Detector automàtic de XSS

Hèctor Garcia Berzosa

01-07-2019

Resum– Aquest projecte consisteix en el desenvolupament d'una eina automàtica per a la detecció de *Cross-Site Scripts* (XSS). Aquest detector automàtic de XSS analitza les pàgines que li passi l'usuari buscant els diferents formularis amb l'objectiu d'omplir-ne els *inputs* amb diferents combinacions de caràcters per detectar si són vulnerables a una possible injecció de codi JavaScript. Després, genera un informe amb els resultats obtinguts. Alternativament, es pot utilitzar el Detector per a detectar respostes concretes quan s'envien diferents input a un formulari concret.

Paraules clau– XSS, Cross-Site Script, JavaScript, HTML, tag, web, Python, input, formulari, site, payload

Abstract– This project consist in the development of an automatic tool to detect Cross-Site Scripts (XSS). This automatic detector of XSS analyzes the pages that the user gives it searching for different forms with the objective of filling its inputs with different character combinations in order to detect if they are vulnerable to a possible injection of JavaScript code. Then, it generates a report with the results. Alternatively, the Detector can be used to detect particular responses when different input is sent to a form.

Keywords– XSS, Cross-Site Script, JavaScript, HTML, tag, web, Python, input, formulari, site, payload

1 Introducció

Aquest document és l'article del Treball de Fi de Grau Detector Automàtic de XSS. En ell es fa un petit anàlisi de l'Estat de l'Art, es descriuen els Objectius i les possibles modificacions que aquests hagin pogut tenir al llarg del desenvolupament, si la Planificació inicial que es va predir i els canvis que ha sofert, com s'ha aplicat la Metodologia o si s'ha variat aquesta, com ha evolucionat el seu Desenvolupament i quins són els Resultats que s'han generat. Finalment, es presenten unes Conclusions.

2 Estat de l'art

Les vulnerabilitats de Cross Site Scripting (XSS) consisteixen en la injecció de codi, normalment JavaScript, en llocs on això no és una funcionalitat esperada, com, per exemple, en un formulari de log-in. Un usuari maliciós pot generar un codi (*payload*), injectar-lo en una web vulnerable i aquest s'executarà en les màquines de les víctimes. Alguns dels atacs més comuns consisteixen en el robatori de cookies de sessió, "segrest" del navegador (l'atacant té el control del navegador de la víctima) o el minat de criptomonedes. N'hi ha de dos tipus: *reflected* i *stored*. En el primer cas, el *payload* només s'executa en la resposta després de fer una injecció en la petició que la genera, mentre que el segon aconsegueix que el codi maliciós quedi registrat al servidor i s'executi cada vegada que un usuari accedeix a la pàgina vulnerada. En aquest article sempre es parlarà del primer tipus.

Un tipus semblant d'atac són les SQL Injections. En aquest cas, el que es fa es injectar codi SQL per a poder manipular els bases de dades que utilitza l'aplicació, així modificar-les o obtenir informació confidencial (credencials d'altres usuaris, per exemple).

Actualment, hi ha una gran quantitat d'eines dedicades a l'escaneig automàtic de vulnerabilitats. El problema és que la majoria són eines comercials que permeten poca personalització per part de l'usuari o la dificulten i requereixen de la interacció amb un navegador o amb una interfície gràfica. També cal dir que no estan especialitzats en la detecció de XSS, sinó que són genèrics. Alguns exemples són Nessus, BurpSuite o Acunetix. El que volia fer era una eina dedicada, que es pogués executar des del terminal i que permetés a l'usuari certa personalització. El projecte més similar al meu que he trobat ha estat XSS Strike, un detector de XSS

• E-mail de contacte: hector.garciabe@e-campus.uab.cat
• Menció Realitzada: Tecnologies de la Informació
• Treball tutoritzat per: Juan Carlos Sebastián Pérez (Departament d'Enginyeria de la Informació i de les Comunicacions)
• Curs 2018/19

automàtic escrit en Python que genera els *payloads* de manera dinàmica. Està dotat d'una gran quantitat de funcionalitats i una elevada eficiència. Tot i la seva semblança amb el projecte que s'ha desenvolupat per a aquest TFG, no s'ha utilitzat XSS Strike com a referència ni inspiració, doncs ha estat descobert durant la redacció d'aquest informe.

3 OBJECTIUS

En aquest apartat es llisten i, seguidament, es defineixen els diferents objectius que es van marcar al principi del treball i els que han anat apareixent al llarg del desenvolupament, ja sigui degut a problemes que han sorgit i no s'havien plantejat durant la planificació o per noves idees sorgides en reunions amb el tutor. El Detector està desenvolupat íntegrament en Python, per tant, cal assumir que, si no s'indica el contrari, sempre que es facin referències a codi, llibreries o programació, s'estarà referint aquest llenguatge.

3.1 Llista d'objectius

- Identificació de formularis en llocs web.
- Desenvolupament d'un diccionari amb múltiples *payloads* dissenyats per a detectar XSS.
- Identificació i classificació de les injeccions que han tingut èxit.
- Generació d'un informe amb els resultats obtinguts.
- Integració d'un mòdul que gestioni les cookies de manera automàtica o semi-automàtica per a poder utilitzar el Detector en webs que exigeixin algun tipus de sessió.
- Desenvolupament de llocs web per a poder realitzar tests.
- Anàlisi de múltiples web en una sola execució.
- Possibilitat de personalitzar els *payloads* que utilitzarà el Detector.
- Generació d'un informe de resultats estèticament agradable.
- Personalització dels paràmetres utilitzats per a determinar l'èxit d'una injecció.
- Identificació i classificació d'*inputs* no pertanyents a formularis.
- Integració de suport a webs que utilitzin AJAX.
- Integració d'un mòdul d'anàlisi de codi estàtic que busqui vulnerabilitats abans de publicar la web.

3.2 Desenvolupament d'objectius

En aquest aparat una explicació més detallada de cada objectiu comentant quin és el seu estat si s'escau.

3.2.1 Identificació de formularis

El Detector és capaç d'identificar els formularis que hi ha en un lloc web i en diferencia l'acció, el mètode i els camps. També és capaç d'enviar els formularis correctament.

3.2.2 Desenvolupament d'un diccionari de XSS

El Detector és capaç d'omplir els diferents camps dels formularis de les pàgines amb diferents tags HTML i codis JavaScript per a intentar burlar els diferents filtres que les pàgines puguin tenir. S'ha generat un diccionari amb una llista amb varis d'aquests valors que el Detector pot utilitzar per a poder intentar trobar formularis vulnerables.

Actualment, el diccionari compta amb 784 entrades diferents dissenyades per a evitar diferents filtres. Tot i tenir una mida acceptable, es podria seguir ampliant però, degut a un canvi de ruta en el desenvolupament del treball, ja no tindria molt sentit.

3.2.3 Identificació i classificació d'injeccions

El Detector és capaç de llegir les respostes que generen les peticions que es realitzen quan s'envia un formulari, i d'identificar si en la resposta es troba o no el codi que s'ha intentat injectar. Seguidament, classifica quins codis han estat filtrats i quins han aconseguit una injecció amb èxit, indicant quins formularis i quins camps són vulnerables i a què ho són.

3.2.4 Generació d'un informe de resultats

El Detector genera un informe amb els resultats indicant les diferents injeccions classificades en èxits i fracassos. En el moment, es poden generar 3 tipus diferents d'informe. El primer és un informe senzill que s'imprimeix directament en la consola quan s'executa el detector, el segon és aquest mateix informe però guardat en en un arxiu de text i el tercer és un informe en HTML.

Aquest és l'últim objectiu de les *core functionlaities* del Detector que es van plantejar en un inici.

3.2.5 Integració de *cookie management*

Algunes pàgines exigeixen que l'usuari estigui registrat o tingui algun tipus de sessió per a poder interactuar amb algunes funcionalitats o formularis.

S'ha afegit una funcionalitat que permet aprofitar les entrades (com *cookies* o *tokens* de sessió) que hi ha en la base de dades local dels navegadors basats en Chromium.

Amb això es conclouen els objectius mínims marcats en un inici per a aquest treball.

3.2.6 Desenvolupament de pàgines per a testejar el Detector

S'han desenvolupat vàries pàgines web per a poder realitzar test. Aquestes pàgines tenen diferents nivells de protecció i filtres dissenyats per a poder testejar les diferents funcionalitats i mòduls del detector.

3.2.7 Anàlisi massiu de webs

Al Detector se li pot passar com a argument una llista d' URL. Aquest les analitza una a una i afegeix els resultats obtinguts a l'informe. També es pot generar un gràfic que mostri la quantitat de URL que han presentat alguna vulnerabilitat. Alternativament, també es pot passar al Detector una única URL si no es desitja utilitzar aquesta funcionalitat.

3.2.8 Personalització dels *payloads*

Al Detector se li pot passar com a argument una llista de *payloads*. D'aquesta manera l'usuari té control sobre que s'utilitzarà per omplir els formularis testejats i pot personalitzar els atacs. Alternativament, se li pot passar al detector un únic *payload*.

3.2.9 Millora de l'informe de resultats

Actualment, l'informe generat en HTML és molt senzill i no està dotat de CSS ni extreu gaires avantatges de poder ser visualitzat amb un navegador. Tot hi això, s'han utilitzat algunes etiquetes i modificacions en l'estil per a millorar l'informe inicial i fer l'informe més fàcil d'interpretar i més agradable a la vista.

3.2.10 Personalització de la detecció

El detector pot rebre com a paràmetre un valor especial que serà utilitzat per a determinar l'èxit o el fracàs de la injecció, en comptes d'utilitzar el mateix payload per a realitzar aquesta classificació.

3.2.11 Identificació i classificació d'altres *inputs*

En algunes pàgines web s'utilitzen *tags input* per a realitzar diferents funcions com, per exemple, mostrar diferents imatges depenent de les accions de l'usuari. Aquests *inputs* també poden ser vulnerables a XSS.

S'intentarà desenvolupar un mòdul que sigui capaç d'identificar aquests *inputs*, classificar-los i diferenciar-los dels que es troben en formularis.

3.2.12 Integració de suport per a AJAX

S'intentarà desenvolupar un mòdul que sigui capaç de realitzar totes les *core functionalities* en pàgines dinàmiques que duguin a terme peticions amb AJAX.

3.2.13 Integració d'un mòdul d'anàlisi de codi estàtic

S'intentarà desenvolupar un mòdul que analitzi codi font i sigui capaç de detectar potencials vulnerabilitats XSS i informar de com pal·liar-les.

4 PLANIFICACIÓ

En aquesta secció s'explica quin és el temps que es va preveure que portaria dur a terme les diferents activitats de desenvolupament del treball i quant de temps s'ha dedicat a cadascuna d'elles en realitat al finalitzar el projecte.

També es mostra el temps dedicat al desenvolupament de les tasques que han aparegut al llarg de la realització del treball. Aquestes no tenen temps inicial assignat doncs no es coneixien al començar el projecte.

Finalment, s'han eliminat de la planificació els objectius que s'han descartat.

- Llegir documentació i aprendre a utilitzar diferents eines i llibreries - 30h (39h)
  Temps invertit en llegir la documentació de diferents llibreries i aprendre a fer-les servir abans de començar a utilitzar-les en el Detector. També es suma el temps dedicat a fer programes de prova amb aquestes eines i llibreries s'hagin o no afegit finalment al Detector.
- Desenvolupar o buscar pàgines web per a realitzar tests - 40h (45h)
  Temps dedicat a muntar un o varis servidors que tenen diverses tecnologies i sistemes per a intentar evitar els XSS. També es té en compte el temps invertit a buscar diferents pàgines d'exemple per a poder imitar-ne els filtres. Finalment, també s'afegeix el temps dedicat a buscar pàgines web vulnerables que permetessin fer probes en elles.
- Desenvolupar la funció d'identificació de formularis - 10h (6h)
  Temps dedicat a investigar com poder assolir aquesta funcionalitat i programar-la, així com els canvis que s'hagin de fer per a poder integrar-la amb la resta del codi.
- Desenvolupar i integrar el diccionari de XSS - 15h (20h)
  Temps dedicat a buscar diferents *tags* HTML i codis JavaScript per a poder eludir les diferents mesures de seguretat i filtres. També es reflexa el temps dedicat a integrar aquest diccionari amb el codi.
- Identificació, classificació i generació d'informes de resultats - 25h (35h)
  Els diferents resultats positius s'han de classificar mostrant el formulari, el camp vulnerable i l'*input* que l'ha fet saltar. També es té en compte el temps dedicat a dissenyar i implementar l'informe en HTML. A més a més, s'ha tingut en compte el temps dedicat a modificar el codi existent per a poder obtenir les dades necessàries per a generar aquest informe.
- Integració de "cookie management" - 15h (40h)
  Temps dedicat a automatitzar l'ús de *cookies* per a poder realitzar les peticions a formularis que exigeixin algun tipus de sessió. En aquest temps també se li han sumat els canvis que s'han hagut de fer en el codi i el dedicat a desenvolupar alternatives que al final no s'utilitzaran en la versió final.
- Proves de validació dels mòduls i gestió d'errors - 40h (65h)
  Temps dedicat a testejar cadascuna de les funcionalitats del Detector en diferents condicions per assegurar que funcionen correctament. També es suma el temps dedicat a corregir els errors detectats.
- Generació de documentació - 60h (60h) Aquest serà el temps total dedicat a escriure els informes i l'article del TFG.
- Anàlisi massiu de webs - (12h) Temps dedicat a afegir aquesta funcionalitat i integrar-la dins el codi existent.
- Personalització de *payloads* - (2h) Temps dedicat a implementar la opció per a que l'usuari pugui escollir quins seran els *payloads* que s'utilitzaran en els test.
- Millora de l'informe de resultats - (2h) Temps dedicat a millorar l'aspecte de l'informe que es genera en HTML.

- Personalització de la detecció - (2h) Temps dedicat a afegir aquesta funcionalitat i a modificar el codi existent per a poder aplicar-la.

Inicialment el projecte estava planificat per a comportar un temps total de desenvolupament d'unes 310 hores però, finalment, s'han invertit 329 hores.

5 METODOLOGIA

Per fer aquest TFG s'està utilitzant una metodologia en espiral. Per a cada funcionalitat del Detector es determinen els seus objectius, es dissenya com integrar-la amb el codi existent (o es modifica el codi existent si escau) i es desenvolupa el mòdul. Seguidament, es dissenyen i desenvolupen diferents pàgines per a realitzar els test i aquests es duen a terme, tant de la funcionalitat en si com de la integració en el Detector. Un cop integrat el mòdul, es segueix amb el desenvolupament del següent. Després d'integrar un nou mòdul, es repeteixen tests que examinen les funcionalitats dels mòduls anteriors per a assegurar que les modificacions del codi no han afectat als apartats ja finalitzats.

En el cas de que un imprevist faci que s'hagin de fer canvis en funcionalitats que s'havien donat com a resoltes, es torna a aplicar aquesta metodologia sobre les àrees afectades repetint tot el procés altra vegada per així assegurar que no s'han generat errors nous.

S'ha d'afegir a aquesta metodologia un desenvolupament lateral per a la generació de test i documentació que no s'havia tingut en compte en la descripció inicial i que no segueix el desenvolupament espiral.

6 DESENVOLUPAMENT

En aquest apartat s'aprofundeix en com s'ha desenvolupat cadascuna de les parts del treball, quins problemes han sorgit i quines han estat les solucions trobades.

6.1 Investigació prèvia

El primer que es va fer va ser investigar i buscar diferents fonts d'informació, sobretot per aprendre quines són les mesures més típiques per a protegir-se dels XSS i per descobrir quines són les tècniques que es poden utilitzar per a evitar-les. El gruix de la informació ha estat extret de la OWASP i els seus articles dedicats als XSS[2][3]. La resta ha estat extreta de l'experiència laboral dels últims mesos i d'altres fonts[1].

6.2 Funció d'identificació de formularis

El primer mòdul que es va desenvolupar va ser l'identificador de formularis. En un primer moment utilitzaven les llibreries de Python 3 urllib.request[5] i BeautifulSoup[4], però un problema de conflicte entre versions van fer que es passés a utilitzar la llibreria requests[9].

El mòdul recorre una pàgina que a rebut com a paràmetre. Guarda cada formulari que troba en una llista. Cada formulari pren la forma d'un diccionari de Python en el que es guarden la pàgina a la que fa la petició, amb quin mètode la fa i quins són els seus *inputs*. Finalment, retorna aquesta llista.

Aquest mòdul s'executa una vegada per a cada URL analitzada.

6.3 Desenvolupament d'un diccionari

Una vegada s'ha desenvolupat el mòdul que identifica els formularis toca desenvolupar el mòdul que els omple i els envia. Però abans vaig decidir generar el diccionari per a poder testejar les *requests* amb *inputs* vàlids.

El desenvolupament del diccionari es va dividir en tres fases:

- La primera va consistir en dissenyar diferents situacions en les que un mateix *payload* podia ser entregat a un *input* concret. Per exemple, el codi `<svg/onload=alert('XSS')>` es pot entregar directament o amb una cometa i un tag davant `'</div><svg/onload=alert('XSS')>` per a "escapar" del lloc on ha d'anar.

- La segona part va ser recollir una certa varietat de *payloads* per a intentar evitar diferents tipus de filtre. Alguns exemples són `<script>alert(1)</script>`, `<BODY BACKGROUND='javascript:alert("XSS")'>` o `<IMG SRC=javascript:alert('XSS')>`.

- Finalment, es van combinar tots els elements de la segona fase amb les posicions de la primera. També es van crear algunes peticions especials[6].

La major part de les idees per a les dues primeres fases van ser extretes de la llista de la OWASP coneguda com XSS Filter Evasion Cheat Sheet[2], la resta són de casos reals d'explotació amb èxit en webs d'empreses privades realitzats en els últims mesos.

Totes les entrades es troben en un arxiu txt que es passa al programa principal on genera una llista que és recorreguda per a cada *input* de cada formulari de cada web que s'està auditant i fa una petició per a cada entrada diferent.

Després de que el diccionari fos desenvolupat, es va dur a terme un canvi de ruta en el treball que el va convertir en un extra, més que en una part imprescindible del projecte. Després d'això es van desenvolupar petits diccionaris amb característiques diferents per a poder provar les noves funcionalitats.

Amb l'estat actual del Detector el que es pretén és que l'usuari es generi ell mateix un diccionari de *payloads* que pretén utilitzar i així provar exactament les funcionalitats concretes que li interessin.

6.4 Desenvolupar les pàgines web per a realitzar tests

Un revés amb el que m'he topat durant el desenvolupament del treball és el fet de que les proves no han pogut ser realitzades contra webs reals de propietaris privats. En un primer moment vaig obtenir el permís dels meus superiors per a poder utilitzar l'eina contra els llocs web que estiguéssim auditant en aquell moment. Però, degut al gran nombre de peticions que el Detector podria arribar a llançar contra un mateix lloc web si els formularis fossin molt grans o molt quantiosos, es va decretar que es podria interpretar com un atac de denegació de servei, una prestació que normalment

no oferim. Per tant, ha calgut que desenvolupi diferents llocs web imitant filtres que m'he anat trobant.

Les web que he desenvolupat són pàgines molt senzilles amb un sol formulari amb dos camps. La major part del temps s'ha invertit en pensar els diferents tipus de filtres i testejar si els *payloads* que els eviten són o no els esperats. Les parts pertanyents al *back-end* es troben desenvolupades en PHP.

També s'ha dedicat temps a buscar pàgines web, ja estiguin *hostejades* a Internet o s'hagin de muntar en un servidor propi, que siguin vulnerables per disseny i donin permís explícit per a atacar-les. Alguns exemples d'aquest tipus de pàgines amb els que he realitzat alguns tests són DVWA[7], prompt.ml[8] o JuiceShop[10].

6.5 Desenvolupament de la identificació d'injeccions amb èxit

Aquest modul rep una URL, la llista de diccionaris que ha generat l'identificador de formularis i la llista de *payloads* que li ha proporcionat l'usuari. Per a cada *input* de cada entrada de la llista de diccionaris, s'identifiquen la pàgina de destí i el mètode i s'omple un *input* amb una entrada del diccionari de *payloads*. Seguidament, s'analitza la resposta determinant si la injecció ha tingut o no èxit per mitjà de la busca del valor introduït en el formulari dins la resposta que s'ha generat. Depenent del resultat, s'afegeix aquest valor a una llista d'èxits o de fracassos. Aquest procés es repeteix per a cada *input* i cada entrada del diccionari de *payloads*. Alternativament, aquest mòdul pot rebre un valor extra que serà utilitzat per a classificar el resultat. Quan s'utilitza aquesta funcionalitat, ja no es busca el payload utilitzat en la resposta generada pel servidor, sinó que es determina l'èxit o el fracàs de la injecció a partir de la presència o no del valor extra en aquesta resposta.

Finalment, retorna la llista diccionaris amb tots els canvis afegits.

Aquest mòdul s'executa una vegada per a cada URL analitzada.

6.6 Generació de l'informe de resultats

El Detector pot generar tres tipus d'informe. Els dos primers són molt senzills i semblats entre sí. Consisteixen en una classificació senzilla indicant la URL de destí de cada formulari auditat i quines són les entrades que han tingut èxit i quines han fracassat per a cada *input*. El primer tipus d'informe s'imprimeix directament per pantalla en el terminal en el que s'ha executat el Detector. Els resultats es van imprimint al llarg de l'execució. El segon tipus d'informe és exactament igual però guarda els resultats en un arxiu txt. Ambdós són una serie de bucles for niats que recorren la llista de diccionaris que ha generat el mòdul d'identificació d'injeccions amb èxit.

El tercer tipus d'informe és una mica més complex. Primer de tot, i abans que el Detector comenci l'anàlisi, es crea una carpeta resultats amb el fitxer que contindrà l'informe i una carpeta on hi haurà els gràfics. Després, i una vegada ha començat l'anàlisi, es va omplint el fitxer html de l'informe de manera molt similar a la generació dels dos informes anteriors, via bucles for niats que van recorrent la llista de diccionaris. D'altra banda, una funció dedicada a la generació de gràfics va dibuixant i guardant a la carpeta d'imatges els gràfics obtinguts a partir dels resultats generats en el mòdul anterior. Aquests gràfics estan generats amb matplotlib[11]. Degut a que el detector genera una sola llista de resultats per a cada URL, i aquesta no es conserva entre diferents URL, aquesta funció retorna un valor que va alimentant una variable que es manté al llarg de tota la execució perquè, més tard, una altre funció pugui generar el gràfic de resultats mostrant el percentatge de pàgines analitzades que contenen vulnerabilitats. El resultat final és un informe que es pot visualitzar amb qualsevol navegador i que mostra el percentatge de pàgines vulnerables,el percentatge de formularis vulnerables pera a cada pagina i, per a cada formulari, el percentatge de *payloads* que han sigut injectats amb èxit. Finalment, també mostra quins són aquests *payloads* que han tingut èxit i quins han fracassat.

6.7 Accedir als llocs que necessiten sessió

Una part important de les web d'Internet obliguen als usuaris a registrar-se i navegar mantenint una sessió d'algun tipus per a poder accedir a les seves funcionalitats principals. Aquesta sessió sol estar controlada mitjançant *cookies* o *tokens* que s'envien en les capçaleres de les peticions. Per tant, si es vol utilitzar el Detector en llocs que obliguen a tenir un usuari autenticat d'algun tipus, és necessari poder afegir la informació necessària a les capçaleres.

L'objectiu principal era que aquesta funcionalitat fos automàtica o semi-automàtica i suposes el menor esforç possible per a l'usuari del Detector. Per tant, volia evitar que aquest es veiés obligat a editar manualment les capçaleres de les peticions. Es va experimentar amb diferents llibreries i combinacions d'aquestes, com serien CookieJar[12] o la funció session de la llibreria requests[9]. El problema era que, a part de que no eren 100% fiables, aquestes encara requerien que l'usuari, manualment, generés la petició d'inici de sessió i n'introduís les dades. Això comportava fer molts canvis i augmentava la dificultat per a implementar l'escaneig massiu de webs considerablement.

Finalment, s'ha utilitzat pycookiecheat[13], una llibreria que permet obtenir les *cookies* de les bases de dades locals dels navegadors basats en Chromium. D'aquesta manera, l'usuari només ha de tenir una sessió vàlida iniciada en el navegador per a cada lloc web, i el Detector afegeix automàticament els valors necessaris per a aprofitar aquestes sessions en les peticions a cadascuna de les URL que li han passat. Amb això, es considera assolit aquest objectiu i també es dona per finalitzat el desenvolupament de les funcionalitats mínimes que s'esperaven del detector a l'inici del projecte.

6.8 Anàlisi massiu de webs

Durant una de les reunions de seguiment, va sorgir la idea que el Detector hauria de ser capaç de poder analitzar conjunts de webs i donar informació global sobre aquestes com, per exemple, quantes de les pàgines analitzades tenien algun formulari que presentés vulnerabilitats.

Per a assolir aquest objectiu es van fer canvis en el flux principal del programa. En un inici, el Detector demanava que l'usuari introduís manualment la URL que es desitjava analitzar una vegada iniciada l'execució. Com que ara es

pretenia que s'analitzessin grans quantitats de llocs de cop, aquesta funcionalitat ja no era viable. Ara, l'usuari ha de passar un arxiu de text amb la llista de URL com a argument en el moment d'execució. Després, s'executen tots els mòduls descrits anteriorment per a cadascuna de les web que hi hagi en aquest arxiu i es van generant els informes i els gràfics.

Alternativament, l'usuari pot passar com a argument una sola URL, així que la funcionalitat inicial del Detector no s'ha perdut.

6.9 Personalització de *payloads*

En un inici, el Detector utilitzava sempre el mateix diccionari per a introduir els valors en els diferents *inputs* dels formularis. Aquest diccionari estava en un arxiu de text expressament per tal de que fos fàcil de modificar i d'ampliar, però no es contemplava que l'usuari tingués cap motiu per interactuar-hi ni per canviar-lo.

Després de la modificació descrita en l'apartat anterior se'm va acudir que era probable que un usuari no volgués fer un anàlisi completament exhaustiu de tots els formularis d'una gran quantitat de webs, que potser només volgués comprovar si uns certs *payloads* concrets eren o no filtrats. Per aquest motiu, es va modificar altre vegada el flux principal. Ara, en comptes d'agafar el diccionari directament, el programa utilitzarà una llista de *payloads* que li proporciona l'usuari com a argument en el moment d'execució.

Alternativament, l'usuari pot passar com a argument un sol valor.

6.10 Millora de l'informe de resultats

La primera versió de l'informe en HTML que generava el Detector no estava dotada de cap norma d'estil, ni aprofitava cap de les funcionalitats característiques que els tags d'aquest llenguatge ofereix. Totes les imatges tenien la mateixa mida i tot el text era idèntic a excepció dels títols. A més a més, tot el contingut estava enganxat a l'esquerra sense deixar gens de marge. S'ha modificat l'informe per a que sigui més agradable a la vista, centrant el contingut, variant la mida de les imatges i del text per destacar-ne la importància i dividir els apartats de forma clara. També s'han afegit petites descripcions als gràfics per aclarir-ne el significat. Es podria haver desenvolupat un arxiu d'estil amb CSS per a fer més atractiu l'informe de resultats, però no es va considerar una prioritat durant el desenvolupament del projecte.

6.11 Personalització de la detecció

La idea d'aquesta funcionalitat va sorgir d'una manera similar a la de la personalització de *payloads*. En un inici, el Detector classificava una injecció com a èxit si el *payload* es trobava en la resposta que emetia el servidor. Després d'afegir la funcionalitat que permetia personalitzar els valors que s'introdueixen en els diferents *input* dels formularis, vaig decidir que l'usuari també hauria de poder decidir quins valors en la resposta es podrien considerar indicatius d'una injecció. A més a més, d'aquesta manera es poden també detectar filtres que funcionen parcialment o que modifiquen l'*input* d'alguna manera, però que segueixen sent vulnerables a *payloads* específics.

A més a més, amb aquesta funcionalitat, el Detector passa a poder utilitzar-se per a detectar no només vulnerabilitats de XSS, però també altres tipus, com SQL i, fins i tot, pot funcionar com una eina de test per a comprovar si el comportament d'un formulari en front de diferents valors és l'esperat.

Aquesta funcionalitat s'ha aconseguit permetent que l'usuari passi com a argument un valor concret que és després passat com a paràmetre al mòdul que fa les peticions. Aquest comprova si l'usuari a introduït el valor. Si és així, serà buscat en la resposta que genera el servidor després d'enviar una petició. Si no, es farà com fins ara i es buscarà el *payload* en la resposta.

7 RESULTATS

En aquest apartat es s'explica com actua el Detector en diferents situacions i els resultats que genera. Per a realitzar les proves que es mostren aquí, s'han utilitzat varies web senzilles desenvolupades per mi, una instància de la DVWA, i algunes pàgines accessibles des de Internet.

El Detector accepta diversos arguments:

- url_list (-U): Aquest argument ha de ser un arxiu de text que contingui una URL per línia. El Detector recorrerà el document i s'executarà en cadascuna de les pàgines especificades. Si no es pot accedir a alguna d'elles, s'informa a l'usuari per pantalla.

- url (-u): Aquest argument ha de ser una URL. El Detector s'executarà en aquesta URL. En el cas de que l'usuari utilitzi els arguments url i url_list en una mateixa execució, es donarà prioritat al segon. L'execució falla si l'usuari no proporciona cap dels dos.

- payload_list (-P): Aquest argument ha de ser un arxiu de text que contingui un *payload* per línia. El Detector recorrerà el document i en guardarà els continguts en una llista. Aquesta llista serà recorreguda íntegrament per a cada *input* que s'hagi d'omplir al llarg de l'execució.

- payload (-p): Aquest argument ha de ser un *string*. El Detector introduirà aquest *string* en tots els *inputs* que s'omplin al llarg de l'execució. En cas de que l'usuari utilitzi els arguments payload i payload_list en una mateixa execució, es donarà prioritat al segon. L'execució falla si l'usuari no proporciona cap dels dos.

- output (-o): Aquest argument ha de ser un *path*. L'informe de resultats en HTML es guardarà en el *path* indicat. Si l'usuari l'utilitza, la carpeta amb l'informe es guardarà en la localització des de la que s'ha executat el Detector.

- find (-f): Aquest argument ha de ser un *string*. El Detector determinarà l'èxit o el fracàs d'una injecció a partir de la presència o absència d'aquest *string* en la resposta generada pel servidor. Si l'usuari no l'utilitza el Detector farà servir el *payload* introduït per a classificar el resultat de la injecció.

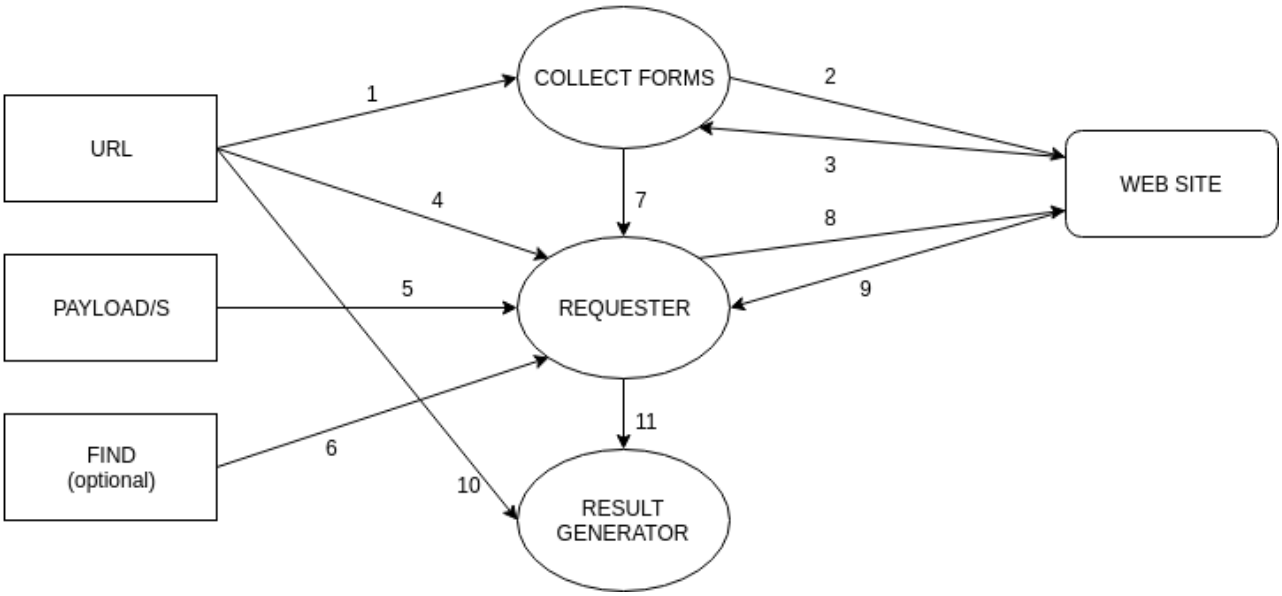

Fig. 1: Diagrama mostrant el flux d'execució de la funció main del Detector

La figura 1 mostra els flux que segueix la funció main del Detector. Totes les accions mostrades en el diagrama es repeteixen per a cada URL que l'usuari proporcioni al Detector. Seguidament s'explica cadascun dels passos que hi ha numerats en el diagrama:

1. Es passa la URL que es vol analitzar a mòdul de recol·lecció de formularis.
2. El modul de recol·lecció de formularis (collect forms) fa una petició al lloc web.
3. El lloc web respon i el mòdul de recol·lecció de formularis analitza la resposta.
4. El mòdul encarregat de fer les peticions (requester) amb les *payloads* rep la URL que es vol analitzar.
5. El requester rep el *payload* o la llista de *payloads* que s'utilitzaran en l'anàlisi.
6. El requester pot, opcionalment, rebre un paràmetre que serà utilitzat per a determinar l'èxit de les injeccions en comptes del *payload* en si.
7. Collect forms entrega a requester un diccionari amb tots els formularis que ha trobat en el lloc web
8. El requester fa una petició al lloc web omplit un *input* d'un formulari amb un *payload*. Aquest pas es repetirà *inputs* multiplicat per *payloads* vegades.
9. El lloc web respon a cada petició feta pel requester. Aquest analitza cada resposta i classifica les injeccions entre èxits i fracasos.
10. El mòdul de generació de resultats (result generator) rep la URL que s'ha analitzat.
11. El result generator rep un diccionari amb tots els formularis i els èxits i els fracassos per a cada *input*, genera unes gràfiques i afegeix la informació a l'informe final.

Ara es mostraran alguns resultats generats després de diverses execucions:

7.1 Varies URL, múltiples *payloads*, no find

En aquesta execució es van analitzar 7 pagines web amb un total de 13 formularis utilitzant un diccionari amb 7 *payloads*. El factor que determinava l'èxit o el fracàs d'una injecció és la presencia del *payload* en la resposta del servidor. De les pàgines web, 5 es troben *hostejades* en un servidor local i les altres dues a Internet. S'han triat aquestes web perque tenen gran varietat de filtres (formularis vulnerables i formularis segurs), diversos mètodes (GET i POST), diferents protocols (HTTP i HTTPS) i algunes requereixen una sessió i d'altres no. L'execució a trigat 1 minut i 7 segons.

Per a aquesta execució s'ha utilitat la comanda `python main.py -U urlList.txt -P smallDict.txt`

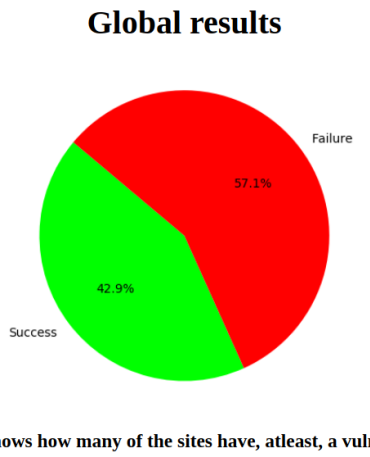

Fig. 2: Fragment de l'informe mostrant els resultats globals

En la figura 2 es pot observar que de les 7 web, 3 tenen, com a mínim, un formulari vulnerable.

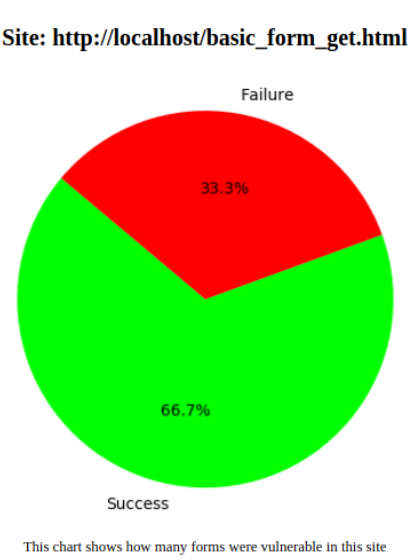

Fig. 3: Fragment de l'informe mostrant els resultats concrets d'un lloc web

En la figura 3 es pot observar que dos tersos dels formularis han tingut, com a mínim, una injecció amb èxit. Aquesta pàgina consta de tres formularis amb diverses mesures anti-XSS, però només un d'ells ha estat capaç de filtrar tots els *payloads*.

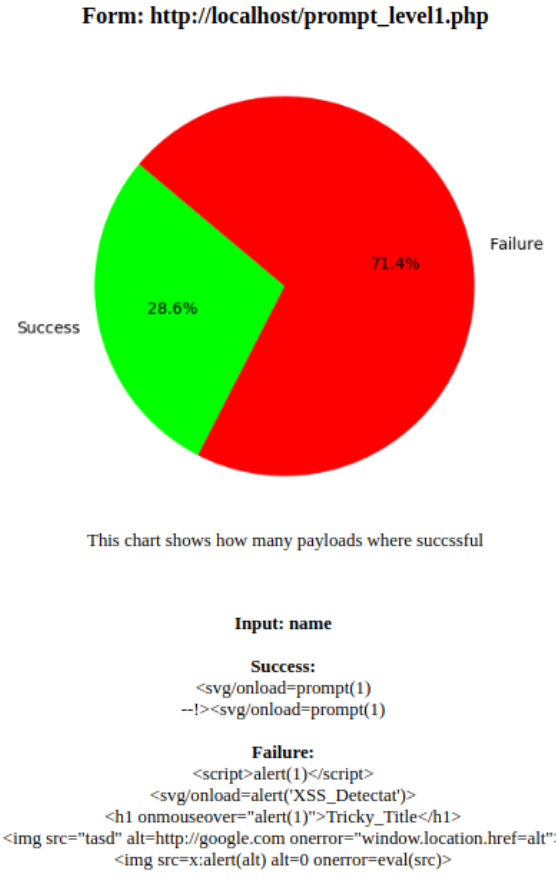

Fig. 4: Fragment de l'informe mostrant els resultats concrets d'un formulari

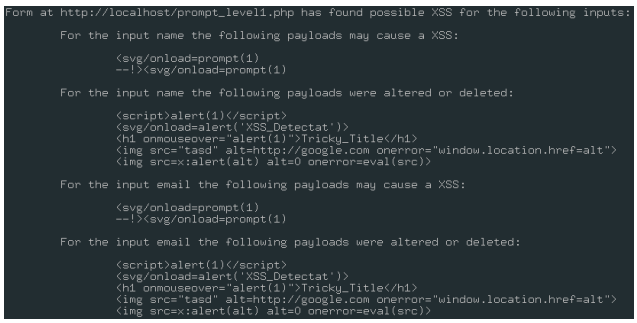

Fig. 5: Resultats de la figura 4 mostrats per pantalla

En la figura 4 es poden veure els resultats concrets d'un dels formularis del lloc del la figura 3. El gràfic mostra el percentatge de *payloads* que han estat injectats amb èxit. Sota es troben els diferents *payloads* que s'han utilitzat durant l'anàlisi, classificats entre els èxits i els fracassos.

A mode d'exemple, per a que es vegi com són els altres tipus d'informe, en la figura 5 es mostren exactament els mateixos resultats que en la figura 4 però tal i com apareixen en el terminal al llarg de l'execució del programa.

A continuació es mostren els continguts dels fitxers smallDict.txt (a dalt) i urlList.txt(a baix). Ambdós casos és una entrada per linia:

```
<script>alert(1)</script>
<svg/onload=alert('XSS_Detectat')>
<h1 onmouseover="alert(1)">Tricky_Title
    </h1>
<img src="tasd" alt=http://google.com
    onerror="window.location.href=alt">
<img src=x:alert(alt) alt=0 onerror=
    eval(src)>
<svg/onload=prompt(1)
--!><svg/onload=prompt(1)
```

```
http://localhost/basic_form_get.html
http://localhost/basic_form_post.html
https://www.w3schools.com/html/
    html_forms.asp
hello
http://hgb-pc.com/dvwa/vulnerabilities/
    xss_d/
http://hgb-pc.com/dvwa/vulnerabilities/
    xss_r/
http://hgb-pc.com/dvwa/vulnerabilities/
    xss_s/
https://www.celestrak.com/satcat/search
    .php
```

7.2 Una URL, un sol *payload*, amb find

En aquesta execució es va analitzar una sola pàgina web amb 5 formularis i utilitzant un *payload*. El factor que determinava l'èxit o el fracàs de la injecció era la presència de la cadena de caràcters "an error in your SQL syntax" en la resposta del servidor. Aquesta pàgina web es troba *hostejada* a Internet, utilitza el protocol HTTP i hi ha formularis amb metodes GET i POST.

L'execució ha trigat 10 segons. Per a aquesta execució s'ha utilitzat la comanda `python main.py -u http://www.easygosg.com/ch/index.php -p "'"-f 'an error in your SQL syntax'`

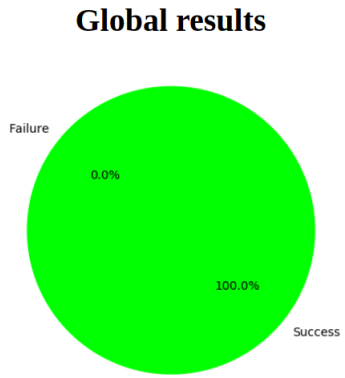

Fig. 6: Fragment de l'informe mostrant els resultats globals

En la figura 6 es veu que hi ha un 100% d'èxit. Això vol dir que la pàgina que s'ha analitzat té contingut vulnerable.

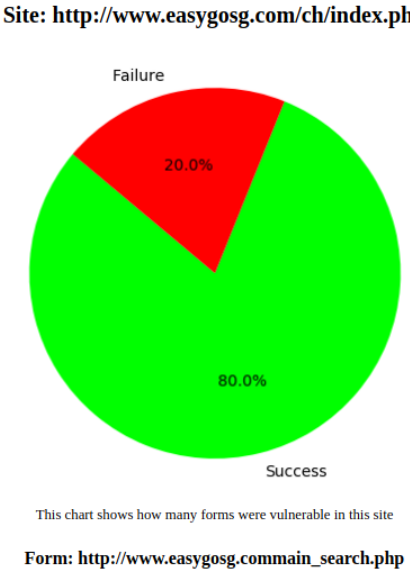

Fig. 7: Fragment de l'informe mostrant els resultats concrets del lloc web

En la figura 7 es mostra com quatre dels cinc formularis que s'han analitzat presenten, com a mínim, un *input* potencialment vulnerable a una injecció SQL.
Finalment, en la figura 8, es pot veure el resultat concret en el lloc de la figura 7. El gràfic mostra com l'únic *payload* que s'ha enviat ha tingut èxit i així es mostra en la classificació que hi ha just a sota.

Com a resum dels resultats es podria dir que ara mateix es disposa d'un detector de XSS capaç d'analitzar la susceptibilitat a XSS Reflected dels formularis de pàgines web, requereixin o no alguna mena de sessió, de manera massiva i generar un informe senzill mostrant els resultats d'aquest anàlisi. Alternativament, també es pot utilitzar per a

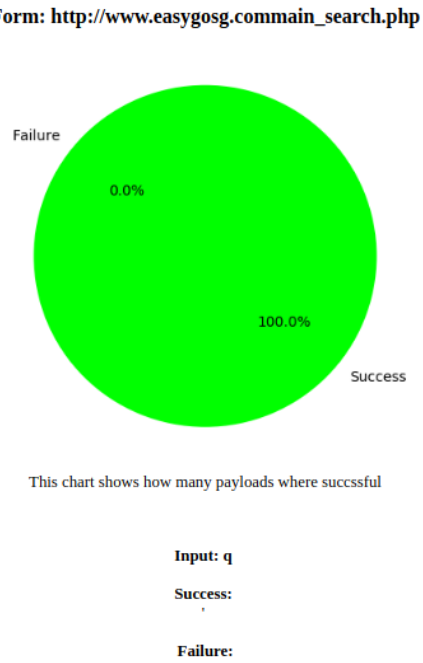

Fig. 8: Fragment de l'informe mostrant els resultats concrets del lloc web

detectar altres tipus de vulnerabilitats com, per exemple, injeccions SQL o realitzar test de comportament. En altres paraules, s'ha obtingut un producte funcional que actua de la manera esperada. Cal recordar que, com totes les eines automàtiques que hi ha al mercat, els resultats que mostra no són completament fiables i mai s'haurien de donar per bons sense una certa comprovació manual. L'aparició de falsos positius i d'error a l'hora de determinar la seguretat d'alguns *inputs* és inevitable.

8 Conclusions

L'objectiu principal del TFG s'ha aconseguit, fer un Detector de XSS automàtic funcional. Tot i això, com ja s'ha comentat en apartats previs, han hagut objectius que no s'han assolit, ja sigui per contratemps o per canvis de ruta en el desenvolupament, i hi ha hagut errors en la predicció del temps i en la planificació de tasques. Tot i això, considero el producte actual com a un producte acceptable que compleix la seva funció: ser una eina de suport per a gent que vulgui auditar la seguretat de llocs web.

Com a resum es pot dir que dels 13 objectius plantejats en la secció 3 d'aquest article, s'han assolit els 10 primers. Els altres 3 s'han descartat per que el seu desenvolupament s'hauria iniciat en dates molt pròximes a l'entrega final i no volia deixar-los incomplerts, i per que diferien bastant de la resta de funcionalitats ja implementades.

En l'estat actual, el Detector mostra moltes vies de millora. El primer de tot seria paral·lelitzar l'anàlisi de les diferents webs per a millorar el temps d'execució. També consideraria necessari millorar l'aspecte de l'informe en HTML. Caldria buscar la manera de detectar restriccions que pogués presentar el formulari a l'hora d'omplir els seus *inputs*, com serien l'opció required o el type email d'HTML. Finalment, també es podrien desenvolupar els objectius no assolits.

Segurament, al finalitzar el TFG, faré públic el repositori del Detector i en continuaré el desenvolupament encara que sigui per a poder mantenir fresques les meves habilitats com a programador.

9 AGRAÏMENTS

Al meu tutor, Juan Carlos Sebastián Pérez, per ser clar i concís en les reunions i les avaluacions dels informes. A la comunitat de developers de Python per posar a la meva disposició gran quantitat d'eines i resoldre tots els dubtes que m'hagin pogut sorgir. A Judith Campoy per llegir i corregir els errors de la documentació. I als meus companys de pis, per aguantar-me quan estava nerviós o estressat al llarg del desenvolupament del treball.

REFERÈNCIES

[1] A. Cebrián, Atacs a aplicacions web. Catalunya: Universitat Oberta de Catalunya, 2011, pp. 7-18. [Accessed: 24- Feb- 2019].

[2] OWASP Comunity, XSS Filter Evasion Cheat Sheet - OWASP, 2019. [Online]. Available: https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet [Accessed: 18- May- 2019].

[3] OWASP Comunity, Cross-site Scripting - OWASP, 2018. [Online]. Available: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)[Accessed: 13- Apr- 2019].

[4] L.Richardson, Beautiful Soup 4.4.0 Documentation, 2015. [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/[Accessed: 13- Apr- 2019].

[5] The Python Fundation, urllib.request - Extensible library for opening URLs, 2009. [Online] Available: https://docs.python.org/3.0/library/urllib.request.html[Accessed: 13- Apr- 2019].

[6] @filedescriptor, XSS Polyglot Challenge, 2018. [Online] Available: https://polyglot.innerht.ml/[Accessed:14- Apr- 2019].

[7] ethicalhack3r, DVWA - Damn Vulnerable Web Aplication, 2019. [Online] Available: http://www.dvwa.co.uk/[Accessed:6- Apr- 2019].

[8] @filedescriptor, prompt(1) to win, 2019. [Online] Available: https://prompt.ml[Accessed:25- May- 2019].

[9] A. Ronquillo, Python's Requests Library (Guide) - Real Python, 2019. [Online] Available: https://realpython.com/python-requests/[Accessed:26- May- 2019].

[10] OWASP Comunity, OWASP JuiceShop Project - OWASP, 2019. [Online] Available: https://www.owasp.org/index.php/OWASP_Juice_Shop_Project[Accessed:26- May- 2019].

[11] Pythonspot, Matplotlib Pie chart - Python Tutorial, 2019. [Online] Available: https://pythonspot.com/matplotlib-pie-chart/[Accessed:18- May- 2019].

[12] The Python Foundation, http.cookiejar — Cookie handling for HTTP clients — Python 3.7.3 documentation, 2019. [Online] Available: https://docs.python.org/3/library/http.cookiejar.html[Accessed:28- Apr- 2019].

[13] n8henrie, n8henrie/pycookiecheat: Borrow cookies from your browser's authenticated session for use in Python scripts., 2018. [Online] Available: https://github.com/n8henrie/pycookiecheat[Accessed:9- May- 2019].