Inhaltsverzeichnis

1 Einleitung

1.1 Ausgangslage

Eine der häufigsten Schwachstellen im Internet sind so genannte Cross-Site-Scripting Angriffe. Dabei wird versucht auf eine fremde Website Schadcode zu hinterlegen, welcher danach von anderen Benutzern unbeabsichtigt ausgeführt wird. Die Websecurity Abteilung von SAP versucht solche Schwachstellen automatisiert aufzudecken. Aus diesem Grund wurde ein modifizierter FireFox entwickelt, welcher es ermöglicht herauszufinden, welche Eingaben des Benutzers an kritischen Stellen in der Website gelangt.

1.2 Aufgabenstellung

Oft ist es Sinnvoll nicht nur eine einzige Seite auf Schwachstellen zu testen, sondern automatisiert eine Vielzahl von Seiten zu besuchen. Um dies zu ermöglichen soll im Rahmen dieser Arbeit ein Crawler entwickelt werden, der eine Liste von Startseiten besucht und aus diesen Verweise auf neue Seiten extrahiert. Dafür muss eine Architektur gefunden werden, die sich in das aktuelle System einfügen lässt und sich gut skalieren lässt.

2 Grundlagen

2.1 Crawler

Ein Web Crawler (oft auch Spider gennant) ist ein Computerprogramm, welches es ermöglicht, eine große Anzahl von Webseiten automatisiert zu besuchen.

2.2 Einsatzgebiete

Einer der bekanntesten Einsatzgebiete von Crawlern ist in Suchmaschinen. Diese bauen eine große Datenbank mit indizierten Webseiten auf und erlauben es einem Benutzer diese Datenbanken mit einer Suchanfrage zu durchsuchen. Um diese Datenbank zu füllen und aktuell zu halten wird ein Crawler verwendet, wie zum Beispiel der Googlebot der Suchmaschine Google. [https://support.google.com/webmasters/answer/182072?hl=de]

2.3 Funktionsweise

Die Funktionsweise eines Webcrawlers lässt sich im wesentlichen auf einen einfachen Algorithmus reduzieren. Im ersten Schritt wird eine Seite identifiziert, welche heruntergeladen werden soll. Alle Seiten, die noch besucht werden sollen, sind in einer Warteschlange gespeichert. Zu Beginn des Programms wird diese Schlange mit einer Reihe von Startseiten initialisiert, welche als Ausgangspunkt des Crawlprozesses dienen.
Nachdem der Schlange eine zu besuchende Seite entnommen wurde muss geprüft werden, ob das Crawlen dieser Seite überhaupt erlaubt ist. Mit der Hilfe einer robots.txt Datei oder dem HTTP Header kann festgestellt werden, ob der Webseitenbetreiber das Crawlen nicht erwünscht. Falls die Seite

besucht werden darf, wird sie komplett heruntergeladen und zu einer Liste mit besuchten Seiten hinzugefügt. Aus dem so erhaltenen HTML-Code werden nun alle Links auf der Seite extrahiert. Diese neuen Links werden der Warteschlange aus dem ersten Schritt hinzugefügt, sofern sie nicht in der Liste mit besuchten Seiten enthalten sind. Somit wird verhindert, dass eine Seite mehrmals besucht wird.
Zum Schluss wird die heruntergeladene Seite analysiert und daraus gewonnene Informationen abgespeichert. Eine Suchmaschine könnte zum Beispiel alle Wörter die auf der Seite zu finden sind speichern. Danach beginnt der Prozess von die nächste zu besuchende Seite wird bestimmt. [http://www.ijcttjournal.org/Volume13/number-3/IJCTT-V13P128.pdf]

2.4 Herausforderungen

Dieser Algorithmus erscheint zunächst recht simpel, allerdings ist das World Wide Web sehr groß, wodurch sich einige Probleme ergeben. Um eine akzeptable Performance zu erreichen, muss der Crawlvorgang hoch parallel ablaufen. In modernen Anwendungen ist es sogar üblich diesen Vorgang auf viele verschiedene Computer zu verteilen. In einem solchen Fall spricht man dann von "Distributed web crawling".
Durch die enormen Datenmengen wird es auch schwierig die Datenstrukturen des Crawlers im Arbeitsspeicher zu halten. Zum Beispiel die Liste mit bereits besuchten Seiten kann sehr schnell groß werden. Diese Daten auf ein sekundäres Speichermedium zu verlagern und dabei annehmbare Performance zu behalten ist keine leichte Aufgabe.

2.5 Cross Site Scripting

3 Architektur

3.1 Bestehende Umgebung

In diesem Abschnitt wird nun zuerst die bestehende Umgebung beschrieben. Die Abbildung XXX zeigt einen Diagramm mit den einzelnen Komponenten. Wie zu sehen ist existiert bereits ein Plugin für den Taintfox mit dem Namen "Taintnotifier". Dieses Plugin reagiert auf das taintreport Event des Taintfox. Um zu überprüfen, ob es sich bei dem gefundenen Taintflow um eine echte Schwachstelle handelt, die ausgenutzt werden kann, gibt es einen extra validation service. Das Plugin sendet diesem die potenzielle Schwachstelle per HTTP. Der Service verwendet PhantomJS, um verschiedene Angriffe auszuprobieren, ohne einen kompletten Browser nutzen zu müssen. Sobald die Validierung abgeschlossen ist werden die Ergebnisse an das Plugin zurückgeliefert. Alle Taintflows, die auf einer Seite gefunden wurden, und die entsprechenden Validierungsergebnisse werden dann in den Entwicklertools angezeigt.
Das Taintnotifier Plugin bietet außerdem die Möglichkeit einen export Server anzugeben. An diesen werden per Http alle Funde auf den besuchten Seiten gesendet. Dort werden sie in einer MongoDB gespeichert und können später über eine Weboberfläche abgerufen werden.
Dieser Aufbau erlaubt es einem Benutzer das Taintnotifier Plugin zu installieren und spezielle Seiten gezielt aufzurufen, um diese auf Schwachstellen zu testen. Wie in der Aufgabestellung bereits angesprochen, soll dieser Prozess nun automatisiert werden, damit viele Seiten in kurzer Zeit überprüft werden können.

3.2 Crawler Anforderungen

Die folgenden Anforderungen muss der Crawler erfüllen:

- Der Crawler soll horizontal skalierbar sein. Das heißt, es soll möglich sein mehrere Taintfox Instanzen zu starten, die alle an dem Crawlprozess teilnehmen. Dies soll auch möglich sein, wenn diese Instanzen auf unterschiedlichen Rechner laufen.
- Der Crawlvorgang soll gestartet werden können, indem eine Liste von URLs angegeben wird, die als Startseiten verwendet werden.
- Nachdem der Crawler gestartet wurde soll dieser automatisch alle Startseiten besuchen und aus diesen weitere URLs extrahieren, die danach besucht werden.
- Auf jeder Seite, die vom Crawler besucht wird, soll ein Analyseplugin ausgeführt werden können. Erst wenn dieses mit der Bearbeitung der Seite fertig ist, darf sie geschlossen werden.
- Das Analyseplugin kann dem Crawler zu jeder besuchten Seite ein Datenpaket mit Analyseergebnissen übergeben. Diese Daten sollen dann zentral in einer Datenbank abgelegt werden.
- Es soll die Option auf allen Seiten keine externen Links zu verfolgen. Das heißt der Crawlvorgang wird auf die Domains der Startseiten beschränkt.
- Es soll eine maximale Suchtiefe angegeben werden können. Sind alle Seiten besucht, ist der Crawlvorgang beendet.

3.3 Crawler Architektur

In diesem Kapitel wird eine Architektur für den Crawler entworfen. Dabei wird darauf geachtet, die Anforderungen aus Kapitel XXX zu erfüllen und gleichzeitig in die Bestehende Umgebung zu passen.

Anforderung XXX verlangt, dass der Crawlvorgang beschleunigt werden kann, indem mehrere Taintfox Instanzen gestarten werden, die alle am Crawlprozess teilnehmen. Dabei muss allerdings gewährleistet werden, dass keine Seite doppelt besucht wird. Die einzelnen Instanzen müssen sich also in irgendeiner Form darüber verständigen, wer für welche URLs zuständig ist. Es handelt sich also um "Distributed Web Crawling", da viele einzelne Prozesse verwendet werden, um das Crawlen zu beschleunigen. Cho and

Garcia-Molina haben in diesem Umfeld 3 Arten von verteilten Crawlern erforscht:

Independent Bei diesem Typ von Crawlern sind die einzelnen Arbeiter vollkommen unabhängig voneinander und es findet keinerlei Kommunikation statt. Jeder Arbeiter erhält zu beginn eine Liste von URLs, die er besucht, ohne sich mit anderen Arbeitern zu verständigen. Dabei kann es dazu kommen, dass es Überlappungen gibt und die gleiche Seiten von verschiedenen Arbeiter besucht wird. Dieser Ansatz ist nur sinnvoll, wenn davon ausgegangen werden kann, dass diese Überlappung insignifikant ist.

Dynamic assignment Bei diesem Aufbau gibt es zusätzlich zu den Arbeitern einen zentralen Koordinator. Dieser hält fest welche Seiten schon besucht wurden und speichert die URLs der noch unbesuchten Seiten. Die Arbeiter erhalten dann nur noch Arbeitspakete mit einer Liste von URLs, die untersucht werden sollen. Hat ein Arbeiter dies getan, schick er alle gefundenen URLs zurück. Die Arbeiter nehmen hier also eine sehr passive Rolle ein und bearbeiten nur die Anfragen des Koordinators. Es kann zusätzlich eine dynamische Lastverteilung implementiert werden. Ist ein Arbeiter beispielsweise besonders schnell, so können ihm größere Arbeitspakete zugewiesen werden. Falls nötig können sogar zur Laufzeit weitere Arbeiter hinzugefügt oder abgeschaltet werden

Problematisch ist bei diesem Ansatz nur, dass alle Anfragen über den Koordinator gehen und dieser dadurch schnell zum Flaschenhals der Anwendung wird. Das System kann dann nicht mehr über die Anzahl der Arbeiter horizontal skaliert werden, da der Koordinator nicht schnell genug neue Arbeitspakte verteilen kann.

Static assignment Hier wird kein zentraler Server benötigt, da es eine feste Regel gibt, die jede URL einem Arbeiter zuweist. Es kann zum Beispiel eine Hash-Funktion verwendet werden, um festzustellen von welchem Arbeiter eine URL verarbeitet werden soll. Da es passieren kann, dass ein Arbeiter eine URL findet, die einem anderen Arbeiter zugewiesen ist, muss es eine Möglichkeit die URLs zwischen den Arbeitern auszutauschen.

Im Rahmen dieses Projektes wurde sich für einen verteilten Crawler mit dynamischer Zuweisung (dynamic assignment) entschieden. Komplett unabhängige Arbeiter kommen nicht in Frage, da die Überlappung zu hoch ist. Eine statische Zuweisung ist schwer umzusetzen, da die einzelne Arbeiter

dafür untereinander kommunizieren müssten.
Bei der dynamischen Zuweisung kann der Koordinator zwar zum Flaschenhals werden, allerdings ist dies unwahrscheinlich, da die Analyse einer Seite oft mehrere Sekunden dauert. Dadurch brauchen die Arbeiter vergleichsweise lange, um ein Arbeitspaket zu bearbeiten und der Koordinator wird nicht überlastet.
Die Aufteilung in einen Koordinator-Server und vielen Arbeitern ist außerdem Hilfreich um die Anforderung XXX aus Kapitel XX umzusetzen. Dabei ging es darum die Analyseergebnisse von jeder Seite an einer zentralen Stelle zu speichern. Die einzelnen Arbeiter können dafür einfach die Daten an den Koordinator schicken, welcher sie dann in einer Datenbank ablegt.

4 Implementierung

4.1 Worker Plugin

4.2 Master Plugin

5 Anwendung

6 Fazit

Listings

Abbildungsverzeichnis

Tabellenverzeichnis