KeyCrawling Dokumentation

Disclaimer: Diese Dokumentation ist ein work in progress und nicht vollständig. Bei gerne Fragen und Anmerkungen gerne an Christian Löffler wenden.

Grundlegendes:

Der Web Crawler durchsucht das Internet nach private Keys, die ungewollt öffentlich zugänglich sind, so dass dem Eviden Trustcenter diese Schlüssel als unsichere schlechte Schlüssel angegeben werden können und somit nicht mehr ausgestellt werden. Grundlegend lässt sich der Web Crawler von einer Url die HTML-Datei schicken und analysiert diese nach Links zu weiteren Webseiten. Des Weitern überprüft das Programm mit Hilfe einer Regular Expression, ob sich ein privater Schlüssel auf dieser Webseite befindet.

Vor dem Gebrauch:

Um den Web Crawler verwenden zu können, müssen folgende Dinge erledigt werden:

- Virtuelle Umgebung erstellen
- Alle benötigten Pakete instalieren (bs4, requests, cryptography)
- Starting_urls festlegen

Virtuelle Umgebung erstellen:

- Ein Verzeichnis erstellen, in dem das Programm durchgeführt werden soll
- Ein Terminal oder eine Befehlszeile öffnen.
- Zum gewünschten Verzeichnis für das Projekt navigieren.
- Den folgenden Befehl eingeben:

  ```
  python3 -m venv crawler
  ```

  Dieser Befehl erstellt ein Verzeichnis mit dem Namen "crawler", welches die virtuelle Umgebung enthält.
- Aktivieren der virtuellen Umgebung:
  Für Windows:
  ```
  venv\Scripts\activate
  ```
  Für Unix oder MacOS:

  ```
  Source venv/bin/activate
  ```

  Nachdem die virtuelle Umgebung aktiviert ist, wird der Name der virtuellen Umgebung (hier "crawler") in Ihrer Befehlszeile vor dem aktuellen Pfad angezeigt.

Alle benötigten Pakete instalieren:

- Folgende Befehle müssen im Terminal ausgeführt werden: wda
  ```
  pip install requests
  pip install beautifulsoup4
  pip install cryptography
  ```

Starting_urls festlegen:

KeyCrawling Dokumentation

- Die Text-Datei „starting_urls.txt“ erstellen oder dem Projekt hinzufügen.
- Der Datei alle Domains übergeben, welche verwendet werden sollen.
- Domains müssen im angegebenen Format vorliegen:

  https://www.youtube.com
  https://github.com
  https://stackoverflow.com
  https://www.tutorialspoint.com
  https://www.google.com

Inhalt der Implementation

Die Implementation besteht aus:

Methoden:

- crawling(base_url_object)
- scrape_page(html_in_text)
- convert_key()
- crawl_page(soup, base_url, visited_urls, urls_to_visit, new_base_urls)
- main_method()
- threat_task(threat_num, known_base_urls, threat_urls)
- if __name__ == ‘__main__’

Variablen:

- List “known_base_urls”
- int “line_count”
- List “threat_urls”
- int “num_worker”

Klasse:

- BaseUrl: speichert Referenzen zu einer base url (mit der „base url“ ist im Folgenden die Domain einer Webseite gemeint)

Methoden der Klasse BaseUrl:

- Konstruktor (def __init__(self, base))
- is_url_known(self, url)
- robot_txt_initialise(self)
- insert_url(self, url)
- is_completed(self)
- get_url(self)
- get_base_url(self)
- set_completed(self, value)
- get_count_visited_urls(self)
- update_key_count(self, found_keys)

Variablen der Klasse BaseUrl:

- String “base”
- List “url_queue”
- List “visited_urls”
- robotTxT

KeyCrawling Dokumentation

- Boolean “completed”
- Int “keys_found”

Output des Programms:

- Datei “private_key.pem”
- Ordner “Keys_found”
- Dateien mit Format f”{hash(pem)}pub.pem”

Dokumentation

Im Folgenden werden nähere Erläuterungen der Funktionen gegeben

Methoden

crawling(base_url_object):

Die Methode crawling durchläuft anhand eines übergebenen Base_url_objects alle Seiten einer Webseite.

Die Domain wird in der Variablen „base_url“ gespeichert. (Zeile 121)

Anschließend wird überprüft ob die Robot.txt Datei bereits initialisiert wurde, ist dem nicht der Fall wird die Methode robot_txt_initialise() des Objekts aufgerufen, um die Robot txt Datei zu initialisieren. (Zeile 123 – 125)

Die folgende Schleife (127 – 145) läuft solange die Funktion is_completed() False zurückgibt. Es demnach noch Urls gibt, welche nicht besucht wurden.

Innerhalb der Schleife wird eine neue Url mit Hilfe der Funktion „base_url_object.get_url()“ abgerufen und in der Variable „current_url“ gespeichert. (Zeile 129)

Im Anschluss wird mit Hilfe des Moduls „request“ eine „get Anfrage“ an die current_url gestellt und das Ergebnis wird in der Variablen „response“ gespeichert.

Ist bei dieser Abfrage kein Problem entstanden wird der Inhalt der Antwort an den html parser von BeautifulSoup übergeben und in der Variablen „soup“ gespeichert. (Zeile 138)

In Folge dessen wird die HTML Antwort zu einem String konvertiert und an die Methode scrape_page übergeben, welche prüft ob ein privater Schlüssel in dem Dokument zu finden ist und die Anzahl der konvertierten Schlüssel zurück gibt. (Zeile 140)

In Zeile 143 wird crawl_page() aufgerufen, um weitere Urls auszumachen, welche besucht werden müssen

Ist die Schleife beendet gibt die Funktion abschließend wieder wie viele Webseiten dieser Domain besucht wurden.

scrape_page(html_in_text):

Die Methode überprüft anhand einer Regulären Expression, ob ein privater Schlüssel in dem übergebenen String ist. Dieser übergebene String „html_in_text“ ist ein Html-Dokument, welches in einen String konvertiert wurde.

Das Modul „re“ bringt einige Möglichkeiten zur Analyse eines Strings mit Hilfe von Regular Expressions. Die Zeile 92 sucht nach allen Instanzen, innerhalb des „html_in_text“ welche die

KeyCrawling Dokumentation

Bedingungen erfüllen und somit ein privater RSA Schlüssel ist. Der Teil „([^*]*?)“ gibt an, dass alle Zeichen außer dem „*“ zwischen den beiden Strings ein Match ergeben und das die geringste an möglichen Zeichen ausgegeben werden soll.

Anmerkung: Zum Testen von Regular Expressions bietet sich https://regex101.com/ an.

Gibt es mindestens ein Ergebnis dieser Suche, wird jede Suche einzeln verarbeitet.

Hierzu wird der gefundene String in den Zeilen 97 – 99 so verarbeitet, dass unnötige HTML Attribute, welche noch Teil des Strings sein können entfernt werden und es werden der Präfix und Suffix hinzugefügt, welche durch die Methode re.find_all() entfernt werden.

Anschließend wird die Datei „private_key.pem“ mit dem gefundenen privaten Schlüssel überschrieben und die Methode convert_key() wird aufgerufen.

Sollte die erste Suche nicht erfolgreich sein, wird der gleiche Ablauf mit einer alternativen regular Expression in Zeile 105 – 116 durchgeführt.

Abschließend gibt die Methode die Anzahl der konvertierten Schlüssel (count) zurück.

convert_key():

Mit Hilfe des serialization tools des Moduls crypthograpy, wird ein öffentlicher Schlüssel aus der Datei „private_key.pem“ extrahiert und in dem Ordner „keys_found“ gespeichert.

Hierzu wird die Datei „private_key.pem“ als binary readable geöffnet und an die Funktion „load_pem_private_key()“ übergeben. (Zeile 70) Ist die Funktion erfolgreich, wird ein privates Schlüssel Objekt übergeben.

Die Funktion public_key() dieses Objektes gibt den zugehörigen public key in pem Schreibweise zurück. (Zeile 72 – 75)

Ist bei der Extrahierung kein Fehler aufgetreten, so wird der public key im Ordner keys_found gespeichert. Der Name dieser Datei setzt sich aus dem hash-Wert des public keys und dem String „pub.pem“ zusammen. (Zeile 81)

Die Funktion returnt entweder 1 oder 0 je nachdem ob die Konvertierung erfolgreich war oder nicht.

if __name__ == __main__:

Dies ist der Einstieg in das Programm.

Es werden der Ordner für die public_keys und beide Hash-Tabellen erstellt.

Die Hash-Tabellen werden im Anschluss mit den übergebenen Domains aus „starting_urls.txt“ gefüllt. Diese Domains stellen den Start des Crawlens dar. Vor diesen Domains aus werden weiter Urls gesucht.

Variablen

List “known_base_urls”:

Die Liste “known_base_urls” ist eine Hash-Tabelle, welche alle Ojekte der bekannten base_domains speichert und verwaltet. Der Hash wird anhand der Domain berechnet. (bsp.: https://www.example.de)

KeyCrawling Dokumentation

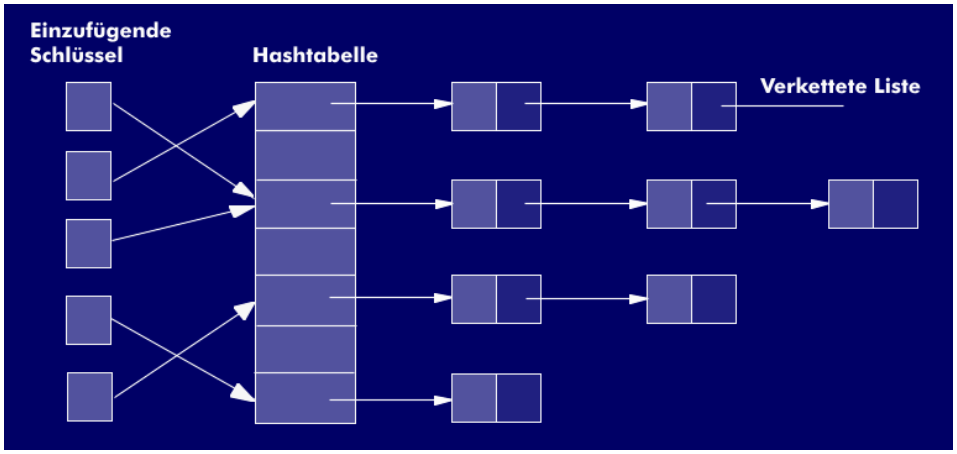

Figure 1 Beispiel einer Hash-Tabelle, welche die Objekte der base_urls verwaltet.

int “line_count”:

Der Integer “line_count” legt die Größe der Hash-Tabelle fest. Er ist standardmäßig auf 12 gesetzt, kann einfach in Zeile 167 verändert werden.

List “threat_urls”

Die Liste „threat_urls“ ist eine Hash-Tabelle, wie die „known_base_url“ und teilt den Threats die zu bearbeitenden Objekte zu.

int “num_worker”

Der Integer „num_worker“ legt die Anzahl der Threats fest. Die Anzahl ist auf 4 festgelegt, und lässt sich in Zeile 169 verändern.

Klasse BaseUrl

Die Klasse BaseUrl, dient der Bearbeitung

Methoden der Klasse BaseUrl

Output des Programms

Datei „private_key.pem“:

Die Datei ist ein Zwischenspeicher eines gefundenen privaten Schlüssels. Wird ein privater Schlüssel gefunden, werden die Werte dieses Schlüssels in diese Datei geschrieben. Mit Hilfe dieser Datei findet die Konvertierung zum zugehörigen öffentlichen Schlüssel statt, welcher gesperrt werden soll.

Wird ein neuer privater Schlüssel gefunden, so wird der Inhalt dieser Datei überschrieben, da das System lediglich die öffentlichen Schlüssel benötigt, um ein Schlüsselpaar sperren zu können.

KeyCrawling Dokumentation

Ordner „keys_found“:

In diesem Ordner werden die konvertierten öffentlichen Schlüssel gespeichert. Sollte der Ordner zum Start des Programms noch nicht existieren, wird dieser erstellt. (Zeile 163 – 165)

Dateien mit Format f”{hash(pem)}pub.pem”:

Diese Dateien sind die öffentlichen Schlüssel, welche an das System übergeben werden und innerhalb des Ordners „keys_found“ gespeichert werden. Sie sind das Produkt der Methode „convert_key()“.

Um die Namen einzigartig zu machen und Dopplungen zu erkennen, wird der Hash des Inhalts bei der Erstellung des Namens verwendet.

Anmerkung: Wird das Programm neu gestartet, ändert sich der Seed der hash-Funktion und somit wird der selben Datei bei einem Neustart ein anderer Wert zugeschrieben. Dopplungen der öffentlichen Schlüssel können demnach auftreten und lassen sich dadurch begründen.