equivalenti a settimane o mesi. Poichè la rete è dinamica, le pagine che la compongono possono cambiare in contenuti e quant'altro. Per tale motivo viene stabilita una *policy di gestione delle visite* per assicurarsi che i contenuti salvati nel database vengano aggiornati periodicamente.
- **Politeness policy (Cortesia):** le performance di un sito web possono subire un forte impatto nel corso della scansione di un crawler. Attraverso la *policy di cortesia*, vengono stabiliti intervalli, limiti per le risorse utilizzate e così via, per garantire un corretto funzionamento. Una soluzione parziale a questo problema è l'utilizzo di un file *robots.txt*.
- **Parallelization policy (Parallelizzazione):** per evitare problemi di url ridondanti nella lista dei *seeds* o nella lista dei *crawl frontier*, viene adottato un sistema di parallelizzazione (multi thread) dei bot, che effettueranno più scansioni contemporaneamente in una sola volta, anziché scansioni complete eseguite più volte [5].

2.4 File robots.txt

Come anticipato nel paragrafo precedente, una delle soluzioni utilizzate al fine di garantire la cosiddetta *policy di cortesia*, è quella di utilizzare un semplice file di testo, denominato *robots.txt*. Quest'ultimo va ad indicare agli spider dei motori di ricerca alcune semplici direttive circa l'accesso a determinati file o intere cartelle. Pertanto, prima che un crawler inizi ad operare, è necessario che legga il contenuto di questo file. Ogni sito web può avere un solo *robots.txt*, il quale deve essere un file di testo con caratteri *UTF-8* necessariamente denominato col nome in questione e deve essere ospitato sul server web proprio come qualsiasi altro file sul sito. Al fine di essere individuato facilmente, dovrà essere contenuto nella cartella di root del sito web in questione. In effetti, questo file può in genere essere visualizzato digitando l'url completo per la *home page* di un particolare sito e quindi aggiungendo **/robots.txt**, come <https://www.google.it/robots.txt>. I file *robots.txt* utilizzano una serie di protocolli differenti, quello principale è *Robot Exclusion Protocol*. Quest'ultimo consente di dire ai bot quali pagine web e risorse evitare. L'altro protocollo utilizzato è *Sitemap*, il quale può essere considerato un protocollo di inclusione dei robot. Le *Sitemaps* mostrano al web crawler quali pagine possono scansionare, ciò aiuta a garantire che un bot del crawler non perda nessuna pagina importante. Grazie al file *robots.txt*, si potrebbe chiedere ad uno specifico spider di non scansionare determinate cartelle (ad esempio cartelle con file riservati che non vogliamo siano raggiungibili) oppure addirittura precludere l'accesso all'intero sito ad uno specifico crawler. Nello specifico, un file *robots.txt* è formato da uno o più gruppi che come detto, bloccano (o consentono) l'accesso di un determinato crawler a un percorso di file specificato nel sito web in questione. Di seguito riportiamo un esempio di file *robots.txt* con 2 regole [6]:

```

# Group 1
User-agent: Googlebot
Disallow: /nogooglebot/

# Group 2
User-agent: *
Allow: /

Sitemap: http://www.example.com/sitemap.xml

```

Figura 2.2: File robots.txt

Nella prima regola, lo *User-agent* *Googlebot*, ovvero il crawler utilizzato da Google, non deve eseguire la scansione della cartella *http://example.com/nogooglebot/* o di eventuali sottocartelle. Nella seconda regola è indicato che tutti gli altri *User-agents* possono accedere all'intero sito. Naturalmente questa regola si potrebbe omettere perché il presupposto è l'accesso completo. Infine, il file *Sitemap* del sito si trova all'indirizzo *http://www.example.com/sitemap.xml*.

2.5 Web crawler e gli errori di indicizzazione

È possibile che errori riguardanti file o l'architettura del sito condizionino il risultato dell'indicizzazione. Per fare in modo che gli spider possano accedere agevolmente a tutti i contenuti del sito, è necessario verificare la *crawlability*, cioè la capacità di consentire ai crawler di consultare i contenuti. Ecco gli errori più comuni [7]:

- **Indicizzazione bloccata dal file robots.txt:** la *Search Console* di Google permette di verificare se il file *robots.txt*, il primo che viene analizzato dai web crawler, non permette la consultazione dei contenuti da parte dei bot dei motori di ricerca.
- **Meta tag robots:** la presenza di una stringa di questo tipo *<meta name="robots" content="noindex"/>* nel codice del sito.
- **Errori nella Sitemap:** il file *Sitemap* è la guida che conduce i web crawler nell' indicizzazione dei contenuti. Se è errato può condizionare fortemente questa operazione. Se è mancante, è bene inserirlo nel sito.

- **Problemi nel server:** un server sovraccarico, o che restituisce errori *5xx*, deve essere verificato segnalando il problema all’hosting del sito.

2.6 Scraping dei dati

Un ulteriore aspetto fondamentale riguarda lo scraping dei dati. Gli attaccanti possono utilizzare gli strumenti di web scraping per accedere ai dati molto più rapidamente del previsto, il chè può comportare l’utilizzo di dati per scopi non autorizzati. Lo scraping dei dati consiste in una tecnica in cui un programma per computer estraie i dati dall’output generato da un altro programma. Lo scraping dei dati si manifesta comunemente nello scraping del web, il processo di utilizzo di un’applicazione per estrarre informazioni da un sito web. Il processo di web scraping risulta semplice, sebbene l’implementazione possa essere complessa. Lo scraping del web avviene mediante tre passaggi:

1. Il robot scraper, ossia il pezzo di codice utilizzato per estrarre le informazioni, invia una richiesta *HTTP GET* a un sito web specifico.
2. Nel momento in cui il sito web risponde, lo scraper analizza il documento *HTML* per uno specifico modello di dati.
3. Una volta estratti, i dati vengono convertiti in qualsiasi formato specifico progettato dall’autore del robot scraper.

2.7 Differenza tra Scraping e Crawling dei dati

Il *crawling* si riferisce al processo che i grandi motori di ricerca come Google intraprendono quando inviano i loro crawler robot, come Googlebot, nella rete per indicizzare i contenuti Internet. Lo scraping invece, è strutturato in modo specifico per estrarre dati da un determinato sito web. Differenze tra un robot scraper e un bot crawler web:

1. I robot scraper faranno finta di essere browser web, mentre un bot crawler indicherà il suo scopo e non tenterà di indurre un sito web a pensare che sia qualcosa che in realtà non è.
2. Lo scraper interpreta azioni avanzate come la compilazione di moduli o altri componenti per raggiungere una determinata parte del sito web, il crawler no.
3. Lo scraper generalmente non tiene conto del file *robots.txt*, in quanto progettato per estrarre il contenuto esplicitamente contrassegnato per essere ignorato.

2.8 Crawling, Scraping e legalità

Al giorno d'oggi, è davvero difficile determinare la legalità del web scraping. La pratica del web scraping non è di per sé illegale, in realtà essa varca la soglia dell'illegalità quando viene impiegata per finalità illecite e quando i dati estratti vengono utilizzati per altri usi all'insaputa e senza il consenso del titolare del sito e/o del titolare dei contenuti e delle informazioni presenti su di esso. Un modo per limitare questa pratica, da utilizzare in particolare per proteggere quei contenuti del sito che si vuole non vengano estratti, è quello di creare aree riservate il cui accesso è consentito solo previa registrazione. Il web scraping può configurare diverse violazioni, come la violazione del diritto d'autore, di diritti di proprietà industriale, violazioni sulla privacy, violazioni di condizioni contrattuali (ad esempio, i termini d'uso del sito), appropriazione indebita e, in alcuni casi, anche il reato di accesso abusivo ad un sistema informatico o sito web. Il web crawling può essere utilizzato a scopo dannoso, ad esempio:

1. Raccolta di informazioni private o classificate.
2. Ignorare i termini e il servizio del sito web, fare scraping senza il permesso dei proprietari.
3. Un modo abusivo di richieste di dati porterebbe ad arresti anomali del server web sotto carico eccessivo.

Fare scraping legalmente è comunque possibile. Difatti, un esempio molto famoso è quello della celebre *Import.io*, società che nel mondo si occupa di

scraping per poi generare comparazione di prezzi, analisi di mercato e di prodotto e tanti altri servizi di svariato tipo [8].

2.9 Cross-Site Scripting

Il *cross-site scripting (XSS)* è una vulnerabilità a cui sono esposti i siti web dinamici che impiegano un insufficiente controllo dell'input nei form.

Un *XSS* permette ad un attaccante di inserire o eseguire codice lato client al fine di attuare un insieme variegato di attacchi quali ad esempio, raccolta, manipolazione e reindirizzamento di informazioni riservate, visualizzazione e modifica di dati presenti sui server, alterazione del comportamento dinamico delle pagine web, ecc [9]. Tale attacco è possibile grazie all'utilizzo di diversi linguaggi di scripting lato client, il più utilizzato è *JavaScript*.

L'effetto dell'attacco può comportare piccoli fastidi alla vittima oppure rischi significativi per la sicurezza di informazioni sensibili.

Un utente malintenzionato può iniettare *markup* arbitrario utilizzando *document.location* come vettore di attacco creando un collegamento che inietta uno script controllato da un utente malintenzionato nella pagina.

Esempio di vulnerabilità *DOM-based XSS*:

```
document.write('<a href=' + document.location + '>Link</a>');
```

2.9.1 DOM-based XSS

Una delle varianti più diffuse del *cross-site scripting* è l'attacco *DOM-based XSS*. Prima di introdurre le modalità dell'attacco, bisogna specificare che il *DOM*, acronimo di *Document Object Model*, è costruito come un albero di oggetti, dove la root è un documento e tutti i nodi sottostanti sono gli elementi del documento. Nel momento in cui un utente accede ad una pagina web, il browser costruisce il *DOM*. Se la pagina in questione è vulnerabile, il browser interpreta il codice dannoso inserito da un utente malintenzionato durante la costruzione del *DOM* [10]. Quindi il payload viene iniettato direttamente nel *JavaScript* della pagina ed è eseguito pertanto al livello del *DOM*. In questo modo, se nella pagina vulnerabile è presente uno script che si interfaccia agli

elementi della stessa, come ad esempio campi form, cookie o url e se il valore di questi ultimi è controllabile da un attaccante, allora è possibile l'iniezione e l'esecuzione del payload [11]. Questo tipo di attacco viene chiamato anche *XSS locale* in quanto va a colpire una vulnerabilità lato client ed il server web non prende parte a questo processo.

Esempi di fonti potenzialmente controllate da aggressori includono: l'url del documento, a cui si accede tramite l'oggetto *JavaScript document.location*; dati passati in messaggi di origine incrociata utilizzando l'API postMessage; cookie, accessibili tramite l'oggetto *document.cookie*; il *referrer HTTP* a cui accede l'oggetto *document.referrer JavaScript* e altri metodi.

2.10 Analisi delle vulnerabilità

Spesso le reti informatiche presentano delle vulnerabilità, hardware e software, che opportunamente sfruttate, possono portare a un furto dati, al danneggiamento di un servizio o ad altre gravi disfunzioni. L'attività di *vulnerability assessment* (analisi di vulnerabilità), effettuata mediante tool specifici in grado di eseguire una lunga serie di controlli su ogni singolo dispositivo o applicazione, permette di ottenere una fotografia dello stato di esposizione dei propri sistemi a tutte le vulnerabilità note e di poter intervenire su di essi adottando tutti gli strumenti più adeguati a prevenire eventuali danni. Le vulnerabilità della sicurezza possono originarsi in un prodotto non appena siano state scritte le prime righe di codice, e il vero pericolo è che non vengano rilevate se non quando è troppo tardi. Pertanto, è necessario utilizzare strumenti in grado di rilevare possibili vulnerabilità già durante la scrittura del codice. Al fine di poter effettuare l'analisi di vulnerabilità possiamo adottare tool di analisi statica e/o dinamica. L'adozione di un tool di analisi statica è una buona scelta in quanto, analizza sintassi, semantica, stima delle variabili, nonché controllo e flusso dati per identificare problemi legati al codice. L'analisi dinamica del programma è diversa, dato che è attuata mediante l'esecuzione di programmi su una architettura del processore vera e propria o simulata. Secondo quanto emerso da uno studio delle vulnerabilità *DOM-XSS* [12], molti metodi sono stati usati per mitigare o

difendere dalle vulnerabilità *XSS*, ma non si applicano a *DOM-XSS*. Analisi statica e dinamica trovano tipi di vulnerabilità qualitativamente diversi, potrebbero pertanto essere necessari test con approcci sia statici che dinamici, ciò al fine di proteggere le applicazioni web da *DOM-XSS*. Gli strumenti di analisi statica che supportano *JavaScript*, sono ad esempio *ScanJS* e *JSLint*, tali strumenti staticamente, senza codice executing, tentano di rilevare errori di programmazione comuni in *JavaScript*, ad esempio, sottolineando l'uso di funzioni pericolose o variabili indefinite. Gli strumenti di analisi statica stanno diventando una parte generalmente accettata del modo in cui viene sviluppato il software: passare un'analisi statica senza errori è un requisito spesso elencato nelle guide di stile. In parte, ciò è dovuto alla loro capacità di essere eseguiti in modo pratico e ripetibile con poca configurazione, ad esempio nelle build notturne. Alcuni strumenti di rilevamento delle vulnerabilità, come *Burp Suite*, hanno anche un componente di analisi statica. Tuttavia, non è noto fino a che punto gli strumenti di analisi statica possano rilevare e prevenire le vulnerabilità reali di *DOM-XSS*. Vi sono differenti strumenti per rilevare tali vulnerabilità, tra cui *ScanJS*, *Esflow* e gli strumenti di analisi statica in *Burp Suite Pro*.

- **ScanJS** è uno strumento pensato per aiutare le persone a evitare pratiche di codifica che portano, tra le altre cose, a vulnerabilità *DOM-XSS*. Come tale, contrassegna il codice che potrebbe essere pericoloso senza mirare a identificare se il codice porta a un bug sfruttabile. Ad esempio, può indicare tutte le posizioni in cui la funzione *document.write* è stata utilizzata con una stringa non statica come argomento. Sebbene sia una buona pratica da evitare, non è sempre indicativo di una vulnerabilità. In effetti, la maggior parte dei casi è benigna.
- **Esflow** è unico in quanto spesso allega informazioni di sink ai suoi rapporti sui problemi per facilitare il debug.
- **Burp Suite** attribuisce un punteggio di confidenza a ogni potenziale vulnerabilità che segnala, fornendo indicazioni su quali risultati siano più affidabili. *Burp Suite* riceve anche il codice dal sito web fungendo

da proxy tra il browser e il sito stesso, il che significa che ha accesso al codice che viene caricato dinamicamente (ad esempio da un tag `<script>` aggiunto durante l'esecuzione) a differenza degli altri strumenti; tuttavia, non è ancora in grado di analizzare il codice generato dinamicamente, ad esempio, utilizzando la funzione `eval()`.

Per indirizzare *JavaScript*, l'analisi dinamica tradizionalmente è vista come avere meno falsi positivi; tuttavia, l'analisi statica è spesso più utile per i programmatore durante lo sviluppo a causa della mancanza di analisi personalizzate per l'aggiunta di nuovo codice. Inoltre, l'analisi statica può essere più completa, in grado di rilevare le vulnerabilità nel codice che non è stato eseguito su una determinata esecuzione del programma. Tuttavia, un vincolo degli strumenti di analisi statica che limita particolarmente in *JavaScript* è l'incapacità di analizzare il codice generato dinamicamente. Un modo per prevenire le vulnerabilità di *DOM-based XSS* è rilevarle prima che il software venga rilasciato. Una direzione promettente per la ricerca su larga scala delle vulnerabilità *DOM-XSS* è l'utilizzo di tecniche che analizzano porzioni più ampie dello spazio del programma. Il problema della copertura del codice delle tecniche di analisi dinamica non è nuovo o specifico per le vulnerabilità *DOM-XSS*; tuttavia, può essere un ostacolo maggiore per l'analisi su larga scala delle applicazioni web rispetto ai programmi tradizionali.

2.11 Burp Suite

Burp Suite è un framework basato su *java*, progettato per analizzare il traffico di rete generato da applicazioni web-based. Utilizzato per esaminare e manipolare il flusso di dati dal browser web verso il server di riferimento, come ad esempio un sito Internet, è strutturato in una serie di strumenti che consentono all'utente di eseguire diversi *Penetration Test* e attacchi di vario genere. *Burp Suite* agisce quindi come una sorta di *Man In The Middle* acquisendo e analizzando ogni richiesta da e verso l'applicazione web di destinazione. I *Penetration test* possono mettere in pausa, manipolare e riprodurre singole richieste *HTTP* al fine di analizzare potenziali parametri o punti di iniezione. Naturalmente, il primo passo da compiere per utilizzare questo software, riguarda la sua configurazione. In altre parole, è necessario configurare il browser che si desidera utilizzare, in modo tale che tutte le richieste ad un determinato sito passino attraverso *Burp* [13]. A questo punto, è possibile sfruttare le componenti principali di *Burp*, le quali riguardano: il **Proxy**, che permette di ispezionare e manipolare il traffico di rete tra il browser web e l'applicazione da analizzare; lo **Spider**, che indicizza il contenuto del sito, generando una lista di url da verificare per individuare eventuali falle di sicurezza; lo **Scanner**, il quale esegue uno scan automatizzato in cerca di vulnerabilità note; l'**Intruder**, utile per generare attacchi personalizzati; il **Repeater**, il quale registra ogni richiesta eseguita tra client e server ed il **Sequencer**, che permette di verificare la “casualità” dei dati. Sono disponibili due versioni di *Burp*: la *Free Edition* e la *Professional*, la quale offre strumenti aggiuntivi [14].

2.12 Dynamic taint analysis

Un'analisi dinamica effettuata da un gruppo di ricercatori Americani, utilizza questo approccio dinamico per rilevare le vulnerabilità *DOM-based XSS* su Internet. Andremo ad analizzare il loro studio al fine di poter valutare la tipologia di analisi da attuare. Tale ricerca è stata effettuata mediante il *crawling* di 10.000 siti web *Alexa Top*, utilizzando il browser *Chromium*.

da loro integrato con modifiche che consentono tale analisi, rispettando le direttive *robots.txt*.

La *Dynamic taint analysis* è una metodologia che prevede l'allocazione dello spazio in ogni primitiva di stringa *JavaScript* che memorizza la provenienza di ciascun byte della stringa. Ciò consente alle *taint information* di essere propagate con precisione durante la concatenazione delle stringhe. Oltre alla provenienza di ogni byte di stringa, ogni byte di contabilità registra anche quali metodi di codifica integrati sono stati applicati.

Il browser verifica gli argomenti delle *sensitive functions* (ad es. funzione *eval* o *document.write*) per byte contaminati.

Se un argomento contiene byte contaminati, viene scritto un record in un file di registro che descrive il flusso, tra cui: il tipo di contaminazione, le posizioni dei byte contaminati, la *sensitive sink function* e una traccia dello stack. Successivamente, analizzano i log per determinare quali flussi sono potenzialmente vulnerabili agli attacchi *DOM-based XSS* e quali non lo sono. Per verificare che i flussi individuati siano potenzialmente vulnerabili simulano un *exploit*. Di seguito riportiamo i due metodi sperimentati per creare automaticamente l'*injection*.

Method A: injection at end of URL

Observed URL:

example.url.com/path?param=test&a=b

Generated indjection URL:

*example.url.com/path?param=test&a=b#**INJECT***

Method B: injection into parameter

Observed URL:

example.url.com/path?param=test&a=b

Observed *eval-ed* string:

`var a = 'test';`

Observed taint location:

The 9th through 13th byte of the string-string with the first ‘t’ in test and ending with the last ‘t’ in test.

Generated injection URL:

example.url.com/path?a=b#¶m=INJECT

Nel metodo A, l'injection viene inserita alla fine della stringa. Nel metodo B, tentiamo di inserire l'injection nel valore del parametro che corrisponde alla stringa contaminata nel testo dell'argomento osservato al sink sensibile. Mediante la combinazione di entrambi i metodi, generano l'83% di exploit in più rispetto a quanto ottenuto dal metodo A.

Capitolo 3

Metodologie

Come anticipato nei capitoli precedenti, il primo obiettivo di questo lavoro sarà l'implementazione di un Crawler. Il linguaggio di programmazione che utilizzeremo per tale scopo è *Java*. *Java* infatti ci mette a disposizione alcune librerie, tra cui *JSoup* e *Crawler4j* dedicate appositamente allo sviluppo di applicazioni di questo genere. In particolare, *Jsoup* è un *parser HTML5* le cui API consentono di estrarre dati e manipolare in maniera estremamente semplice documenti [15]. Ci mette a disposizione alcuni comandi i quali permettono di fare il parsing dando in input un url:

- *Document doc = Jsoup.connect("http://html.it/").get();*

Oppure dando in input un file:

- *File input = new File("documento.html");
Document doc = Jsoup.parse(input, "UTF-8", "http://html.it/");*

Oppure dando in input una stringa che rappresenta il contenuto di un documento *HTML*:

- *String html = "<div> DIV inserito </div>";
Document doc = Jsoup.parseBodyFragment(html);
Element body = doc.body();*

Crawler4j invece è un crawler web open source per *Java* che fornisce una semplice interfaccia per la scansione del web. Grazie ad esso è possibile automatizzare alcune operazioni che con *Jsoup* andrebbero fatte manualmente

ed inoltre può lavorare in parallelo mediante l’allocazione di Crawlers multi-thread che permettono di migliorare il throughput e l’accuratezza del nostro Crawler. Poiché le due librerie sono poco documentate e non è presente molto materiale online, valuteremo l’utilizzo dell’una o dell’altra o anche una combinazione di entrambe per sfruttare al massimo le loro caratteristiche. Naturalmente tale decisione sarà collegata anche ai pro e ai contro che queste due ci offrono. I *seeds* che daremo in pasto a questo modulo saranno i siti presenti nell’elenco di *Alexa*, dai quali cercheremo di estrarre informazioni ai fini del nostro studio. Al fine di migliorare la ricerca, sarà fondamentale introdurre l’opzione di scelta della profondità dei link da analizzare in modo tale da porre un limite al crawler nella scansione. Naturalmente, ci servirà un database per poter immagazzinare gli url recuperati dal crawler. La scelta in questo caso ricade sull’utilizzo di *MongoDB*, un database *NoSql*, ciò in quanto gli url che solitamente analizza un crawler potrebbero essere migliaia, un altro database, ad esempio *mysql*, non garantirebbe le stesse performance. Pertanto, le migliori performance e la maggiore scalabilità di questo database non relazionale ci inducono a questa scelta.

Un ulteriore obiettivo di tale lavoro sarà quello di andare effettivamente ad analizzare i link che il nostro crawler darà in output al fine di individuare eventuali vulnerabilità di tipo *DOM-based XSS* degli stessi. Vi sono differenti strumenti per poter effettuare l’analisi di tale vulnerabilità, motivo per il quale bisogna fare una scelta se utilizzare *Burp* e quindi procedere con un’analisi statica, altrimenti progettare un nostro tool di analisi dinamica, ed eventualmente analizzare entrambi i risultati al fine di ottenere maggiori informazioni sulle vulnerabilità riscontrate. Una caratteristica fondamentale che cercheremo di introdurre nel tool qualora sarà implementato, sarà quella di rendere quest’ultimo indipendente, ovvero potrà essere utilizzato successivamente per qualunque tipo di vulnerabilità sul web, a prescindere anche dal browser che andremo ad utilizzare. Nella seguente figura è riportato lo schema di quello che sarà il lavoro in oggetto.

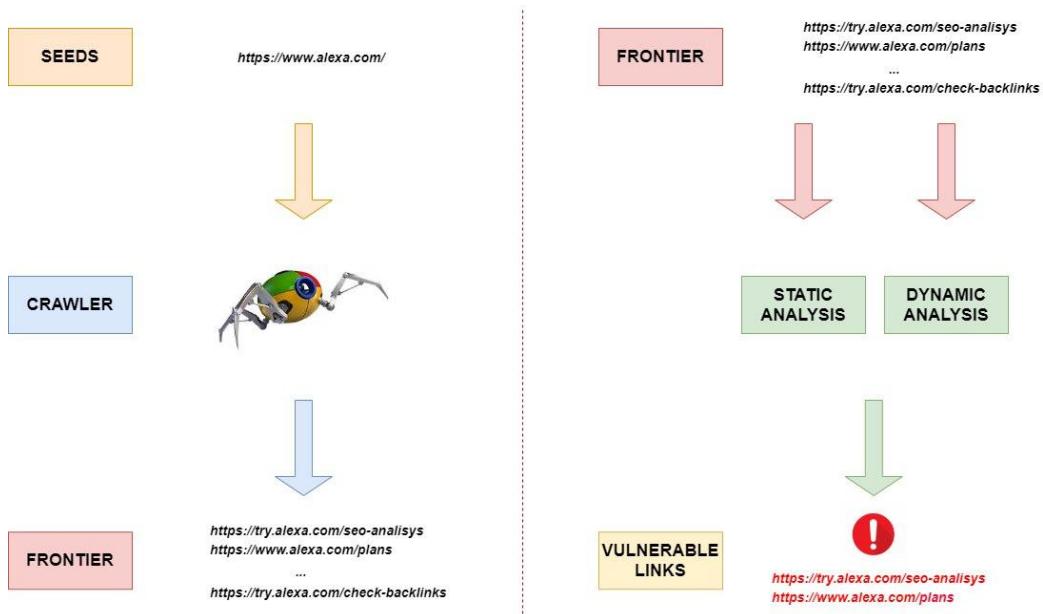


Figura 3.1: Schema generale

Capitolo 4

Realizzazione del progetto

In questo capitolo andremo a descrivere le tecniche di implementazione utilizzate per lo sviluppo del Crawler. Inoltre, verrà descritta la procedura da seguire al fine di poter effettuare un'analisi delle vulnerabilità. Per quest'ultima sono state prese in considerazioni due alternative: analisi statica e analisi dinamica. La *Dynamic taint analysis* richiede l'implementazione di un tool del quale, all'interno del paper [12], ne viene descritto il funzionamento. Tuttavia, per poterlo implementare bisognerebbe disporre di maggiori informazioni al fine di procedere con uno studio più approfondito. A tal fine, la scelta del tipo di analisi da effettuare ricade sull'analisi statica ed in particolare mediante l'utilizzo del tool *Burp Suite*, scanner per vulnerabilità web maggiormente utilizzato da professionisti della sicurezza.

4.1 Realizzazione del crawler

Il Crawler sviluppato si presenta con un'interfaccia semplice ed intuitiva la quale facilita l'utente nell'esecuzione della cosiddetta attività di *crawling*. Difatti, è importante garantire una serie di proprietà, oltre all'**usabilità**, in quanto ritenute fondamentali ai fini di un corretto funzionamento del software. A tal fine, abbiamo cercato di rendere il sistema **robusto**, ovvero in grado di presentare all'utente un messaggio di errore ogniqualvolta un'operazione effettuata non vada a buon fine, offrendo inoltre la possibilità di

eseguire nuovamente l’azione, ciò per impedire malfunzionamenti o arresti improvvisi. Il tool, in quanto scritto in *Java*, potrà essere eseguito su una qualunque macchina sulla quale sia in esecuzione tale linguaggio ma con l’aiuto di *MongoDB*, ovvero il *DBMS* utilizzato per la gestione e memorizzazione dei vari insiemi di url. Per quanto riguarda l’implementazione dell’algoritmo di *crawling*, nel capitolo precedente si era presa in considerazione l’idea di combinare i vantaggi delle librerie *Jsoup* e *Crawler4j*. Tuttavia, non è stato possibile utilizzarle entrambe. In primis perché, *Crawler4j*, come già anticipato, è poco documentata, il che ne rende complesso il suo utilizzo; un ulteriore problema deriva dal fatto che, a causa delle modalità di utilizzo dei metodi che vengono messi a disposizione dalle due librerie, non è stato possibile utilizzarle in combinazione. Ragione per cui la nostra scelta è ricaduta su *Jsoup*. Grazie ai metodi che la libreria in questione mette a disposizione, è possibile in maniera semplice, fare il parsing di pagine *HTML*. Infatti, attraverso il metodo *Jsoup.connect(url).get()* la libreria ci consente di collegarci all’url passato come parametro e di conseguenza estrarre il codice *HTML* della pagina assegnandolo ad un tipo *Document*, sempre messo a disposizione da *Jsoup*. Pertanto, avendo a disposizione questo documento, è possibile andare alla ricerca di eventuali tag $\langle a \ href=\text{"http://www.example.com"} \rangle$ presenti, grazie al metodo *select(“a[href]”)* al fine di estrarre tutti i link a cui è collegata la pagina sottoposta a scansione. Per la sua esecuzione l’algoritmo chiede in input due parametri, un *seed* ed un numero, il quale rappresenta un limite sugli url da sottoporre a scansione, i cosiddetti *Visited*. Di conseguenza, in seguito all’avvio del processo, vengono estratti tutti gli url a partire dal *seed* specificato, secondo le modalità appena descritte previste da *Jsoup*. Dunque, questo *seed* viene inserito all’interno dell’insieme relativo ai *Visited*, in modo tale da evitare di essere sottoposto nuovamente a scansione, mentre i nuovi link estratti finiranno in *frontiera*, pronti per essere processati in seguito. Infatti, come mostrato in figura 2.1, un generico algoritmo di *crawling* prevede un procedimento iterativo, pertanto all’iterata *i*-esima, sarà:

- Estratto il primo url presente attualmente in *frontiera*, che costituisce il *seed* nell’iterata corrente, viene sottoposto a scansione;

- I nuovi url estratti saranno inseriti all'interno della *frontiera* pronti per essere eventualmente processati in seguito;
- Il *seed* verrà eliminato dalla *frontiera* ed inserito nell'insieme degli url *Visited*.

4.1.1 Robots.txt

Come ampiamente riportato in precedenza, un Crawler, durante il suo lavoro, deve rispettare alcune direttive presenti all'interno del cosiddetto file *robots.txt*, il quale si trova nella directory root di un sito. Infatti, un aspetto importante che abbiamo introdotto all'interno del nostro tool, riguarda l'implementazione di una classe *java* che si occupa della gestione di queste direttive. Questa classe da noi realizzata prende in input un url, va alla ricerca del file *robot.txt* del sito in questione e semplicemente tiene conto delle risorse che può o meno sottoporre a scansione. Pertanto, nel momento in cui si trova di fronte un sito bloccato dalle direttive, questo verrà ignorato. In altre parole, non sarà memorizzato all'interno del database e di conseguenza non potrà essere sottoposto a scansione in seguito. La sintassi utilizzata generalmente da un *robots.txt* consiste nell'andare ad identificare, mediante l'attributo *User-agent*, il nome del crawler (es. Googlebot) che deve seguire le particolari direttive, indicate attraverso gli attributi *Allow* (risorse consentite alla scansione) o *Disallow* (risorse non consentite). Poiché il *robots.txt* è un semplice file di testo, la classe *java* realizzata può leggerne semplicemente il contenuto e quindi tener conto di tutti gli *Allow* o *Disallow* relativi alla direttiva *User-agent: **, ovvero quella che vieta o consente la scansione a qualunque Crawler, di conseguenza anche al nostro. Di seguito è riportato un piccolo esempio di *robots.txt* a cui il nostro crawler deve ‘obbedire’:

```

User-agent: *
Disallow: /search
Allow: /search/about
Allow: /search/static
Allow: /search/howsearchworks
Disallow: /sdch
Disallow: /groups
Disallow: /index.html?
Disallow: /?
Allow: /?hl=
Disallow: /?hl=&
Allow: /?hl=*&gws_rd=ssl$
Disallow: /?hl=&*&gws_rd=ssl
Allow: /?gws_rd=ssl$
Allow: /?pt1=true$
Disallow: /imgres
Disallow: /u/
Disallow: /preferences

```

Figura 4.1: Esempio *robots.txt*

4.1.2 Database

Gli insiemi di url generati durante l'utilizzo del tool sono immagazzinati all'interno del database *MongoDB*, un database non relazionale orientato ai documenti. Il vantaggio di utilizzare documenti, cioè dati memorizzati in formato *Bson*, consente l'utilizzo del Crawler anche non avendo una struttura del database su un sistema, in quanto verrà creata dinamicamente nel momento in cui i dati devono essere salvati, senza la necessità di disporre di una struttura predefinita. Pertanto, nel momento in cui viene effettuata una prima attività di *crawling*, verrà generato il database *CrawlerDB* composto da un'unica *collection*, la cui struttura è riportata nella figura seguente:

```
{
  "idSeed": "{type: int, required: true}",
  "seed": "{type: String, required: true}",
  "visited": "{type: Array, required: true}",
  "crawlFrontier": "{type: Array, required: true}"
}
```

Figura 4.2: *Collection*

L'attributo *seed* costituisce l'url dato in input dall'utente nel momento in cui effettua l'attività di *crawling*. A questo *seed* corrisponde un id, *idSeed*,

il quale viene assegnato in automatico (auto_increment). Inoltre, affinché al *seed* possano essere associati i relativi url estratti dal tool, sono presenti i due array *visited* e *crawlFrontier*. Rispettivamente, il primo contiene tutti gli url che sono stati sottoposti ad una particolare attività di *crawling* e da cui ne sono stati estratti ulteriori, il secondo invece contiene gli url estratti a seguito di una scansione, a partire dai *visited* dell'array precedente.

4.2 Analisi con Burp Suite

Con oltre 40.000 utenti, *Burp Suite* è lo scanner per vulnerabilità web più utilizzato al mondo. Professionisti della sicurezza, organizzazioni e team di sviluppo si affidano a *PortSwigger*, il quale consente di ottenere una maggiore consapevolezza sulle vulnerabilità riscontrate in seguito all'analisi. *Burp Suite* è scaricabile in formato *.jar*, il formato di archiviazione di *Java*. A tal fine, per avviarlo occorre scaricare ed installare il *Java Runtime Environment*. Per eseguirlo basterà lanciare il file *.jar* o utilizzare la linea di comando:

```
java -jar C:\burpsuite-version.jar
```

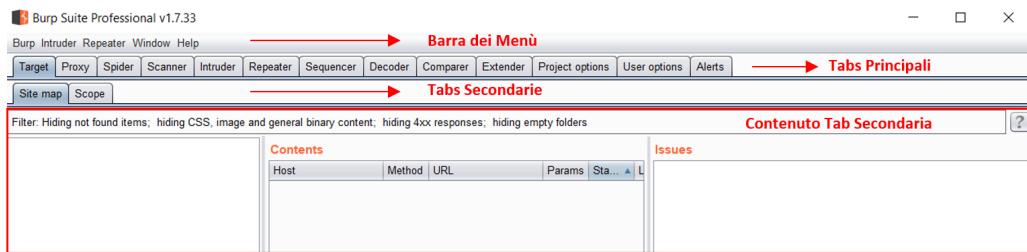


Figura 4.3: Burp Suite: schermata principale

4.2.1 Configurazione di Burp Suite

Come anticipato, in seguito all'apertura delle pagine sul browser, bisogna far passare il traffico attraverso il proxy di *Burp Suite* al fine di consentirne l'analisi. In primis sarà necessario settare su *Burp Suite* le impostazioni del

proxy pertanto, su *Proxy–Options*, impostiamo i parametri come riportato in figura:

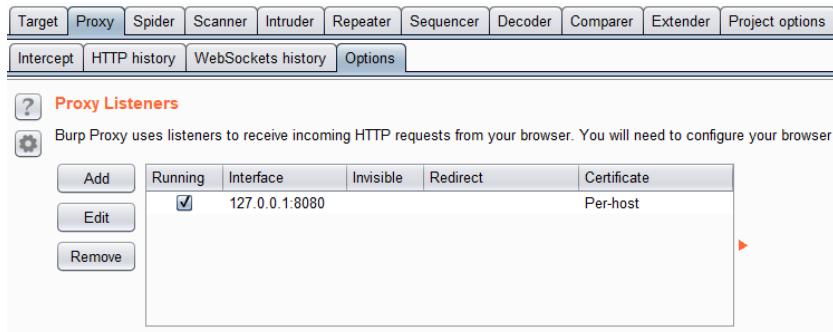


Figura 4.4: *Proxy–Options*

Successivamente selezioniamo *Proxy–Intercept* dove di default la funzione *Intercept is on* è abilitata, quest’ultima difatti consente di intercettare i dati passati dal browser. Qualora non si voglia più intercettare il traffico di rete, basterà disabilitarla e verrà visualizzato *Intercept is off*. Un ulteriore aspetto fondamentale è quello inherente alla scelta del browser su cui effettuare l’analisi di vulnerabilità di un sito, ciò in quanto da browser diversi possono emergere differenti vulnerabilità. In tal caso, per la nostra analisi sceglieremo come browser predefinito *Firefox*. Tuttavia, l’utente in seguito all’attività di *crawling* non avrà vincoli su tale scelta, in quanto il crawler progettato consentirà di scegliere personalmente il browser sul quale lanciare i link estratti.

Apriamo *Firefox* e accediamo alle impostazioni di rete:

Opzioni -> Generale -> Impostazioni di rete

Dopodiché occorre impostare come Http proxy: *127.0.0.1* e come porta: *8080*. Completata la configurazione, il browser non si connetterà più direttamente ai siti, ma tutte le richieste passeranno tramite *Burp*.

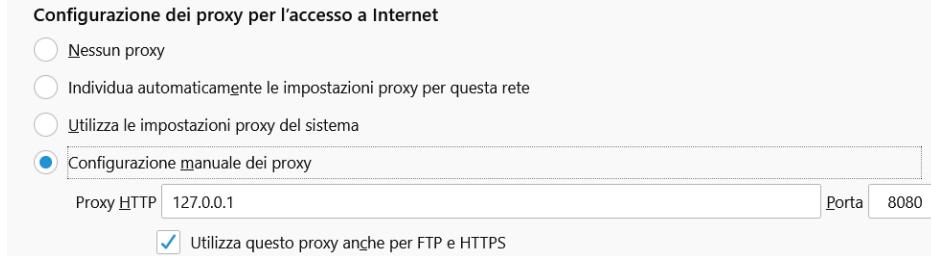


Figura 4.5: Firefox-Configurazione proxy

Configurazione certificato

Per impostazione predefinita, nel momento in cui si visita un sito web *https* tramite *Burp*, il proxy genera un certificato *TLS* per ciascun host, firmato dall'autorità di certificazione *CA*. Alla prima esecuzione di *Burp*, questo certificato viene generato e archiviato localmente. Al fine di poter utilizzare *Burp Proxy* nel modo più efficace con i siti web *https*, è necessario installare il certificato *CA* di *Burp* come radice attendibile (root) nel browser. Inizialmente, provando a connettersi tramite il proxy di *Burp Suite* in un sito abilitato con protocollo *SSL*, il risultato sarà il seguente:

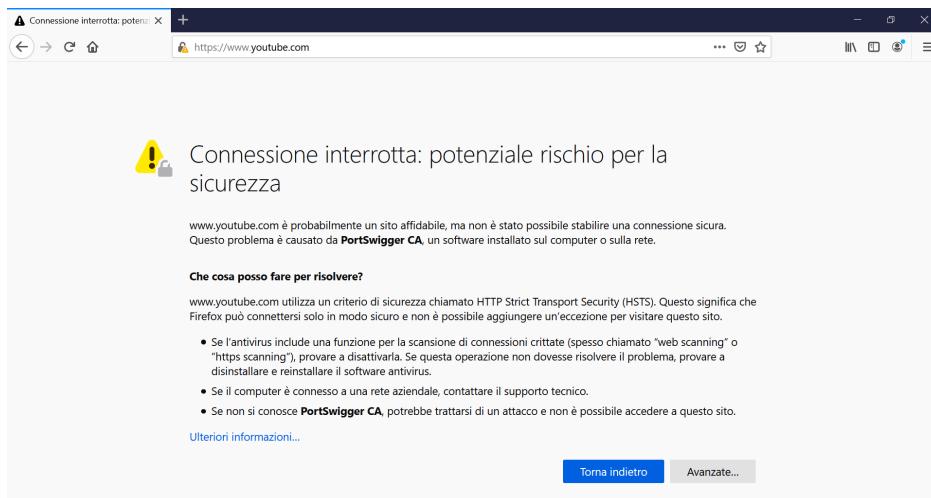


Figura 4.6: Connessione interrotta

Per risolvere questo problema, *PortSwigger* ci fornisce un certificato da installare sul browser (*Firefox* nel nostro caso) [16]. Per poter scaricare il

certificato bisogna collegarsi su *http://burp* dal browser configurato e procedere con il download. In alternativa sarà possibile scaricarlo direttamente dalla pagina di errore indicata in figura:

Ulteriori informazioni -> Download PortSwigger CA

Dopo aver scaricato il certificato bisogna importarlo nel browser:

Opzioni -> Privacy e Sicurezza -> Visualizza certificati -> Autorità

Successivamente, in *Gestione certificati* si procede selezionando la voce *Importa* dove si andrà a caricare il file del certificato *CA Burp* salvato in precedenza. Nella finestra di dialogo che viene visualizzata, selezionare la casella *Affida a questa CA per identificare i siti Web*.

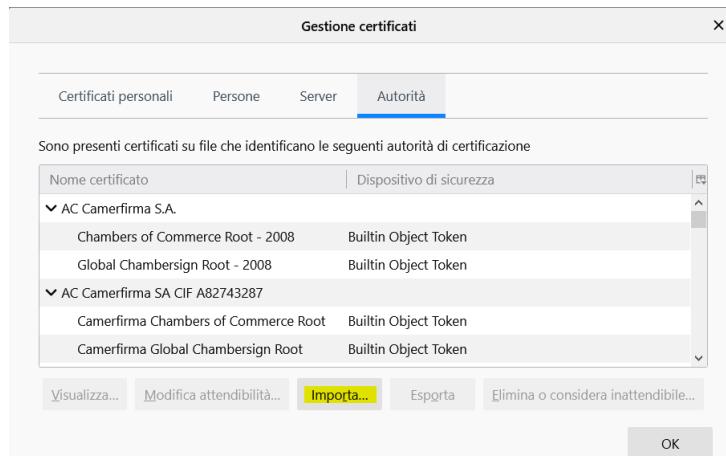


Figura 4.7: *CA Certificate*

A questo punto sarà possibile visitare qualsiasi url *https* tramite *Burp* senza alcun avviso di sicurezza. In seguito all'apertura di un sito sul browser ritorneremo sull'interfaccia di *Burp Suite* dove, all'interno delle tabs *Proxy–HTTP History*, possiamo vedere passare tutte le richieste.

The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. The 'HTTP history' tab is active. A table lists 16 network requests. The columns include: #, Host, Method, URL, Params, Edited, Status, Length, MIME t., Extension, Title, Comment, SSL, and IP. All requests are from 'detectportal.firefox.c...' with a GET method and '/success.txt' URL. Status is 200, Length is 379, Extension is 'txt', and IP is 151.29.122.17.

#	Host	Method	URL	Params	Edited	Status	Length	MIME t...	Extension	Title	Comment	SSL	IP
1	http://detectportal.firefox.c...	GET	/success.txt			200	379	text	txt				151.29.122.17
2	http://detectportal.firefox.c...	GET	/success.txt			200	379	text	txt				151.29.122.17
3	http://detectportal.firefox.c...	GET	/success.txt			200	379	text	txt				151.29.122.17
4	http://detectportal.firefox.c...	GET	/success.txt			200	379	text	txt				151.29.122.17
5	http://detectportal.firefox.c...	GET	/success.txt			200	379	text	txt				151.29.122.17
6	http://detectportal.firefox.c...	GET	/success.txt			200	379	text	txt				151.29.122.17
7	http://detectportal.firefox.c...	GET	/success.txt			200	379	text	txt				151.29.122.17
8	http://detectportal.firefox.c...	GET	/success.txt			200	379	text	txt				151.29.122.17
9	http://detectportal.firefox.c...	GET	/success.txt			200	379	text	txt				151.29.122.17
10	http://detectportal.firefox.c...	GET	/success.txt			200	379	text	txt				151.29.122.17
11	http://detectportal.firefox.c...	GET	/success.txt			200	379	text	txt				151.29.122.17
12	http://detectportal.firefox.c...	GET	/success.txt			200	379	text	txt				151.29.122.17
13	http://detectportal.firefox.c...	GET	/success.txt			200	379	text	txt				151.29.122.17
14	http://detectportal.firefox.c...	GET	/success.txt			200	379	text	txt				151.29.122.17
15	http://detectportal.firefox.c...	GET	/success.txt			200	379	text	txt				151.29.122.17
16	http://detectportal.firefox.c...	GET	/success.txt			200	379	text	txt				151.29.122.17

Figura 4.8: *Proxy-HTTP history*

4.2.2 Scansione vulnerabilità del sito Web

La scansione delle vulnerabilità dei siti web è il modo più rapido per trovare buchi nella sicurezza di un sito. Difatti, al fine di risolvere eventuali problemi, è possibile eseguire regolarmente scansioni automatizzate. Ciò è importante in quanto senza effettuare una scansione delle vulnerabilità, può essere molto difficile riuscire ad evitare un'eventuale violazione dei dati.

Una scarsa consapevolezza della sicurezza evidenzia che i siti web sono spesso costruiti con difetti, di conseguenza ciò comporta un aumento del rischio di attacchi informatici. A tal fine, mediante i test di vulnerabilità che utilizzano software come *Burp Suite*, è possibile ridurre drasticamente tale rischio. Inoltre, anche gli esperti di *Penetration testing* traggono vantaggio dall'uso degli scanner di vulnerabilità. Difatti, gli esseri umani, non possono esaminare un sito web tanto velocemente e nei dettagli pertanto, l'uso di uno scanner fornirà una panoramica della sicurezza di un sito in breve tempo [17].

Burp Scanner

Burp Scanner è uno strumento utilizzato per trovare automaticamente le vulnerabilità di sicurezza nelle applicazioni web. Nella maggior parte degli scanner web, dopo aver fornito all'applicazione l'url iniziale, si procede con la scansione, successivamente viene mostrato l'aggiornamento della barra di avanzamento ed infine elaborano un rapporto di tale analisi. Tuttavia, ciò che emerge è che questo modello di scansione presenta limiti significativi che

portano ad una copertura incompleta, vulnerabilità mancate e sforzi errati. A tal fine, l'approccio di *Burp* per la scansione utilizza un diverso paradigma guidato dall'utente. Quest'ultimo dà un controllo accurato su ogni richiesta che viene scansionata e un feedback diretto sui risultati. Dunque, con il pieno controllo su ciò che viene scansionato è possibile evitare funzionalità pericolose, riconoscere funzionalità duplicate e passare attraverso tutti i requisiti di convalida dell'input che uno scanner completamente automatizzato potrebbe incontrare [18]. *Burp Scanner* può utilizzare metodi sia passivi che attivi per testare la sicurezza di un sito:

- **Modalità di scansione passiva:** lo scanner analizza semplicemente i contenuti delle richieste e delle risposte esistenti e ne deduce le vulnerabilità. Molti tipi di vulnerabilità della sicurezza possono essere rilevati utilizzando solo tecniche passive. Per impostazione predefinita, *Burp Suite* esegue la scansione passiva di tutto il traffico che passa attraverso *Burp Proxy*.
- **Modalità di scansione attiva:** *Burp* invia varie richieste elaborate all'applicazione e successivamente analizza le risposte risultanti alla ricerca di prove di vulnerabilità. La scansione attiva consente di identificare una gamma molto più ampia di vulnerabilità ed è essenziale nel momento in cui si esegue un test completo di un'applicazione.

#	Host	Method	URL	Params	Edited	Status	Length	MIME t...	Extension	Title
1	http://detectportal.firefox.c...	GET	/success.txt			200	379	text	txt	
2	http://detectportal.firefox.c...	GET	/success.txt			200	379	text	txt	
3	http://detectportal.firefox.c...	GET	/success.txt			200	379	text	txt	
4	http://detectportal.firefox.c...	GET	/success.txt			200	379	text	txt	
5	http://detectportal.firefox.c...	GET	/sucess.txt	http://detectportal.firefox.com/success.txt?ip6			79	text	txt	
6	http://detectportal.firefox.c...	GET	/succes.txt	Add to scope			79	text	txt	
7	http://detectportal.firefox.c...	GET	/succes.txt	Spider from here			79	text	txt	
8	http://detectportal.firefox.c...	GET	/succes.txt				79	text	txt	
9	http://detectportal.firefox.c...	GET	/succes.txt				79	text	txt	
10	http://detectportal.firefox.c...	GET	/succes.txt				79	text	txt	

Figura 4.9: Avvio scansione

Nella scansione manuale, si selezionano le singole richieste (o interi rami della mappa del sito di destinazione) e si utilizza il menu di scelta rapida per avviare scansioni attive su di essi. Mentre, nella scansione in tempo reale durante la navigazione, si configura lo scanner per eseguire automaticamente scansioni attive su tutte le richieste che passano attraverso il proxy durante la navigazione dell'applicazione. Quando viene visualizzata una richiesta contenente parametri interessanti, è possibile eseguire una scansione attiva di tale richiesta per ottenere un rapido feedback su eventuali vulnerabilità. Gli elementi inviati per la scansione attiva vengono aggiunti alla coda di scansione attiva, che visualizza lo stato di ciascun elemento e il numero di problemi rilevati. È possibile monitorare l'avanzamento della scansione e utilizzare il menu di scelta rapida per annullare o assegnare nuovamente la priorità ai singoli elementi [19].

Burp Intruder Repeater Window Help						
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts						
Issue activity Scan queue Live scanning Issue definitions Options						
#	Host	URL	Status	Issues	Requests	Errors
1	http://detectportal.firefox.c...	/success.txt	finished	2	266	1
2	http://detectportal.firefox.c...	/success.txt	finished	1	250	
3	http://detectportal.firefox.c...	/success.txt	finished	1	250	
4	http://detectportal.firefox.c...	/success.txt	finished	1	250	
5	http://detectportal.firefox.c...	/success.txt	66% complete		287	

Figura 4.10: *Scan queue*

Per il nostro studio di vulnerabilità utilizzeremo la modalità di scansione attiva, definita come il più aggressivo di questi metodi, la quale simulerà effettivamente un attacco al fine di trovare le vulnerabilità. *Burp Suite* consente di personalizzare le scansioni in base alle esigenze, fornisce un metodo rapido e discreto ma anche una visione più approfondita della sicurezza. *Burp Scanner* è difatti in grado di rilevare una serie di bugs comuni, inclusi *Cross-site scripting (XSS)* e *SQL Injection* e tutta una serie di altre vulnerabilità. Tuttavia, noi ci concentreremo maggiormente, come anticipato, sulle vulnerabilità *Cross-site scripting (XSS)*, *DOM-based XSS*.

Avvio di una scansione

In primis, apriamo il sito web su cui eseguire la scansione sul browser e osserviamo contemporaneamente cosa succede su *Burp Proxy*. Ciò che possiamo notare è che l'analisi di *Burp* non consente di navigare nel sito finché non selezioniamo il pulsante *Forward* con il quale inoltriamo le richieste ed infine, al completamento dell'inoltro delle richieste, sarà possibile visualizzare sul browser la pagina web correttamente caricata.

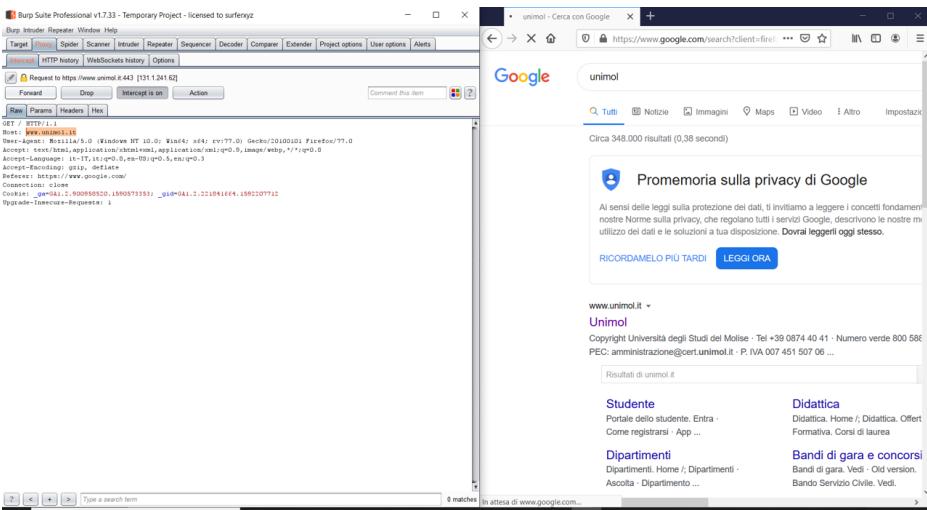


Figura 4.11: *Burp Proxy*

A questo punto, in *Burp*, spostandoci sulla scheda *Target* ed in particolare in *Site map*, è possibile avere una visione completa dei link attraversati e delle vulnerabilità riscontrante in seguito all’attività di scansione.

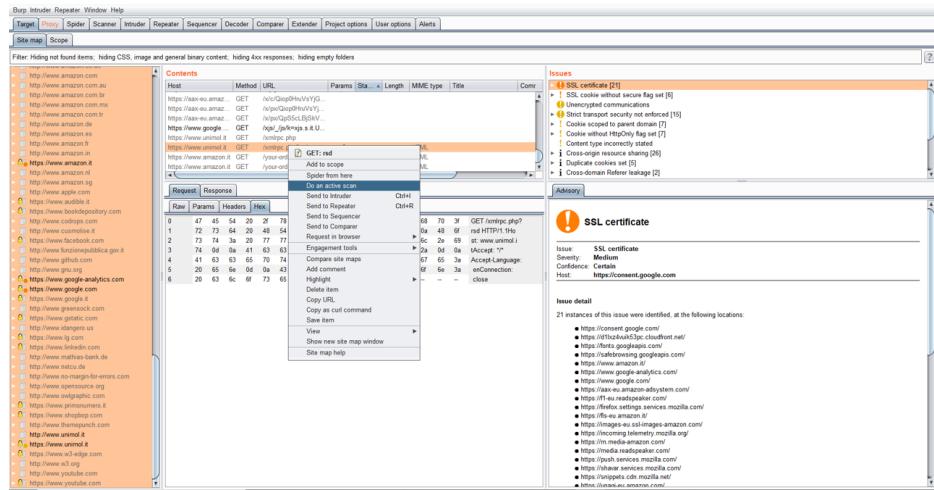


Figura 4.12: Target

Dopo aver selezionato gli elementi sui quali eseguire la scansione ed aver avviato tale attività, è possibile accedere alla sezione *Scanner -> Scan queue*, all'interno della quale è possibile vedere l'avanzamento delle scansioni effettuate e i risultati ottenuti durante l'analisi.

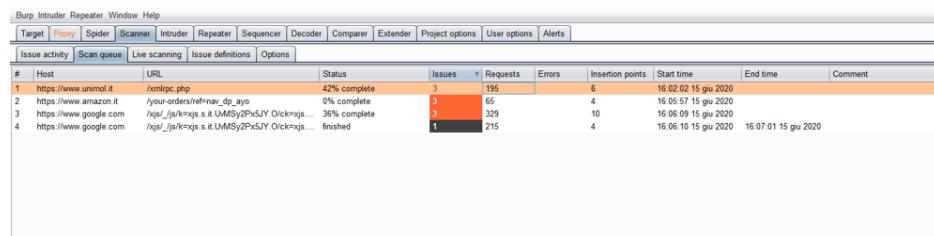


Figura 4.13: Scan queue

Inoltre, nella scheda *Issues activity* abbiamo una visione completa delle vulnerabilità riscontrate.

The screenshot shows the Burp Suite interface with the 'Issues activity' tab selected. The main pane displays a table of findings:

#	Time	Action	Issue type	Host	Path	Insertion point	Severity	Confidence	Comment
207	16:00:33 19 giu 2020	Issue found	Cachable HTTP/1.0 response	https://media.reading.../X/buckets/monitor/collections/changes/rec...	/		Information	Certain	
208	16:00:33 19 giu 2020	Issue found	Cachable certificate	https://media.reading...	/		Information	Certain	
209	16:00:40 19 giu 2020	Issue found	Strict transport security not enforced	https://media.reading...	/		Low	Certain	
210	16:00:40 19 giu 2020	Issue found	Cachable HTTPS response	https://www.unomi.it	/xampp.php		Information	Certain	
211	16:00:40 19 giu 2020	Issue found	Flash cross-domain policy	https://www.unomi.it	/xampp.php		Information	Certain	
212	16:00:40 19 giu 2020	Issue found	Flash cross-domain policy	https://www.unomi.it	/xampp/unomi.xml		Low	Certain	
213	16:00:58 19 giu 2020	Issue found	Robots.txt file	https://www.amazon.it	/robots.txt		Information	Certain	
214	16:00:58 19 giu 2020	Issue found	Robots.txt file	https://www.google...	/robots.txt		Information	Certain	
215	16:07:39 19 giu 2020	Issue found	XSS (DOM-based)	https://www.unomi.it/your-orders/review_dp_ayo		name of an address	High	Firm	
216	16:07:39 19 giu 2020	Issue found	XSS (DOM-based)	https://www.unomi.it/your-orders/review_dp_ayo		name of an arbitrar...	Information	Certain	
217	16:08:13 19 giu 2020	Issue found	XPath injection	https://www.amazon.it	/your-orders/review_dp_ayo	User-Agent HTTP h...	High	Firm	

The bottom pane shows a detailed view of the last entry (XPath injection), including issue details, background information, remediation steps, and classification.

Figura 4.14: *Issues activity*

Tuttavia, poichè la nostra analisi si concentra sulle vulnerabilità *Cross-site scripting* ed in particolare *DOM-based XSS*, andremo a personalizzare l'attività di scansione in quanto *Burp* ci consente di poter selezionare quali tipi di problemi rilevare. A tal fine, in *Scanner -> Options* andiamo a selezionare l'attività di scansione sulle vulnerabilità *Cross-site scripting*.

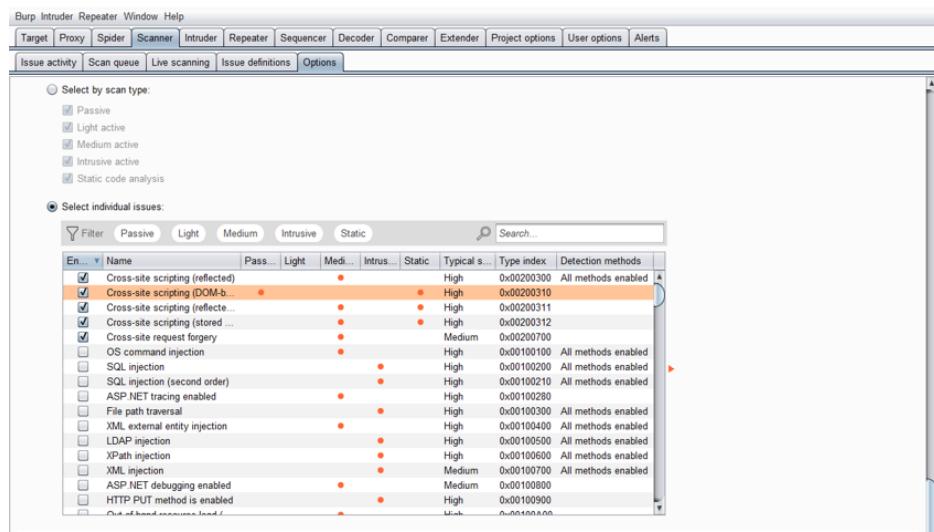


Figura 4.15: Scelta analisi

Continuiamo la navigazione sul sito dell'*Unimol* nella sezione *Amministrazione* dove, come è possibile osservare nelle immagini riportate di seguito, abbiamo rilevato la presenza di vulnerabilità *XSS* ed in particolare *DOM-based XSS* sull'url [/www.unimol.it/ateneo/amministrazione/](http://www.unimol.it/ateneo/amministrazione/)

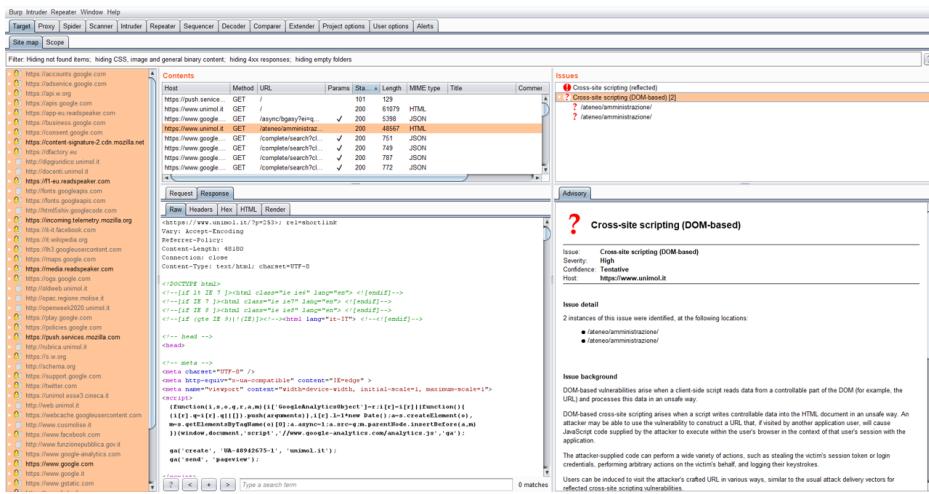


Figura 4.16: Vulnerabilità *Cross-Site Scripting*

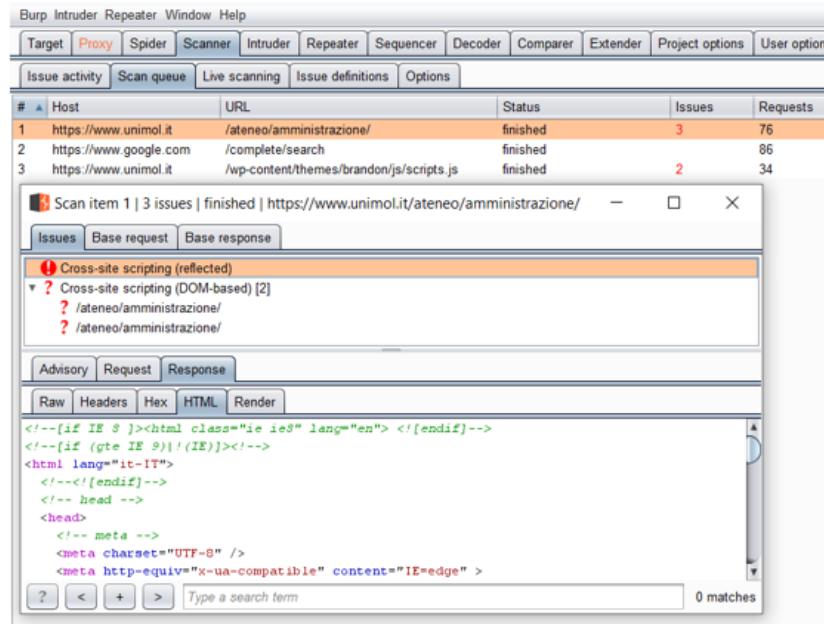


Figura 4.17: *Scan Issue*

Capitolo 5

Manuale Utente

In questa sezione è riportato il Manuale Utente relativo al Crawler sviluppato in ambito del presente lavoro. Pertanto, di seguito verrà mostrata un'anteprima delle varie schermate che compongono il tool.

5.1 Splash Screen

Poiché, come anticipato più volte, il tool memorizza i dati all'interno del database *Mongo*, all'esecuzione del programma verrà fatto un breve check il quale verifica che sia installato il *DBMS* sul sistema in questione ed inoltre che il server *Mongod* sia in esecuzione su tale sistema. In caso di risposta negativa, al fine di avere, come detto, un sistema robusto, verrà generato un messaggio di errore e verrà arrestata l'esecuzione. Al contrario, nel caso in cui il check sia andato a buon fine, il tool mostra la prima schermata, ovvero un semplice *Welcome Panel* da cui è possibile dare il via al programma, altrimenti chiuderlo.



Figura 5.1: Splash Screen

5.2 Welcome Panel

La figura seguente mostra il *Welcome Panel* del tool, da cui è possibile iniziare ad utilizzare il programma, altrimenti chiuderlo attraverso l'apposito pulsante.



Figura 5.2: Welcome Panel

5.3 Schermata *Crawling*

La schermata principale del tool, quella da cui è possibile eseguire l'attività di scansione su un particolare url, si presenta nel seguente modo:

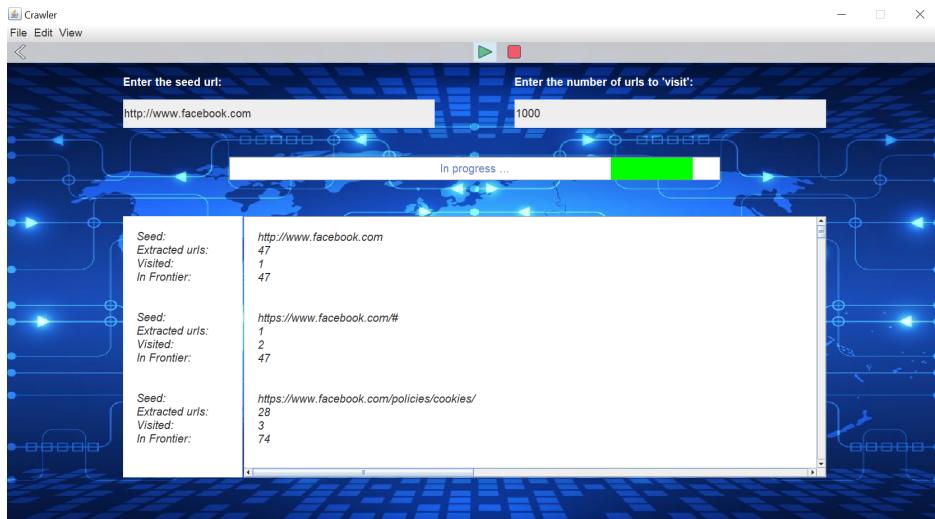


Figura 5.3: Schermata *Crawling*

Al fine di eseguire la scansione il tool chiede in input 2 parametri:

- Nel primo campo di testo bisogna inserire l'url di partenza su cui effettuare l'analisi, il cosiddetto *seed*. In generale, la scelta del *seed* per l'attività di *crawling* è molto importante, poichè da questo dipenderà il risultato finale. Infatti, una buona scelta del *seed* potrebbe portare il crawler all'estrazione di numerosi url, al contrario di una scelta considerata “meno buona” che rende l'attività più rapida e con un numero di risultati molto basso o addirittura nullo. Nell'esempio in figura si è deciso di utilizzare come *seed* l'indirizzo <http://www.facebook.com>;
- Il secondo campo richiede invece di specificare il numero di url da visitare ovvero, il numero di url su cui il tool andrà ad effettuare la scansione e di conseguenza estrarne ulteriori link. Il valore in questione è necessario in quanto abbiamo deciso di porre una sorta di soglia all'attività di scansione, che altrimenti potrebbe continuare quasi all'infinito. Nell'esempio in figura si è deciso di utilizzare come limite di scansione 1000 url.

In seguito all'inserimento dei parametri richiesti, l'attività potrà iniziare nel momento in cui l'utente seleziona il pulsante *Play* nella barra in alto. A

questo punto, il tool verificherà:

- che uno o entrambi i campi non siano vuoti;
- che il parametro inserito nel primo campo sia valido, ovvero del tipo *protocollo://dominio/path*;
- che il parametro inserito nel secondo campo sia valido ovvero, in primis che sia un intero e inoltre che sia maggiore di zero;
- che il link dato in input non sia eventualmente “bloccato” dal *robots.txt* del sito in questione.

Se la verifica non va a buon fine, naturalmente verrà visualizzato un messaggio di errore. Al contrario, l’attività di *crawling* ha inizio. Tale attività proseguirà finché il numero di url visitati non sia pari all’input specificato, ma è possibile utilizzare il pulsante *Stop* nella *sub bar* per porre fine alla scansione. All’esecuzione della scansione, il tool visualizza una *progress bar* e *freeze*a il resto della schermata impedendo all’utente di compiere ulteriori operazioni sui pulsanti oppure sui campi di testo (ad eccezione del pulsante *Pausa*). Nel riquadro centrale vengono riportati 4 attributi che tengono traccia di ciò che sta avvenendo, consultabili attraverso la *scroollbar* laterale. In particolare, ogni blocco costituito dai 4 attributi corrisponde ad un’iterata dell’algoritmo. Dettagliamo il significato dei valori in questione:

- **Prima iterata:**

- *Seed* specifica qual è l’url che abbiamo dato in input, quindi il seme su cui viene eseguito l’algoritmo. Nel caso in figura è *http://www.facebook.com*, pertanto a partire da questo, il crawler tirerà fuori un insieme di altri url;
- *Extracted urls* specifica il numero di url che sono stati estratti dal *seed* in questione. In questo caso, a partire dal seed *http://www.facebook.com*, sono stati estratti 47 url;

- *Visited* rappresenta il numero di url visitati fino a questo momento, ovvero il numero di url che sono stati sottoposti a scansione. Naturalmente in questo caso indicherà 1 in quanto l'unico url processato è il *seed* dato in input;
- *In frontier* è l'attributo che tiene traccia del numero di url che si trovano attualmente in *frontiera*. Nel caso della prima iterata sono 47, ovvero pari al numero di *Extracted urls*.

- **Seconda iterata:**

- Nelle iterate successive alla prima, *Seed* rappresenta l'url che è stato estratto dalla *frontiera* e che quindi diventerà il nuovo seme su cui effettuare la scansione;
- *Extracted urls*: nella seconda iterata è pari a 1, in quanto dalla pagina <https://www.facebook.com/#> è stato estratto un solo url;
- *Visited* naturalmente ora sarà pari a 2, in quanto gli url ‘visitati’ a questo punto sono il vecchio seme <http://www.facebook.com> e il nuovo pescato dalla frontiera <https://www.facebook.com/#>;
- *In frontier* ora sarà pari a 47, ovvero nella seconda iterata abbiamo 47 url in *frontiera*. Ciò in quanto ai 47 che avevamo inizialmente, è stato sottratto il *seed* iniziale, che in quanto scansionato è stato eliminato dalla *frontiera*, ed è stato aggiunto l'unico url estratto.

Eventualmente potrebbe comparire un ulteriore attributo che notifica che c’è stato un problema con un particolare url, pertanto il tool non è riuscito a scansionarlo.

5.4 Status bar

Ogni schermata del tool presenta una *status bar*, mediante la quale è possibile eseguire semplici operazioni o navigare tra le varie schermate. Il seguente *Menu Item* illustra le opzioni *New* e *Exit*. La prima consente di fare un *refresh* della schermata in questione da eventuali input inseriti o da una

scansione appena effettuata. La seconda consente semplicemente di uscire dal programma.

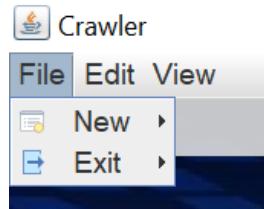


Figura 5.4: File

Il *Menu Item* seguente mostra invece le opzioni *Delete one* e *Delete all* in riferimento all'eliminazione di una o di tutte le scansioni effettuate. Infatti, selezionando la prima opzione, si viene ricondotti ad un'ulteriore schermata del tool, la quale verrà mostrata in seguito e da cui è possibile eliminare una particolare attività di *crawling*. L'opzione *Delete all* invece consente di eliminare l'intero database, di conseguenza tutti gli insiemi di url che sono stati memorizzati a seguito di eventuali attività di *crawling* effettuate.

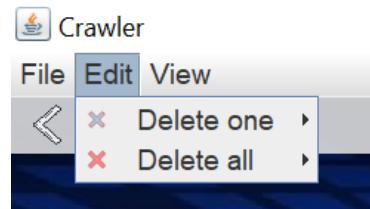


Figura 5.5: Edit

L'ultimo *Menu Item* mostra l'opzione *History*, il quale riconduce ad una successiva schermata da cui è possibile visualizzare la storia relativa ad una scansione in precedenza effettuata, mediante la selezione di un *seed*.

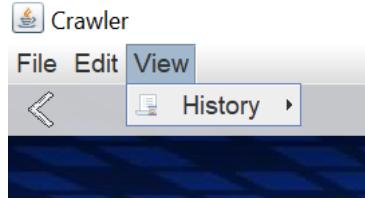


Figura 5.6: View

5.5 Schermata *Seeds*

Nella seguente figura vi è la schermata *Seeds* del tool. L'area centrale costituisce la lista dei *seeds* dati in input dall'utente nel momento in cui ha effettuato una o più attività di scansione. Alla prima esecuzione del programma, verrà visualizzato il messaggio **No Seeds** ad indicare che non è presente nessun *seed*, ovvero l'utente non ha effettuato ancora il *crawling*, oppure ha precedentemente eliminato ogni attività.

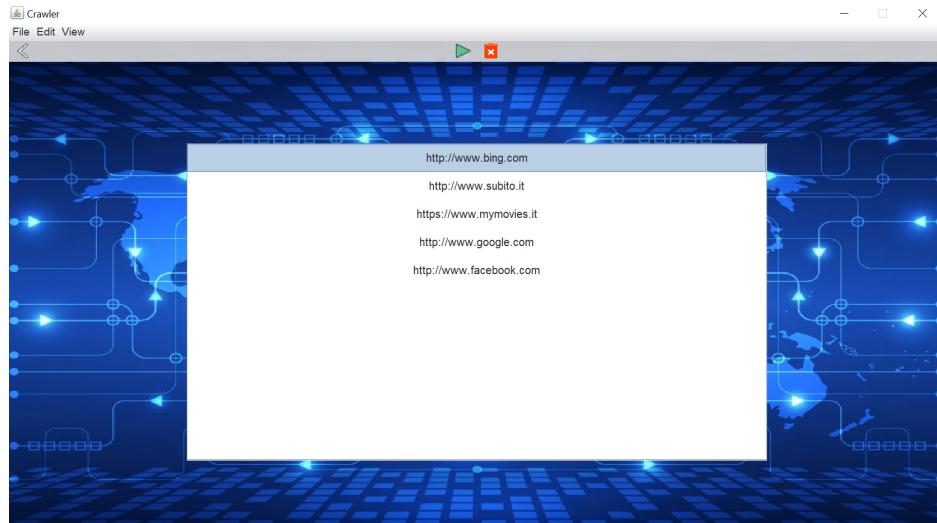


Figura 5.7: Schermata *Seeds*

Tuttavia, in caso di scansioni avvenute, è possibile selezionare uno dei *seeds* presenti nella lista ed eseguire una delle due opzioni mostrate nella *sub bar* in alto.

- Il pulsante *Play* effettua la sua azione dopo che è stato selezionato un *seed* nell'area sottostante. Infatti, questo conduce l'utente verso la

schermata successiva del tool, descritta in seguito, relativa alla *History* riguardante il *seed* selezionato;

- Il secondo pulsante, rappresentato da un cestino, fa una *Delete* del *seed* selezionato e dei relativi url restituiti come output dal tool. In altre parole, consente la cancellazione dal database dell'attività di *crawling* precedentemente effettuata a partire dal *seed* selezionato. Pertanto, oltre al *seed*, saranno eliminati i relativi *Visited* e url in *frontiera*.

In caso di eccezioni generate dal tool a seguito dell'utilizzo di tali opzioni, verrà visualizzato anche in questo caso un messaggio di errore.

5.6 Schermata *History*

In questa schermata è possibile visualizzare gli insiemi di url che sono stati costruiti a seguito di una particolare scansione selezionata nella schermata precedente. Nella figura in questione stiamo visualizzando la *History* della scansione effettuata sul seed <http://www.facebook.com>:

ID	URL
1	https://i.facebook.com/l.php?u=https%3A%2F%2Fen.facebookbrand.com%2Ftrademarks%2F
2	https://i.facebook.com/l.php?u=https%3A%2F%2Fen.facebookbrand.com%2Fguidelines%2F
3	https://www.facebook.com/help/245628975556747?ref=tos
4	https://www.facebook.com/help/103873106370583?ref=tos
5	https://www.facebook.com/legal/commercial_terms
6	https://www.facebook.com/legal/music_guidelines
7	https://www.facebook.com/help/1506822589577997?ref=tos
8	https://www.facebook.com/ad_guidelines.php
9	https://www.facebook.com/legal/self_service_ads_terms
10	https://www.facebook.com/page_guidelines.php
11	https://developers.facebook.com/policy/
12	https://developers.facebook.com/credits
13	https://www.facebook.com/payments_terms
14	https://www.facebook.com/policies/commerce
15	https://i.facebook.com/l.php?u=https%3A%2F%2Fwww.facebookbrand.com%2F&h=AT0NI2...
16	https://o-o.facebook.com/legal/terms/update
17	https://fr-fr.facebook.com/legal/terms/update
18	https://co-it.facebook.com/legal/terms/update
19	https://de-de.facebook.com/legal/terms/update
20	https://es-es.facebook.com/legal/terms/update
21	https://ar-ar.facebook.com/legal/terms/update
22	https://sq-al.facebook.com/legal/terms/update

Figura 5.8: Schermata *History*

In particolare, vengono mostrate 3 tabs:

- La tab relativa al *Seed*, che mostra qual è stato il *seed* dato in input nel momento in cui è stata effettuata questa particolare scansione;
- La tab *Visited*, che invece mostra tutti gli url che sono stati sottoposti a scansione durante l'attività di *crawling* in questione e da cui sono stati quindi estratti ulteriori url;
- La tab *Frontier*, mostrata in figura su. Ci sono tutti gli url che si trovano in *frontiera*, quindi tutti gli url che sono stati estratti a seguito di questa particolare attività di *crawling*.

Inoltre nella *sub bar* sono presenti i due pulsanti di opzioni visibili in figura: *Console*, il quale consente di aprire la console mostrata nella seguente sezione di questo capitolo e *Download*, la cui azione è quella di scaricare l'intera *frontiera* all'interno di un file *.txt*, specificando un path sul disco locale.

5.7 Console

Nella figura seguente viene mostrata la *Console*, ovvero uno strumento molto importante del tool il quale consente di lanciare nel browser gli url estratti a seguito dell'attività di *crawling*. In dettaglio, questa *Console* consente di:

- Eseguire un generico comando inserendolo come input nel relativo campo di testo, il più grande visibile in figura. In questo caso, è possibile lasciare inalterati gli altri campi;
- Lanciare un particolare browser, selezionando uno tra quelli visualizzati, oppure inserendo nell'apposito campo il path al browser sul disco locale, specificando inoltre un range di id da eseguire, presi dall'insieme dei *Visited* o *Frontier*. Inoltre, lasciando inalterato il pulsante *run individual link*, verrà lanciato nel browser specificato il primo url del range. Di conseguenza, verrà successivamente mostrato un pulsante che consente l'esecuzione, uno alla volta, dei successivi url del range. Altrimenti, selezionando il pulsante precedente, verrà attivato il comando *run all links* il quale fa il rendering di tutti i link specificati nel range utilizzando il browser selezionato.

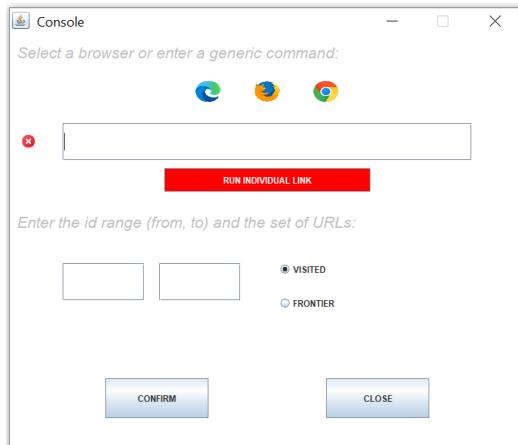


Figura 5.9: Console

La *Console* garantisce un'importante proprietà di cui gode il tool: l'indipendenza. Difatti, mediante le funzioni messe a disposizione da questo strumento, è possibile utilizzare il Crawler per mandare in esecuzione su un browser gli url specificati. Pertanto, nel momento in cui si desidera utilizzare un tool di analisi sui link in questione, è possibile eseguirlo in parallelo e non si è vincolati dal tool da utilizzare.

Capitolo 6

Studio vulnerabilità

In questo capitolo sono descritte le varie fasi inerenti al lavoro svolto, illustrando le modalità utilizzate per raggiungere gli obiettivi prefissati, relative sia all'utilizzo del Crawler realizzato, sia allo studio di una vulnerabilità *DOM-based XSS* sui link restituiti in output da questo tool.

6.1 Alexa Rank

Siamo partiti collegandoci al sito *Alexa Rank* il quale costituisce un sistema di ranking globale che classifica milioni di siti web in ordine di popolarità. Viene calcolato osservando i visitatori unici medi giornalieri stimati ed il numero di *page view* per un dato sito negli ultimi 3 mesi. Più basso è il *rank* di *Alexa*, più popolare è il sito web [20]. Abbiamo utilizzato il ranking in questione al fine di avere dei siti da utilizzare come *seeds* per il Crawler implementato. Di seguito è riportato il ranking dei *TOP 10 siti Alexa* più popolari [21]:

	Luogo	Tempo giornaliero sul ...	Pagine Viste Giornalier...	% di traffico provenien...	Totale siti collegati
1	Google.com	14:09	15.39	0,40%	1.982.688
2	Youtube.com	14:02	7.87	15,80%	1.523.322
3	Tmall.com	06:53	2.95	1,00%	4.562
4	Facebook.com	18:40	8.11	8,00%	3.365.025
5	Qq.com	03:44	4.02	3,10%	319.765
6	Baidu.com	08:44	4.56	5,20%	139.931
7	Sohu.com	03:45	4.70	1,40%	40.625
8	Login.tmall.com	05:04	1.00	0,60%	72
9	Taobao.com	04:28	3.59	2,90%	34.234
10	360.cn	03:16	4.02	0,40%	20.658

Figura 6.1: Alexa Top 10

6.2 Crawling

Simuliamo un'attività di *crawling* sul primo sito della classifica, *Google.com*, a tal fine abbiamo dato in input al nostro tool il sito in questione, specificando una profondità di url da scansionare pari a 30:

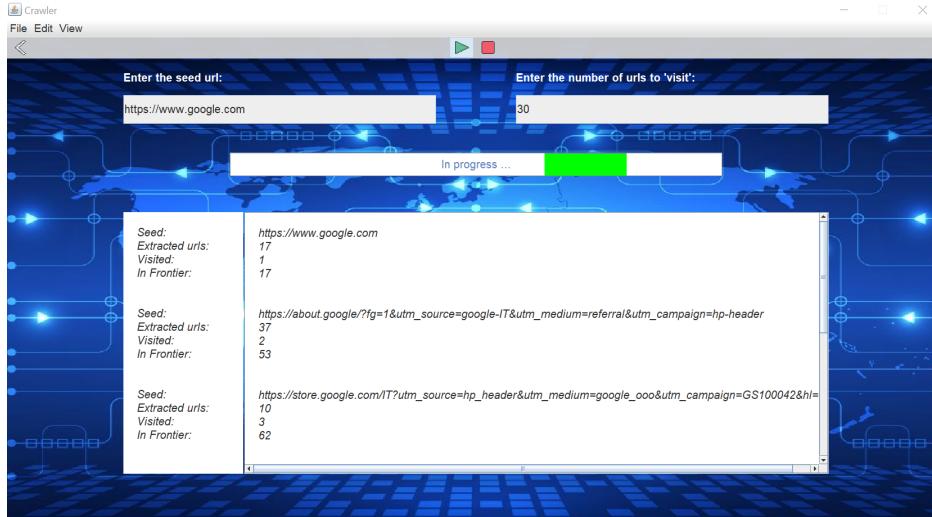


Figura 6.2: Crawling *Google.com*

Terminata l'attività di *crawling*, visualizziamo la relativa *History* navigando tra le tabs del tool. Utilizzando la *scroll bar* laterale, è possibile visualizzare l'intera lista di url ed in particolare, nella tab *Visited* vi sono i 30 url

sottoposti a scansione, mentre nella tab *Frontier* abbiamo ottenuto 867 url, ovvero tutti quelli estratti a partire dai 30 url “visitati”:

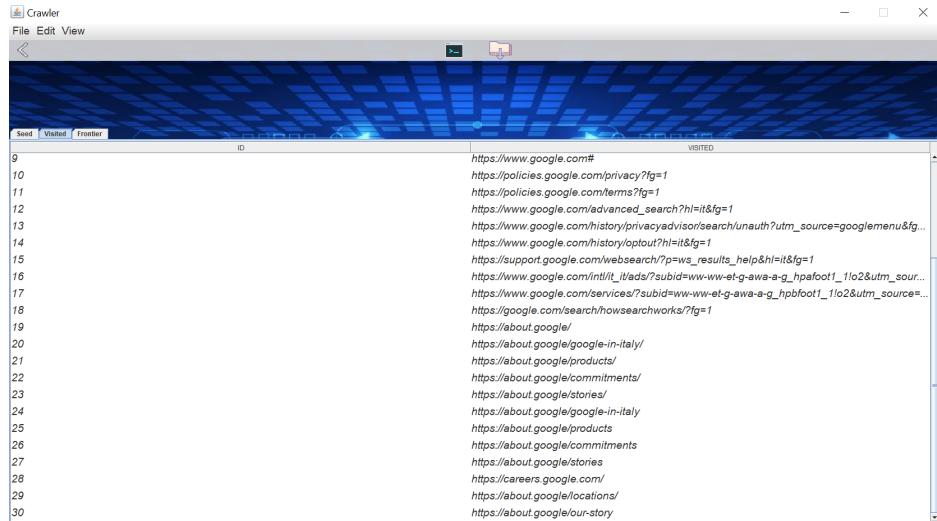


Figura 6.3: Tab Visited

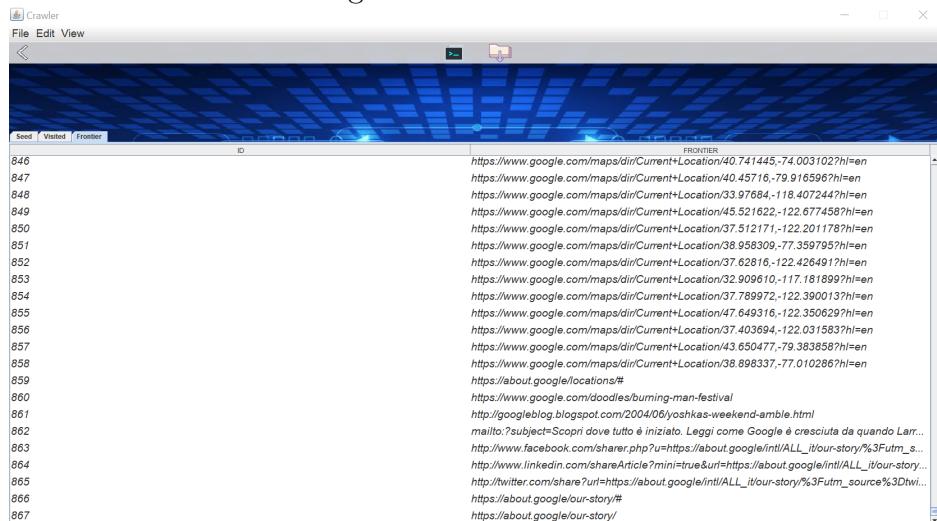


Figura 6.4: Tab *Frontier*

A questo punto, così come mostrato nei capitoli precedenti, abbiamo abilitato il proxy su *Burp Suite* e attraverso l'utilizzo della console del Crawler, abbiamo specificato:

- *Firefox*, come browser da utilizzare per fare il rendering degli url;

- *Visited*, come insieme da cui prelevare gli url:
 - con un range di id da 1 a 20;
- *Frontier*, come insieme da cui prelevare gli url:
 - con un range di id da 1 a 30.

In questo modo è possibile lanciare nel browser, tutti insieme o singolarmente, gli url indicati nel range. Avendo in parallelo il *proxy* abilitato, l'esecuzione dei vari url passa attraverso *Burp Suite* da cui è possibile procedere per ciascuno di essi ad un'analisi di vulnerabilità *Cross-site Scripting*.

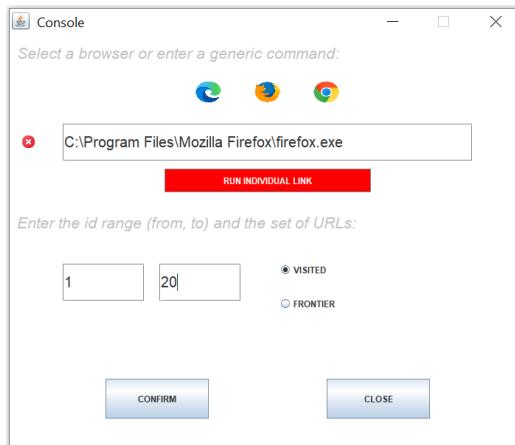


Figura 6.5: Console

Il primo url del range è *www.google.com* e ciò che *Burp* ha rilevato è riportato nella seguente figura:

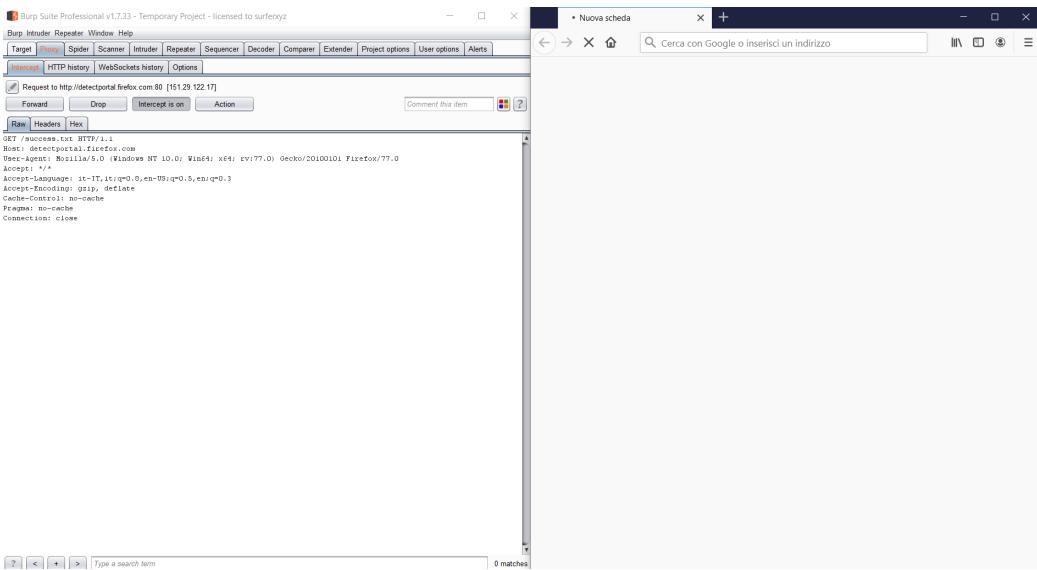


Figura 6.6: Caricamento del primo url

Procedendo mediante l'apposito pulsante della console, al rendering nel browser dei successivi url del range, otteniamo il caricamento di tutte le pagine:

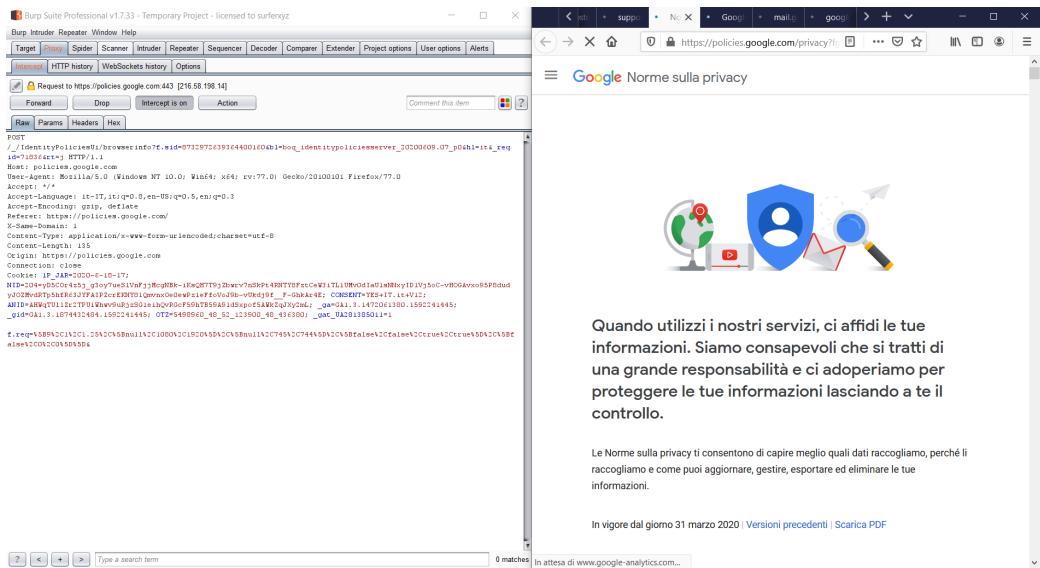


Figura 6.7: Caricamento di tutti gli url

A questo punto in *Burp Proxy* selezioniamo *Forward* per inoltrare le richieste al fine di consentirne il completo caricamento.

Prima di procedere con la scansione, andiamo a settare le impostazioni dello Scanner in *Options* nella sezione *Scan Issues* e selezioniamo le vulnerabilità che vogliamo analizzare, nel nostro caso *Cross-site Scripting*.

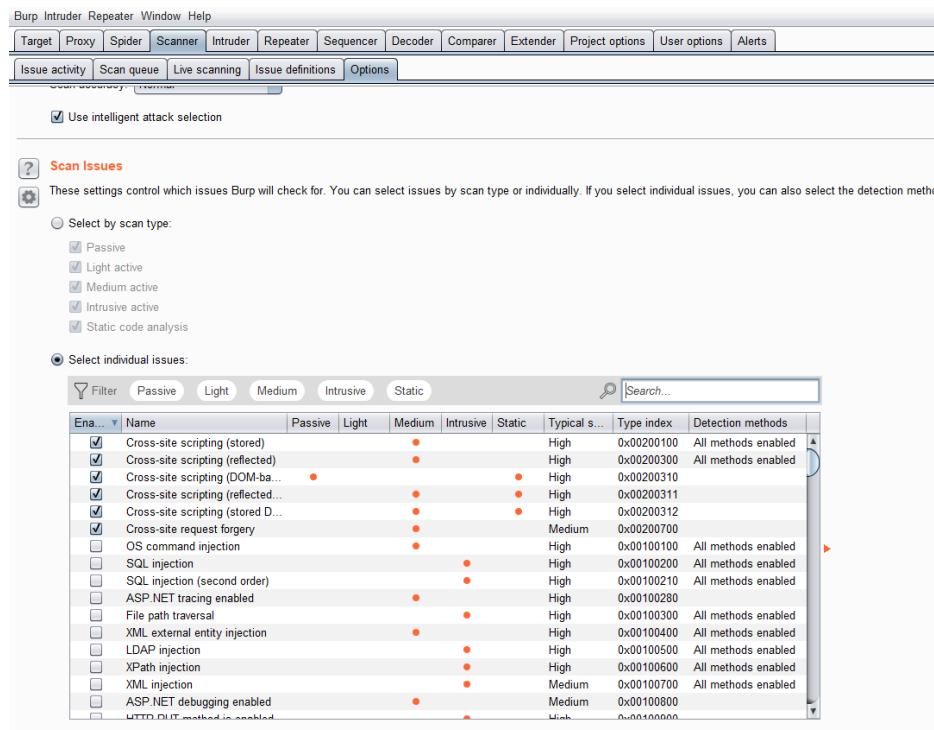


Figura 6.8: Options *Cross-site Scripting*

In *Target* si andrà a definire una *Site map* all'interno della quale vi sono gli url riscontrati da *Burp* durante il caricamento delle pagine. Da quest'ultima sarà possibile selezionare tutti gli url o singoli branch da sottoporre a scansione. In seguito alla scelta di tali elementi, procediamo con una modalità di scansione attiva.

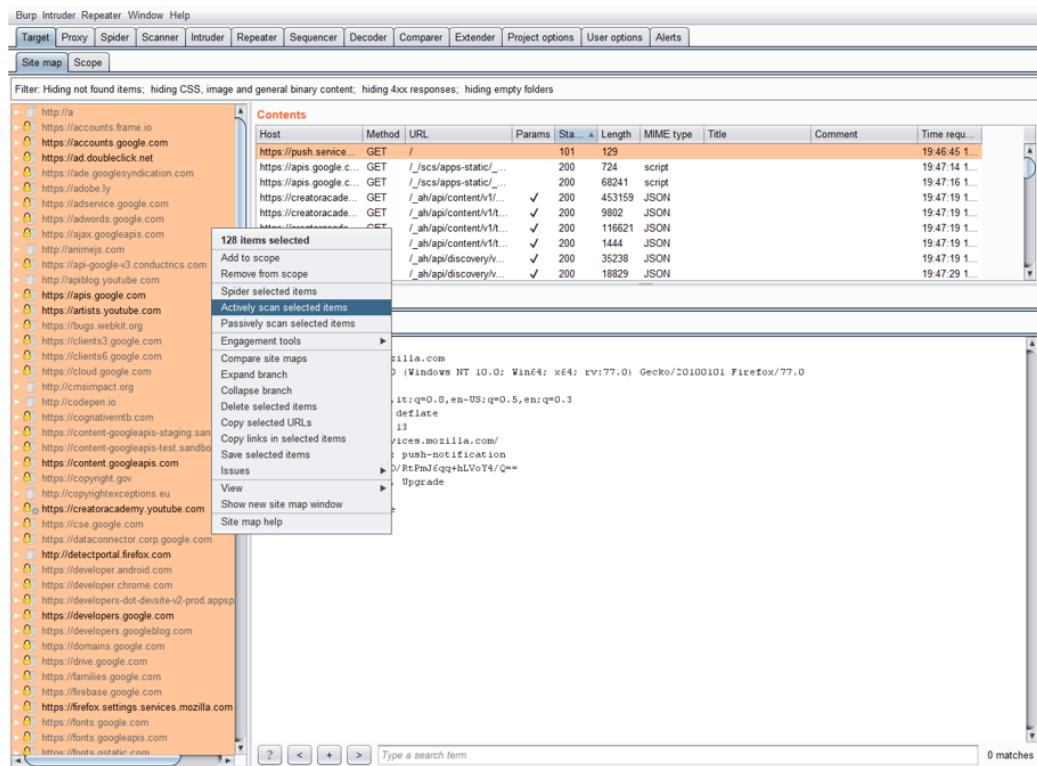


Figura 6.9: *Site map*

All'avvio della scansione in *Scanner -> Scan queue* è possibile osservare l'avanzamento della scansione dei vari url sottoposti a tale analisi.

#	Host	URL	Status	Issues	Requests	Errors	Insertion points	Start time	End time	Comment
1	https://normandy.cdn.mozilla.net	/api/v1/	finished	1	37	4	17:46:48 16 giu 2020	17:47:13 16 giu 2020		
2	https://push.services.mozilla.net	/	finished	26	3	17:46:48 16 giu 2020	17:47:12 16 giu 2020			
3	https://www.google.com	/	finished	99	10	17:46:48 16 giu 2020	17:47:24 16 giu 2020			
4	https://content-signature-2.firebaseio.com	/chains/remote-settings/content-signature.m	finished	34	4	17:46:48 16 giu 2020	17:46:59 16 giu 2020			
5	https://ihaveno.services.mozilla.net	/downloads	finished	192	26	17:46:48 16 giu 2020	17:48:33 16 giu 2020			
6	https://normandy.cdn.mozilla.net	/api/v1/extension/	finished	1	36	4	17:46:49 16 giu 2020	17:47:22 16 giu 2020		
7	https://normandy.cdn.mozilla.net	/api/v1/extension/signed/	finished	33	4	17:46:49 16 giu 2020	17:47:07 16 giu 2020			
8	https://normandy.cdn.mozilla.net	/api/v1/approval_request/	finished	1	36	4	17:46:53 16 giu 2020	17:47:53 16 giu 2020		
9	https://normandy.cdn.mozilla.net	/api/v1/extension/	finished	1	36	4	17:46:54 16 giu 2020	17:47:32 16 giu 2020		
10	https://normandy.cdn.mozilla.net	/api/v1/recipe/	finished	24	4	17:47:04 16 giu 2020	17:58:58 16 giu 2020			
11	https://normandy.cdn.mozilla.net	/api/v1/recipe/signed/	finished	33	4	17:47:08 16 giu 2020	17:47:53 16 giu 2020			
12	https://normandy.cdn.mozilla.net	/api/v1/recipe_revision/	finished	1	36	4	17:47:12 16 giu 2020	17:47:51 16 giu 2020		
13	https://www.google.com	/advanced_search	finished	64	4	17:47:12 16 giu 2020	17:47:59 16 giu 2020			
14	https://www.google.com	/advanced_search	finished	148	6	17:47:12 16 giu 2020	17:51:00 16 giu 2020			
15	https://www.google.com	/history/optout	finished	50	4	17:47:13 16 giu 2020	17:47:32 16 giu 2020			
16	https://www.google.com	/history/optout	finished	134	6	17:47:13 16 giu 2020	17:47:53 16 giu 2020			
17	https://www.google.com	/history/privacyadvisor/search/unauth	finished	33	4	17:47:13 16 giu 2020	17:47:32 16 giu 2020			
18	https://www.google.com	/history/privacyadvisor/search/unauth	finished	49	6	17:47:24 16 giu 2020	17:47:44 16 giu 2020			
19	https://www.google.com	/images/nav_logo299.webp	finished	47	4	17:47:24 16 giu 2020	17:47:36 16 giu 2020			
20	https://www.google.com	/init/init_ads/	finished	64	4	17:47:32 16 giu 2020	17:47:47 16 giu 2020			
21	https://www.google.com	/init/init_ads/	finished	274	9	17:47:32 16 giu 2020	17:48:32 16 giu 2020			
22	https://www.google.com	/log	finished	47	4	17:47:36 16 giu 2020	17:48:13 16 giu 2020			
23	https://www.google.com	/log	finished	63	6	17:47:37 16 giu 2020	17:48:37 16 giu 2020			
24	https://www.google.com	/preferences	finished	90	4	17:47:47 16 giu 2020	17:48:24 16 giu 2020			
25	https://www.google.com	/preferences	finished	132	5	17:47:48 16 giu 2020	17:48:36 16 giu 2020			
26	https://www.google.com	/preferences	finished	174	6	17:47:53 16 giu 2020	17:48:54 16 giu 2020			
27	https://www.google.com	/search	finished	47	4	17:47:53 16 giu 2020	17:49:29 16 giu 2020			
28	https://www.google.com	/services/	finished	54	4	17:47:53 16 giu 2020	17:48:23 16 giu 2020			
29	https://www.google.com	/services/	finished	87	9	17:47:53 16 giu 2020	17:48:29 16 giu 2020			
30	https://www.google.com	/wizRpcul/_NwzRpcul/	finished	33	4	17:48:13 16 giu 2020	17:48:28 16 giu 2020			
31	https://ihaveno.services.mozilla.net	/downloads	finished	33	4	17:48:14 16 giu 2020	17:48:33 16 giu 2020			
32	https://play.google.com	/log	83% complete	132	30	[159 skipped]	17:53:55 16 giu 2020			
33	https://play.google.com	/log	87% complete	137	30	[83 skipped]	17:53:55 16 giu 2020			
34	https://play.google.com	/log	80% complete	131	30	[40 skipped]	17:53:55 16 giu 2020			
35	https://play.google.com	/log	finished	58	7	17:53:55 16 giu 2020	17:54:10 16 giu 2020			
36	https://ajax.googleapis.com	/ajax/libs/angularjs/1.5.9/angular-animate.min.js	finished	40	3	17:53:55 16 giu 2020	17:54:12 16 giu 2020			
37	https://ajax.googleapis.com	/ajax/libs/angularjs/1.5.9/angular-messages.min.js	finished	40	3	17:53:55 16 giu 2020	17:54:11 16 giu 2020			
38	https://ajax.googleapis.com	/ajax/libs/angularjs/1.5.9/angular.min.js	finished	40	3	17:53:55 16 giu 2020	17:54:25 16 giu 2020			
39	https://ajax.googleapis.com	/ajax/libs/angularjs/1.7.9/angular-animate.min.js	finished	48	4	17:53:55 16 giu 2020	17:54:12 16 giu 2020			
40	https://ajax.googleapis.com	/ajax/libs/angularjs/1.7.9/angular-aria.min.js	finished	48	4	17:53:55 16 giu 2020	17:54:10 16 giu 2020			
41	https://ajax.googleapis.com	/ajax/libs/angularjs/1.7.9/angular-messages.min.js	finished	48	4	17:53:55 16 giu 2020	17:54:13 16 giu 2020			
42	https://ajax.googleapis.com	/ajax/libs/angularjs/1.7.9/angular-sanitize.min.js	finished	48	4	17:53:55 16 giu 2020	17:54:13 16 giu 2020			
43	https://ajax.googleapis.com	/ajax/libs/angularjs/1.7.9/angular-touch.min.js	finished	48	4	17:53:55 16 giu 2020	17:54:13 16 giu 2020			

Figura 6.10: *Scan queue*

Per ciascuno sarà possibile visualizzare i relativi risultati ottenuti.

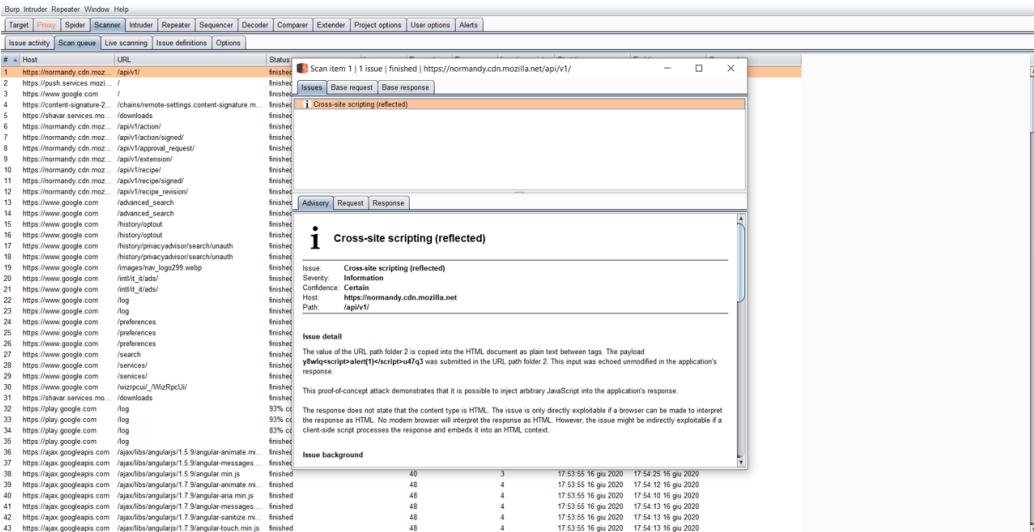


Figura 6.11: *Vulnerability*

Inoltre, in *Scanner -> Issue activity* sarà possibile avere una visione completa di tutte le vulnerabilità riscontrate.

#	Time	Action	Issue type	Host	Path	Insertion point	Severity	Confidence
1	17-47 04 16 giu 2020	Issue found	i Cross-site scripting (reflected)	https://normandy.cdn.mozilla.net	/api/v1/	URL path folder 2	Information	Certain
2	17-47 05 16 giu 2020	Issue found	i Cross-site scripting (reflected)	https://normandy.cdn.mozilla.net	/api/v1/action/	URL path folder 3	Information	Certain
3	17-47 16 16 giu 2020	Issue found	i Cross-site scripting (reflected)	https://normandy.cdn.mozilla.net	/api/v1/extension/	URL path folder 3	Information	Certain
4	17-47 25 16 giu 2020	Issue found	i Cross-site scripting (reflected)	https://normandy.cdn.mozilla.net	/api/v1/approval_request/	URL path folder 3	Information	Certain
5	17-47 34 16 giu 2020	Issue found	i Cross-site scripting (reflected)	https://normandy.cdn.mozilla.net	/api/v1/recipe_revision/	URL path folder 3	Information	Certain
6	17-47 51 16 giu 2020	Issue found	?	https://shavar.servic...	/downloads		Medium	Tentative

Figura 6.12: *Issue activity*

Procediamo allo stesso modo analizzando ulteriori link in *frontiera* al fine di ottenere un'analisi più profonda:

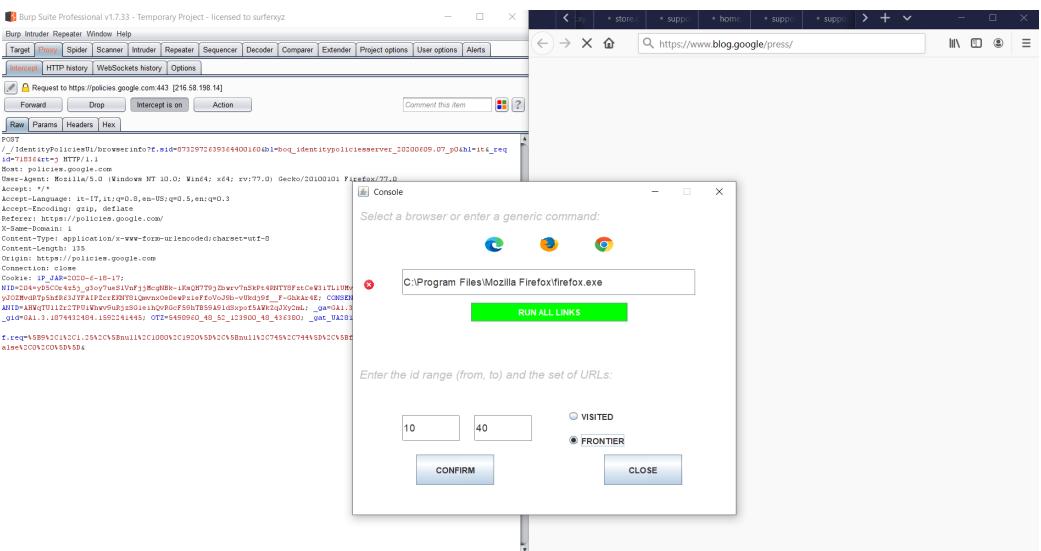


Figura 6.13: Analisi frontiera

In seguito a quest'ultima, sulla base dei risultati ottenuti, riscontriamo, nei seguenti url sottoposti a scansione, un maggior numero di vulnerabilità, tra cui *DOM-based XSS*.

Burp Intruder Repeater Window Help

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts

#	Time	Action	Issue type	Host	Path	Insertion point	Severity
1	17:47:04 16 giu 2020	Issue found	! Cross-site scripting (reflected)	https://normandy.cd...	/api/v1/...	URL path folder 2	Information
2	17:47:05 16 giu 2020	Issue found	! Cross-site scripting (reflected)	https://normandy.cd...	/api/v1/action/	URL path folder 3	Information
3	17:47:16 16 giu 2020	Issue found	! Cross-site scripting (reflected)	https://normandy.cd...	/api/v1/extension/	URL path folder 3	Information
4	17:47:25 16 giu 2020	Issue found	! Cross-site scripting (reflected)	https://normandy.cd...	/api/v1/approval_request/	URL path folder 3	Information
5	17:47:34 16 giu 2020	Issue found	! Cross-site scripting (reflected)	https://normandy.cd...	/api/v1/recipe_revision/	URL path folder 3	Information
6	17:47:51 16 giu 2020	Issue found	? Cross-site request forgery	https://shavar.servic...	/downloads		Medium
7	18:55:19 16 giu 2020	Issue found	! Cross-site scripting (DOM-based)	https://www.google....	/contact/		High
8	18:55:19 16 giu 2020	Issue found	! Cross-site scripting (DOM-based)	https://www.google....	/js/google.js		High
9	18:56:59 16 giu 2020	Issue found	? Cross-site request forgery	https://shavar.servic...	/downloads		Medium
10	18:57:13 16 giu 2020	Issue found	? Cross-site request forgery	https://www.youtube...	/api/stats/qoe		Medium

Advisory Request 1 Response 1 Request 2 Response 2

!

Cross-site scripting (DOM-based)

Issue: Cross site scripting (DOM-based)
 Severity: High
 Confidence: Firm
 Host: https://www.google.com
 Path: /contact/

Issue detail
 The application may be vulnerable to DOM-based cross-site scripting. Data is read from `window.location.search` and passed to the `'insertBefore'` function of JQuery via the following statements:

- `a = window.location.search`
- `window._gat||(a=document.createElement("script")),a.type="text/javascript",a.async=0,a.src=(https:"==document.location.protocol?"https://ssl":"http://"+window.location.host+window.location.pathname+window.location.search).replace(/\?/,"&_gat=1"),document.documentElement.appendChild(a)`

Issue background
 DOM-based vulnerabilities arise when a client-side script reads data from a controllable part of the DOM (for example, the URL) and processes this data in an unsafe way.

DOM-based cross-site scripting arises when a script writes controllable data into the HTML document in an unsafe way. An attacker may be able to use the vulnerability to construct a URL that, if visited by an user's session with the application.

The attacker-supplied code can perform a wide variety of actions, such as stealing the victim's session token or login credentials, performing arbitrary actions on the victim's behalf, and logging their keystrokes. Users can be induced to visit the attacker's crafted URL in various ways, similar to the usual attack delivery vectors for reflected cross-site scripting vulnerabilities.

Burp Suite automatically identifies this issue using static code analysis, which may lead to false positives that are not actually exploitable. The relevant code and execution paths should be reviewed to determine if the issue is exploitable.

Issue remediation

Figura 6.14: Vulnerabilità DOM-based XSS

6.2.1 Attività di analisi

Per testare l'effettiva funzionalità del lavoro svolto, in seguito alla fase di *crawling* su alcuni siti riportati nell'elenco di Alexa, abbiamo effettuato un'attività di scansione per rilevare le possibili vulnerabilità sugli url estratti. Nelle analisi svolte, al fine di disporre di risultati che ci consentano di ottenere una visione generale, abbiamo scelto nella prima fase di procedere all'attività di *crawling* con una profondità media pari a 30. Successivamente, sulla base dei risultati ottenuti procediamo con un'analisi attraverso il tool *Burp Suite*, mediante il browser *Firefox*, degli url estratti presenti in *Visited* con un range medio da 1 a 15, mentre per quelli presenti in *Frontiera* scegliamo un range più ampio al fine di procedere ad una scansione che analizzi gli url più

in profondità. Difatti, sulla base delle prime analisi è emerso che dagli url prelevati dalla *frontiera* abbiamo riscontrato più vulnerabilità.

Di seguito riportiamo i risultati di alcune analisi effettuate. All'interno di tali tavelle vi sono illustrati i *seeds* da cui partiamo, le vulnerabilità riscontrate e i relativi url.

SEED	VULNERABILITA' RISCONTRATE		PATH VULNERABILI
	VISITED	FRONTIER	
www.google.com	Cross-site scripting (reflected) Severity: High		Host: https://normandy.cd/ Path: /api/v1 Host: https://normandy.cd/ Path: /api/v1/action/ Host: https://normandy.cd/ Host: https://normandy.cd/ Path: /api/v1/approval_request Host: https://normandy.cd/ Path: /api/v1/recipe_revision/ Host: https://shavar.service.com Path: /downloads
	Cross-site request forgery Severity: Medium		Host: https://www.google.com Path: /contact/ Path: /js/google.js Path: /downloads Host: https://www.youtube.com Path: /api/stats/qoe
		Cross-site scripting (DOM-based) Severity: High	

Figura 6.15: Vulnerabilità *Google.com*

SEED	VULNERABILITÀ RISCONTRATE		PATH VULNERABILI
	VISITED	FRONTIER	
www.mymovies.it	Cross-site request forgery Severity: Medium	Cross-site scripting (reflected) Severity: High	Host: https://shavar.services.mozilla.com Path: /downloads
			Host: https://15.taboola.com Path: /tb
			Host: https://www.mymovies.it Path: /live/logout.asp
			Host: https://ws.mymovies.it Path: /v12/funzioni/live/app-salva-social.asp
			Host: https://www.mymovies.it Path: /ajax/mappe/mappa.asp
			Host: https://www.mymovies.it Path: /boxoffice/
			Host: https://www.mymovies.it /cinema/agrigento/ [name of an arbitrarily supplied URL parameter] /cinema/agrigento/
			Host: https://ws.mymovies.it Path: /v12/ajax/info_profilo.asp
			Host: https://ws.mymovies.it Path: /profilo/accedi/ Host: https://ws.mymovies.it Path: /profilo/accedi/
		Cross-site scripting (reflected) Severity: High	https://15.taboola.com/tb
			Host: https://www.mymovies.it /cinema/milano/ /cinema/monzabrianza/ /cinema/napoli/ /cinema/padova/ /cinema/palermo/ /cinema/perugia/ /cinema/pesaroerbino/ /cinema/roma/ /cinema/treviso/ /cinema/udine/ /cinema/venezia/
		Cross-site scripting (DOM-based) Severity: Medium	Host: https://www.mymovies.it/oscar/2020/

Figura 6.16: Vulnerabilità *mymovie.it*

SEED	VULNERABILITA' RISCONTRATE		PATH VULNERABILI
	VISITED	FRONTIER	
www.subito.it	Cross-site scripting (reflected) Severity: High		Host: https://incoming.telemetry.mozilla.org/submit/messaging-system/cfr/1/15347a39-7c65-4cf6-8803-fe1ce087e28b Host: https://www.subito.it/magazine/wp/xmlrpc.php Host: https://www.subito.it/magazine/wp/xmlrpc.php Host: https://info.subito.it/policies/privacy.htm
	Cross-site scripting (DOM-based) Severity: High		Host https://www.subito.it/magazine/wp/xmlrpc.php Host: https://www.subito.it/magazine/wp/xmlrpc.php Host: https://help.netflix.com Path: /api/personalization [alerts parameter] /api/personalization [deviceErrors parameter] /api/personalization [recommendations parameter]
www.Netflix.com	Cross-site scripting (reflected) Severity: Information		https://help.netflix.com en/api/personalization [alerts parameter] /en/api/personalization [deviceErrors parameter] /en/api/personalization [recommendations parameter] https://help.netflix.com /it/api/personalization [alerts parameter] /it/api/personalization [deviceErrors parameter] /it/api/personalization [recommendations parameter]
	Cross-site scripting (reflected) Severity: High		https://help.netflix.com Path: /support/103458 https://help.netflix.com /support/109370 https://help.netflix.com Path: /support/2101 https://help.netflix.com Path: /support/412 https://help.netflix.com Path: /support/64916 https://help.netflix.com Path: /support/45074

Figura 6.17: Vulnerabilità *Subito.it* *Netflix.com*

SEED	VULNERABILITÀ RISCONTRATE		PATH VULNERABILI
	VISITED	FRONTIER	
www.youtube.com	Cross-site request forgery Severity: Medium		Host: https://www.youtube.com Path: /api/stats/qo Host: https://www.youtube.com Path: /channel/UCzuqhhs6NWbgTzMuM09WKDQ
		Cross-site request forgery Severity: Medium	Host: https://creatoracademy.youtube.com Path: /_ah/api/analytics/v1/record/save Host: https://aus5.mozilla.org Path: /update/6/Firefox/77.0.1/20200602222727/WIN
		Cross-site scripting (reflected) Severity: High	Host: creatoracademy.youtube.com Path: /_ah/api/static/proxy Host: https://www.inps.it Path: /search122/s/ricerca.js Host: https://incoming.telemetry.mozilla.org Path: /submit/messaging-system/cfr/1/27b5a8bb-eab9-0000-0000-000000000000 Host: https://www.inps.it Path: /search122/s/query-1.12.0.min.js Host: serviziweb2.inps.it Path: /PassiWeb/jsp/SSLAUTH/scLogin.jsp
	Cross-site scripting (reflected) Severity: High		Host: https://servizi2.inps.it Path: /VirtualAgent/SGWeb/EABPluginChat [name of an arbitrarily supplied URL parameter] /VirtualAgent/SGWeb/EABPluginChat [viewIcon parameter] /VirtualAgent/SGWeb/EABPluginChat [Referer HTTP header] /VirtualAgent/SGWeb/EABPluginChat [User-Agent HTTP header] /VirtualAgent/SGWeb/EABPluginChat [Referer HTTP header] /VirtualAgent/SGWeb/EABPluginChat [User-Agent HTTP header] Host: https://servizi2.inps.it Path: /VirtualAgent/SGWeb/EABPluginChat [name of an arbitrarily supplied URL parameter] /VirtualAgent/SGWeb/EABPluginChat [Referer HTTP header] /VirtualAgent/SGWeb/EABPluginChat [User-Agent HTTP header] /VirtualAgent/SGWeb/EABPluginChat [Referer HTTP header] /VirtualAgent/SGWeb/EABPluginChat [User-Agent HTTP header]
	Cross-site scripting (reflected) Severity: Information		
		Cross-site scripting (reflected) Severity: High	
www.inps.it			

Figura 6.18: Vulnerabilità *Youtube.com inps.it*

SEED	VULNERABILITA' RISCONTRATE		PATH VULNERABILI
	VISITED	FRONTIER	
www.facebook.com	Cross-site request forgery Severity: Medium		Host: https://ro-ro.facebook.com Path: /ajax/fbz Host: https://sc-it.facebook.com Path: /ajax/fbz Host: https://www.facebook.com Path: /ajax/fbz Host: https://www.facebook.com Path: /ajax/fbz
		Cross-site request forgery Severity: Medium	Host: https://es-la.facebook.com Path: /ajax/fbz Host: https://fr-fr.facebook.com Path: /ajax/fbz Host: https://developers.facebook.com Path: /ajax/fbz Host: https://www.facebook.com Path: /ajax/fbz Host: http://www.raitalia.it Path: /dlPortal/Rai/Page-ab48738e-506a-4d73-8abb-09248a9db82b.html /dlPortal/RaiPage-ab48738e-506a-4d73-8abb-09248a9db82b.html /dlPortal/RaiPage-ab48738e-506a-4d73-8abb-09248a9db82b.html /dlPortal/RaiPage-ab48738e-506a-4d73-8abb-09248a9db82b.html
www.rai.it	Cross-site scripting (DOM-based) Severity: High		Host: https://fri-italia01.wt-eu02.net Path: /602039762736393/rdfs
	Cross-site scripting (reflected) Severity: Information		Host: https://shavar.services.mozilla.com Path: /downloads Host: www.raipaly.it Path: https://www.raipaly.it/video/2020/06/Domenic-a-in---Gianna-Nannini-canta-You've-got-a-friend-e-La-donna-cannon--1a9a0c30-4e54-4a89-9294-00dc2e78e67.html Host: www.raipaly.it Path: https://www.raipaly.it/video/2020/06/rainews24---vaccino-firmato-l'accordo-per-400-milioni-di-dosi-per-leuropa-arriver-a-fine-anno-273e3884-b8b9-4117-bb8d-c24fe7fcce42.html
	Cross-site request forgery Severity: Medium		

Figura 6.19: Vulnerabilità *Facebook.com*

SEED	VULNERABILITÀ RISCONTRATE		PATH VULNERABILI	
	VISITED	FRONTIER		
www.mediaset.it	Cross-site scripting (reflected) Severity: High		Host: https://feed.entertainment.tv.theplatform.eu Path: /f/PR1GhC/mediaset-prod-all-clips [name of an arbitrarily supplied URL parameter] /f/PR1GhC/mediaset-prod-all-clips [sort parameter]	
			Host: https://feed.entertainment.tv.theplatform.eu Path: /f/PR1GhC/mediaset-prod-all-clips [name of an arbitrarily supplied URL parameter] /f/PR1GhC/mediaset-prod-all-clips [sort parameter] Host: https://feed.entertainment.tv.theplatform.eu Path: /f/PR1GhC/mediaset-prod-all-listings Host: https://feed.entertainment.tv.theplatform.eu Path: /f/PR1GhC/mediaset-prod-all-listings Host: https://feed.entertainment.tv.theplatform.eu Path: /f/PR1GhC/mediaset-prod-all-stations	
	Cross-site scripting (reflected) Severity: Information		Host: https://api.one.accedo.tv Path: /application/logs	
			Host: https://feed.entertainment.tv.theplatform.eu Path: /f/PR1GhC/mediaset-prod-all-clips [name of an arbitrarily supplied URL parameter] /f/PR1GhC/mediaset-prod-all-clips [sort parameter] Host: https://feed.entertainment.tv.theplatform.eu Path: /f/PR1GhC/mediaset-prod-all-stations [fields parameter] /f/PR1GhC/mediaset-prod-all-stations [name of an arbitrarily supplied URL parameter] /f/PR1GhC/mediaset-prod-all-stations [range parameter]	
			Host: https://feed.entertainment.tv.theplatform.eu Path: /f/PR1GhC/mediaset-prod-all-clips [name of an arbitrarily supplied URL parameter] /f/PR1GhC/mediaset-prod-all-clips [sort parameter]	

Figura 6.20: Vulnerabilità *mediaset.it*

Sulla base dei risultati riportati possiamo osservare che tra le vulnerabilità *Cross-site Scripting*, quelle maggiormente riscontrate sono le *Cross-site Scripting Reflected*.

Nel corso di tale analisi abbiamo riscontrato problemi nella scansione di alcuni siti per i quali ad esempio, è richiesta l'autenticazione per l'accesso a determinate pagine, motivo per il quale non è stato possibile analizzare tutti gli url ottenuti. Inoltre, nel corso di alcune scansioni, durante la fase del caricamento delle pagine, mediante *Burp proxy*, alcune di queste, in seguito all'apertura prevedono alcuni controlli, tra cui vi è quello relativo alla verifica *CAPTCHA*: "Non sono un robot" per il caricamento dei siti. Il *CAPTCHA*, acronimo di "Completely Automated Public Turing test to tell Computers and Humans Apart", difatti è un test pubblico effettuato per capire se siamo umani o computer. In seguito a tale analisi, abbiamo osservato che il sito dal quale abbiamo riscontrato vulnerabilità *DOM-based XSS*, sia negli url presenti in *Visited* che in *Frontier*, dopo l'attività di scansione, è *www.mymovies.it*. Pertanto, riteniamo che in caso di un'analisi più dettagliata emergerebbero ulteriori vulnerabilità.

Capitolo 7

Conclusione e sviluppi futuri

Il presente lavoro è stato svolto al fine di effettuare *crawling* e individuare una delle vulnerabilità più diffuse sul web: il *DOM-based XSS*. Quest’ultima altro non è che un particolare tipo di vulnerabilità *XSS* che si verifica interamente nel *JavaScript* lato client e ad oggi costituisce sempre più una minaccia. Sebbene possa definirsi un attacco di tipo *Cross-site scripting*, non è possibile ricorrere a metodi tradizionali per difendersi da tale vulnerabilità, come ad esempio il rilevamento di contaminazioni lato server o i firewall delle applicazioni web, in quanto la vulnerabilità risiede interamente nel codice del client ed il server non è a conoscenza di eventuali attacchi. A tal fine, siamo partiti descrivendo in dettaglio cosa vuol dire fare *crawling* sul web, definendo questa attività come una delle più importanti svolte da un motore di ricerca. Infatti, come più volte riportato, il *crawling* è quella attività che consente di scansionare il web. Obiettivo primario di tale lavoro ha riguardato l’implementazione di un crawler il quale, preso in input un *seed*, restituisce un’insieme di url sui quali poter effettuare la nostra analisi. I *seeds* da cui siamo partiti sono quelli elencati nel *ranking* stilato da *Alexa*, ovvero i top siti più popolari. Caratteristica fondamentale del crawler sviluppato, è la possibilità di poterlo eseguire in parallelo a qualunque tool di analisi delle vulnerabilità grazie all’utilizzo di una *Console* realizzata *ad hoc*. Nel presente lavoro, il tool di analisi scelto per la ricerca di eventuali vulnerabilità è stato *Burp Suite*. Infatti, terminata la nostra attività di *crawling* mediante

il tool realizzato, abbiamo sottoposto a *Burp* gli url ottenuti con l'obiettivo di individuare principalmente vulnerabilità *DOM-based XSS*.

Tra gli sviluppi futuri si potrebbe prendere in considerazione la possibilità di verificare effettivamente che le vulnerabilità riscontrate, a seguito di un'analisi con Burp Suite, siano reali. In altre parole, si potrebbero simulare possibili attacchi al fine di verificare tali vulnerabilità. Inoltre, potrebbe essere interessante integrare all'interno del crawler sviluppato la possibilità di effettuare un'analisi delle vulnerabilità sugli url restituiti in output. In particolare, inserire la possibilità di svolgere un'analisi statica oppure dinamica a seguito di un'attività di crawling, rendendo il tool completo. Infatti, in tal caso, non solo risulterebbe indipendente da altri software di analisi, caratteristica di cui il tool già dispone, ma garantirebbe anche il possibile utilizzo dello stesso senza l'ausilio di altri strumenti.

Bibliografia

- [1] <https://www.posizionamento-seo.com/search-engine-optimization/search-engine-intelligence/funzionamento-motore-ricerca/>. Accessed: 2016.
- [2] [https://it.wikipedia.org/wiki/Bot_\(informatica\)](https://it.wikipedia.org/wiki/Bot_(informatica)). Accessed: 2020.
- [3] <https://nicholasmarmonti.com/glossario/crawler/>. Accessed: 2010.
- [4] Massimo Romano. <https://www.javaboss.it/java-crawler/>. Accessed: 2017.
- [5] <http://www.osting.it/blog/hosting-e-domini-web/web-crawler-e-spider-tutto-cio-che-ce-da-sapere/>. Accessed: 10-11-2019.
- [6] <https://support.google.com/webmasters/answer/6062596?hl=it>. Accessed: 2020.
- [7] <https://www.bytekmarketing.it/blog/come-funziona-crawler-mobile-un-sito-web/>. Accessed: 2020.
- [8] <https://www.laramind.com/blog/e-legale-lo-scraping-di-dati-in-italia-vediamo-insieme/>.
- [9] Rocco Balzamà. <https://www.roccobalzama.it/cosa-e-il-cross-site-scripting-xss-e-come-difendersi-da-un-attacco/>. Accessed: 20-08-2017.
- [10] <https://www.ideabit.com/blog/sicurezza/articolo413.htm>. Accessed: 31-10-2014.
- [11] Gianluca Brindisi. <https://www.html.it/articoli/xss-teoria-e-pratica/>. Accessed: 08-02-2012.
- [12] William Melicher. *Riding out DOMsday: Toward Detecting and Preventing DOM Cross-Site Scripting*. Accessed: 23-04-2012.
- [13] <https://hacktips.it/analizzare-sito-burp-suite/>. Accessed: 02-04-2016.

- [14] Luca Ercoli. <https://blog.seeweb.it/burp-suite-penetration-test-mobile/>. Accessed: 01-03-2016.
- [15] Giuseppe Sicari. <https://www.html.it/articoli/jsoup-parsing-semplice-di-html5-in-java/>. Accessed: 23-04-2012.
- [16] PortSwigger Ltd. <https://portswigger.net/support/installing-burp-suites-ca-certificate-in-your-browser>. Accessed:2020.
- [17] PortSwigger Ltd. <https://portswigger.net/burp/vulnerability-scanner>. Accessed: 2020.
- [18] PortSwigger Ltd. <https://portswigger.net/support/getting-started-with-burp-scanner>. Accessed: 2020.
- [19] PortSwigger Ltd. <https://portswigger.net/support/using-burp-scanner>. Accessed: 2020.
- [20] KINSTA Inc. <https://kinsta.com/it/blog/alexa-rank>. Accessed: 2020.
- [21] Alexa Inc. <https://portswigger.net/burp/vulnerability-scanner>. Accessed: 2020.