Aktuelle Probleme und Potentiale von Web-App Scannern

Markus Schneck[1,2] · Markus Ullmann[2] · Kerstin Lemke-Rust[2]

[1]Fraunhofer SIT
markus.schneck@sit.fraunhofer.de

[2]Hochschule Bonn-Rhein-Sieg
markus.schneck@smail.inf.h-brs.de
{markus.ullmann | kerstin.lemke-rust}@hochschule-bonn-rhein-sieg.de

Zusammenfassung

Web-Applikationen genießen aufgrund ihrer Plattformunabhängigkeit eine große Verbreitung. Diese stellen vielfach das Frontend einer IT-Infrastruktur dar. Aufgrund dessen, spielt die Bedeutung der Informationssicherheit bei Web-Applikationen eine große Rolle. Aufgrund der Quellcodegröße der Web-Applikationen ist eine manuelle Schwachstellenanalyse nicht mehr praktikabel. Aus diesem Umstand sind Web-Application Vulnerability Scanner entstanden, um Web-Applikationen in ihrem Verhalten zu untersuchen und Schwachstellen in Web-Applikationen aufzudecken. In dieser Arbeit werden die Ergebnisse einer Überprüfung der Erkennungsleistung von aktuellen Open Source Web-Application Vulnerability Scannern aus dem Jahr 2015 dargelegt. Hierfür wurde eine Analyseinfrastruktur geschaffen, die eine feingranulare Leistungsfeststellung ermöglicht. Darüber hinaus werden konkrete Optimierungspotentiale für frei erhältliche Web-Application Vulnerability Scanner aufgezeigt, die die Erkennungsleistung der Scanner signifikant steigern können.

1 Einleitung

Bereits seit vielen Jahren ist es üblich, Softwaretools zu nutzen, um Schwachstellen und Sicherheitslücken in informationstechnischen Systemen automatisch zu erkennen. Doch nach wie vor gibt es nur schwer erkennbare Sicherheitslücken und mit jeder technologischen Weiterentwicklung besteht auch latent die Gefahr, neue Angriffsmöglichkeiten auf IT-Systeme und Anwendungen zu schaffen.

Eine Technologieklasse, die in dieser Arbeit behandelt wird, sind Web-Applikationen. Heutzutage werden diese täglich dazu genutzt, um Nachrichten zu verschicken, Banküberweisungen zu tätigen, Bestellvorgänge zu initiieren oder um Einzukaufen. Ausnutzbare Schwachstellen in diesen Web-Applikationen können ernsthafte Folgen für die Benutzer und Betreiber der Web-Applikation haben.

Da Web-Applikationen andere Kommunikationswege benutzen als herkömmliche clientbasierte Software, wurden sogenannte Web-Application Vulnerability Scanner entwickelt, im Folgenden Web-App Scanner genannt. Mit deren Hilfe können die Funktionsweise und potentielle Schwachstellen dieser Art von Applikationen analysiert werden. Die im Rahmen einer

P. Schartner · K. Lemke-Rust · M. Ullmann (Hrsg.) · D•A•CH Security 2015 · syssec (2015) 323-332.

Schwachstellenanalyse einer Web-Applikation detektierten Schwachstellen können im nächsten Schritt durch entsprechende Softwareanpassungen der Web-Applikation geschlossen werden. Auf dem Markt ist eine Reihe von Web Vulnerability Scannern verfügbar. Über ihre aktuelle Erkennungsleistung ist nur wenig bekannt.

Um Web-Applikationen auf Schwachstellen zu untersuchen ist es wichtig, den Erkennungsprozess von Sicherheitslücken gleichermaßen effizient und effektiv zu gestalten. Eine pragmatische Möglichkeit die Effektivität dieses Prozesses zu steigern, liegt darin, mehrere Web Application Vulnerability Scanner mit verschiedenen Analyseschwerpunkten auf dieselbe Zielsoftware anzuwenden, um so zu einer verbesserten Erkennungsleistung zu kommen. Allerdings ist vollkommen unklar, welchen Erkennungsgewinn eine entsprechende Kombination bringt. Die Ergebnisse unserer Arbeit geben hier Aufschluss. In dieser Arbeit wird aufgezeigt, dass eine Reihe von Schwachstellenkategorien von keinem genutzten Web Application Vulnerability Scanner erkannt wird. Daher werden im nächsten Schritt die Eigenschaften dieser Schwachstellen genauer betrachtet, um konkrete Verbesserungspotentiale für Web Application Vulnerability Scanner aufzuzeigen. Für die Untersuchung der Erkennungsleistung aktueller frei erhältlicher Web Application Vulnerability Scanner wurde eine flexible Analyseumgebung konzipiert und realisiert, die im weiteren Verlauf dieses Papier näher dargelegt wird.

2 Stand der Forschung

2.1 “Why Johnny Can’t Pentest”

In der Veröffentlichung von Adam Doupé, Marco Cova und Giovanni Vigna [DoCV10] mit dem Titel "Why Johnny Can’t Pentest: An Analysis of Black-box Web Vulnerability Scanners“ aus dem Jahr 2010 wurden elf verschiedene Web-App Scanner auf ihre Erkennungsleistung untersucht. Ein wichtiger Aspekt dabei ist, dass solche Scanner oftmals als „point-and-click pentesting“ Werkzeuge angepriesen werden, die Autoren jedoch zeigen können, dass manche Schwachstellen nur mit gezielter Konfiguration und dem notwendigen Know-How erkannt werden können. Bei den verwendeten Scannern handelt es sich sowohl um frei erhältliche Scanner, als auch um kostenpflichtige Produkte wie beispielsweise Acunetix, AppScan oder Burp.

Damit die Scanner evaluiert werden können, wurde eine Web-Applikation geschrieben. Diese entspricht einer realen Anwendung und beinhaltet Schwachstellen, die heutzutage häufig in Web-Applikationen zu finden sind. Zur Erstellung der Web-Applikation wurden allerhand Technologien verwendet, um ein möglichst breites Spektrum der Scanner zu untersuchen. Zu diesen Technologien zählen komplexe HTML Formulare, JavaScript, Flash Code und dynamisch generierte Seiten. Um die notwendigen Schwachstellen festzulegen, wurde auf die OWASP Top Ten Liste aus dem Jahr 2010 zurückgegriffen. So konnten populäre Schwachstellenkategorien mitsamt ihren Schwachstellen ausfindig gemacht werden.

Doupé et al. gelang es zudem zu zeigen, dass von den verwendeten Scannern keiner durch alleinige Verwendung mehr als 40% der vorhandenen Schwachstellen ausfindig machen konnte.

2.2 “State of the Art”

In der Veröffentlichung “State of the Art: Automated Black-Box Web Application Vulnerability Testing“ von Jason Bau, Elie Burzstein, Divij Gupa und John Mitchell von der Stanford University [BBGM10] werden acht Web-App Scanner von führenden Unternehmen auf dem

Gebiet genutzt. Zu diesen Unternehmen zählen beispielsweise Acunetix, HP, IBM und McAfee. Neben bekannten öffentlichen Web-Applikationen wie beispielsweise phpBB2 oder Wordpress, wurde auch im Rahmen dieses Projektes eine eigene Applikation geschrieben, um die Leistung der Scanner zu untersuchen.

Bau et al. haben gezeigt, dass die Erkennungsleistung der eingesetzten Web-App Scanner je nach Schwachstellenkategorie schwankt. Darüber hinaus bleiben 46% der vorhandenen Schwachstellen von den Web-App Scannern unentdeckt. Auch bei der Verwendung der selbst geschriebenen Web-Applikation zeigen sich deutliche Leistungsunterschiede bei der Erkennung von Schwachstellen verschiedener Kategorien.

Somit wird die Aussage von Doupé et al. ebenfalls bestätigt, ohne weitere Kenntnisse auf diesem Gebiet lassen sich die Ergebnisse nicht vollständig und korrekt interpretieren. Ein sich wiederholendes Ergebnis ist ebenfalls, dass kein Web-App Scanner alle Schwachstellen ausfindig machen kann, es lediglich einzelne Kategorien gibt, in denen bestimmte Scanner sehr gut abschneiden. Ebenfalls wurde gezeigt, dass populäre Schwachstellen, wie beispielsweise „XSS“, „SQL Injections“ oder „Information Leakage“ häufiger entdeckt werden, als nicht so populäre Schwachstellen. Darunter fallen beispielsweise „second order vulnerabilites“ oder Schwachstellen in Zusammenhang mit dynamischen Inhalten.

2.3 Abgrenzung der eigenen Arbeit

Um über die Ergebnisse bisheriger Veröffentlichungen hinaus zu gehen, wird neben einer verwundbaren Web-Applikation auch eine Infrastruktur entwickelt, mit deren Hilfe nicht nur Scanner evaluiert werden können, sondern auch eine gezielte Analyse der Ergebnisse stattfinden kann. Eine solche Analyse ermöglicht es beispielsweise die Kommunikation zwischen Scannern und Web-Server zu nutzen, um die Ursache für eine schlechte Erkennungsleistung ausfindig zu machen. Durch dieses Vorgehen sollen mögliche Verbesserungspotentiale von Web-Application Vulnerability Scannern aufgezeigt werden.

3 Schwachstellen in Web-Applikationen

Die neuste Version der OWASP Top Ten [Owas13] aus dem Jahr 2013 beschreibt aktuelle Schwachstellen in Web-Applikationen. Die folgende Auflistung zeigt die von OWASP detailliert beschriebenen Schwachstellen im Überblick:

1. Injections
2. Broken Authentication and Session Management
3. Cross-Site Scripting (XSS)
4. Insecure Direct Object References
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing Function Level Access Control
8. Cross-Site Request Forgery
9. Using Components with Known Vulnerabilities
10. Unvalidated Redirects and Forwards

Für die spätere Analyse der Erkennungsleistung werden verschiedene Schwachstellen ausgewählt, darunter Kategorie 2 „Broken Authentication and Session Management“ bzw. Kategorie 3 „Cross-Site Scripting“.

Zu Schwachstellen der Kategorie 2 gehört beispielsweise das Preisgeben einer Session-ID innerhalb der URL. Bei einer Implementierung mittels HTTP-GET kann die Session-ID sichtbar übertragen werden und bei Weitergabe der URL auch unwissentlich durch Dritte genutzt werden. Ein mögliches Szenario ist die Verwendung einer Buchungsfunktion. Ein eingeloggter Benutzer begutachtet ein Angebot und möchte dieses einem Bekannten zukommen lassen. Was dem Benutzer beim Verwenden der URL nicht auffällt, ist, dass seine Session-ID in der URL enthalten ist. Der Bekannte, welcher nicht auf der Seite eingeloggt ist, erhält die URL und handelt je nach Implementierung der Web-Applikation mit einer fremden Session-ID.

Neben unbemerkter falscher Verwendung existieren natürlich auch Schwachstellen die gezielt ausgenutzt werden können. Darunter beispielsweise „Cross-Site Scripting“-Schwachstellen der Kategorie 3. Diese existieren in verschiedenen Versionen, beispielsweise in Verbindung mit anderen Technologien wie JavaScript oder Flash, oder auch als persistente oder mehrschrittige Schwachstelle. Jedoch besitzen alle Cross-Site Scripting Schwachstellen die gleiche Ursache, eine fehlende Validierung der Benutzereingaben. Wird ein Benutzer zu einer Eingabe aufgefordert und missbraucht diese beispielsweise, um Skript-Code einzuschleusen, so kann dieser je nach Implementierung bei der Ausgabe zur Ausführung gebracht werden. Um dies zu verhindern, sollten Benutzereingaben validiert werden und die für Skripte benötigten Sonderzeichen entfernt werden.

Die verschiedenen Versionen dieser Schwachstelle unterscheiden sich meist nur in der Implementierung und Funktion der Benutzereingabe. Wird beispielsweise ein Formular mittels spezieller Technologien wie z.B. JavaScript oder Flash erzeugt um Daten eines Benutzers entgegen zu nehmen, so kann die Verwendung dieser Technologie dazu führen, dass die Schwachstelle schlechter erkannt wird.

Auch die Verwendung einer Weiterleitung zwischen einer Benutzereingabe und der Ausgabe dieser unvalidierten Daten kann problematisch in der Erkennung einer Cross-Site Scripting Schwachstelle sein. Verantwortlich dafür ist, dass die Ein- und Ausgabe einander nicht zugeordnet werden kann und eine solche Schwachstelle dadurch nicht automatisiert nachgewiesen werden kann.

Diese und weitere Schwachstellen sowie die Verhaltensweise von Web-App Scannern können durch eine entsprechende Analyseumgebung näher betrachtet werden.

4 Entwicklung und Implementierung

4.1 Verwundbare Web-Applikation

Die Web-Applikation hat nicht den Anspruch eine reale Applikation darzustellen, vielmehr sollen populäre und aktuelle Schwachstellen klar deklariert werden und durch eine klare Struktur leicht zu erweitern sein. Dabei sollen die Schwachstellen nachvollziehbar bleiben, um die Ergebnisse der Web-App Scanner mit wenig Aufwand zu evaluieren und auch False-Positives mit wenigen Blicken zu deklarieren.

Um bei der Wahl von Technologien und Erweiterungen der Web-Applikation bereits während des Entwicklungsprozesses flexibel zu bleiben, wird auf bisherige verwundbare Webapplikationen verzichtet. Darüber hinaus wird auf diesem Wege die volle Kontrolle über den Quellcode behalten, ohne mit fremden Lizenzen in Kontakt zu geraten. Aus diesem Grund wird bei der Entwicklung ebenfalls auf bereits existierende Bibliotheken verzichtet, sofern diese nicht bereits im jeweiligen Sprach-Standard enthalten sind.

Populäre und aktuelle Schwachstellen wurden in Verbindung verschiedener Technologien wie beispielsweise PHP, JavaScript und Flash implementiert. 25 Schwachstellen aus verschiedenen Kategorien wie beispielsweise XSS, SQL Injections, Command Line Injections, Security Misconfigurations und Sitzungs-Management wurden implementiert. Der modulare Aufbau der Web-Applikation bietet Erweiterbarkeit sowohl hinsichtlich der Schwachstellenkategorien, als auch der verwendeten Technologien.

4.1.1 Beispielhafte Implementierung einer Schwachstelle

Anhand der „Cross-Site Scripting Multistep“-Schwachstelle wird beispielhaft gezeigt, wie Schwachstellen in der verwundbaren Web-Applikation implementiert werden. Nachdem die Schwachstelle ausgewählt ist, werden essentielle Eigenschaften festgehalten, die eine Verwundbarkeit ausmachen.

Eine XSS-Schwachstelle liegt dann vor, wenn eine Benutzereingabe ungeprüft und ungefiltert ausgegeben wird und dadurch die Ausführung von Skripten ermöglicht wird. So beispielsweise, wenn ein Benutzer JavaScript-Code übergibt und die Web-Applikation diesen ungeprüft auf dem Bildschirm ausgeben möchte. Bei der Ausgabe interpretiert der Browser den Code und führt diesen aus.

Der Unterschied zwischen einer herkömmlichen XSS-Schwachstelle und der Multistep-Variante ist, dass die Benutzereingabe temporär gespeichert wird, bevor diese ungeprüft ausgegeben wird. Anders als bei einer persistenten XSS-Schwachstelle werden die Daten beispielsweise nicht in einer Datenbank abgelegt, wo sie dauerhaft gespeichert werden können. Für die Realisierung dieser Schwachstelle wird eine nicht persistente Speichermöglichkeit benötigt. Eine davon ist eine Sitzungs-Variable. Diese speichert Daten solange in einer Variablen, wie auch die Sitzung aktiv ist.

Der Quellcode dieser Schwachstelle ist funktionell in drei Teile gegliedert. Die erste Datei ist eine HTML-Seite, welche die Benutzereingabe beispielsweise über ein Formular entgegennimmt und an ein PHP-Skript sendet.

```
<form action="chain.php" method="get">
  <input type="text" name="comment" value="">
  <input type="submit" name="submit" value="Submit">
</form>
```

Die zweite Datei empfängt die Dateien und speichert sie ungeprüft in einer Sitzungs-Variablen. Dazu wird zunächst eine Sitzung etabliert und daraufhin eine neue Variable angelegt und mit Inhalt befüllt. Nachdem dieser Vorgang abgeschlossen ist, folgt eine Weiterleitung zu einem Ausgabeskript.

```
<?php
session_start();
$_SESSION["input"] = $_GET["comment"];
```

```
echo '<a href="xss_simple_test.php">Redirect</a>';
?>
```

Dieses Ausgabeskript gibt den Inhalt der festgelegten Sitzungs-Variablen aus. Sofern JavaScript-Code enthalten ist, wird der Browser diesen zur Ausführung bringen.

```
<?php
session_start();
echo "Output: ";
echo $_SESSION["input"];
?>
```

4.2 Analyseumgebung

Die Analyseumgebung basiert auf einem LAMP-Stack. Dafür wird ein LAMP-Stack mit aktuellen Versionen der Komponenten virtualisiert. Die Web-App Scanner werden getrennt voneinander auf je einer virtuellen Maschine betrieben, um mögliche Konflikte in etwaigen Abhängigkeiten zu unterbinden.

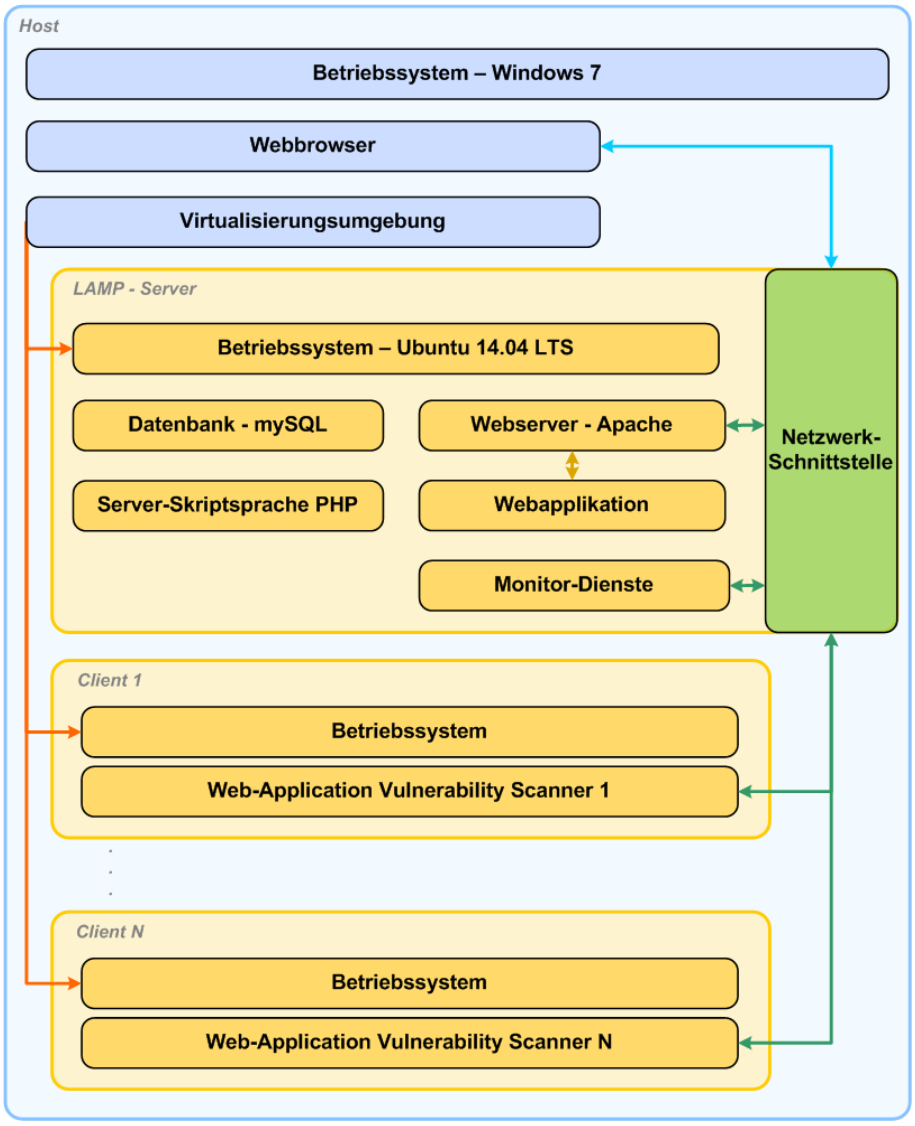

Abb. 1: Aufbau der Infrastruktur

Die ausschlaggebenden Funktionen der Infrastruktur sind das Hosten von unsicheren Web-Applikationen und die kontrollierte Anwendung von Web-App Scanner sowie das Monitoring der Kommunikation zu weiteren Analysezwecken. Dank modularem Aufbau können weitere Web-App Scanner und andere Werkzeuge der Infrastruktur hinzugefügt werden.

4.3 Web-App Scanner

Die verwendete Version der jeweiligen Web-App Scanner-Software, sowie die verwendeten Profile werden in folgender Auflistung festgehalten.

- Arachni v0.4.7 – 0.4.4, Default Scan (Full Audit)
- Skipfish v1.01b, Full Audit
- OWASP ZAP v2.3.0.1, Full Audit
- W3af v1.6.0.2, Full Scan
- Vega v1.0 build devel-108, Full Audit
- Wapiti lokaler Build der Kali Linux Distribution v1.0.6, Default Scan (Full Audit)

5 Ergebnisse

Die Verwendung der Web-App Scanner hat gezeigt, dass kein Web-App Scanner mehr als 12 der 25 vorhandenen Schwachstellen ausfindig machen kann.

Tab. 1: Ergebnisse Web-App Scanner gegenüber manuellem Penetrationstest.

	XSS GET	XSS POST	XSS MS	XSS st.	XSS JS	XSS Fl.	SQLI
w3af	X	X					X
Arachni	X						X
Skipfish	X						X
OWASP ZAP							X
Vega	X	X					X
Wapiti	X	X		X			
Manuell	X	X	X	X	X	X	X

	SQLI JS	CLI	RFI	LFI	SM	PT	CSRF	Admin/PW	DB ct	HTTP ct
w3af		X	X	X		X				
Arachni		X	X	X		X				
Skipfish	X									
OWASP ZAP	X	X			X	X				
Vega		X	X	X	X	X				2X
Wapiti		X	X							
Manuell	X	X	X	X	X	X	X	X		2X

	FB	No AL	SESSION-ID	HTTP Flags	UR	Clickjacking	Mflac
w3af					X	X	
Arachni				X	X		
Skipfish	X						
OWASP ZAP				X		X	
Vega				X		X	
Wapiti							
Manuell		X	X	X	X	X	X

Abkürzungsverzeichnis zu Tabelle 1

XSS GET	XSS reflected GET	XSS POST	XSS reflected POST
XSS MS	XSS reflected Multistep	XSS st.	XSS stored
XSS JS	XSS JavaScript	XSS Fl.	XSS Flash
SQLI	SQL Injection	SQLI JS	SQL Injection JavaScript
CLI	Command Line Injection	RFI	Remote File Inclusion
LFI	Local File Inclusion	SM	Security Misconfiguration
PT	Path traversal	CSRF	Cross-Site Request Forgery
Admin/PW	Weak Admin-Account / Password Combination	DB ct	Sensitive Data Protection Vulnerability
HTTP ct	Sensitive Data Protection Vulnerability: cleartext over http	FB	Forceful Browsing
No AL	Session Management: Logout	SESSION-ID	Session Management: SESSION-ID in URL
HTTP Flags	Session Management: HTTP Flags	UR	Unvalidated Redirect
Mflac	Missing function level access control		

Kombiniert man die Ergebnisse der verschiedenen Scanner, so werden 16 Schwachstellen gefunden. Einzelheiten der Erkennungsleistung der geprüften Web-App Scanner sind in Abbildung 2 dargestellt. Unter den neun nicht gefundenen Schwachstellen sind beispielsweise „Missing function level access control“, verschiedene XSS Schwachstellen, Session-ID in URL, nicht vorhandener automatischer Logout und ein vorhandener Administrator-Account mit vorhersehbarer Namens-Passwort-Kombination.

Tabelle 1 zeigt detaillierte Ergebnisse der sechs verwendeten Web-App Scanner gegenüber einem manuellen Penetrationstest. Das zugehörige Abkürzungsverzeichnis befindet sich im Anhang.

Für weitere Untersuchungen wurden drei Schwachstellenkategorien ausgewählt, die von keinem verwendeten Web-App Scanner gefunden wurden. Die Wahl der Schwachstellen fiel auf die mehrschrittige XSS (XSS MS), XSS in Verbindung mit JavaScript (XSS JS) und der Session-ID in URL (SESSION-ID). Gezielte Analysen der Schwachstellen und der Kommunikation zwischen Server und Web-App Scanner konnten einige Problemstellen in der Erkennung aufzeigen.

Für diese Schwachstellen-Kategorien wurden mögliche Verbesserungen in der Erkennung festgehalten und Konzepte entwickelt, die vorhandenes Potential realisieren könnten. So konnte eine Methode entwickelt werden, die vorhandene XSS Schwachstelle in Verbindung mit JavaScript ausfindig zu macht. Dafür wurde der Datenverkehr mittels Wireshark und WebScarab analysiert und Unterschiede zu herkömmlichen XSS Schwachstellen untersucht. Anhand der erhaltenen Daten wurde festgestellt, dass ein Pattern Matching ein HTML-Formular identifizieren kann, wenn dieses wie in der implementierten Schwachstelle realisiert wurde. An die Zieladresse des so erkannten Formulars können die üblichen Abfragen geschickt werden, um eine XSS Schwachstelle zu erkennen. Die Problematik, dass die Schwachstelle sich innerhalb eines JavaScript-Blocks befindet, wurde durch Wiedererkennung der vom Benutzer eingegebenen Zeichenkette gelöst.

Auch für die anderen Schwachstellen wurden auf diese Art Lösungskonzepte entwickelt, um die Erkennungsleistung von Web-App Scannern auf lange Sicht zu steigern.

5.1 Erkennung von Verbesserungspotential

Anhand der bereits während der Implementierung vorgestellten XSS Multistep-Schwachstelle soll nun gezeigt werden wie mögliches Verbesserungspotential aufgedeckt werden kann.

Durch die Verwendung eines Proxys lässt sich der Datenverkehr zwischen der Web-Applikation und einem Web-App Scanner mitlesen. Durch diese Daten lässt sich das Verhalten von Web-App Scannern erkennen, ohne dafür den Quellcode einzusehen. Darüber hinaus stellt beispielsweise Arachni Informationen bereit, die zur Erkennung einer Schwachstelle genutzt wurden. Da zwischen der nicht erkannten XSS Multistep-Schwachstelle und einer leicht von Web-App Scannern erkannten herkömmlichen XSS Schwachstelle keine großen Unterschiede herrschen, wird zunächst das Vorgehen bei einer herkömmlichen XSS Schwachstelle betrachtet.

Method	Resource	Parameters	
GET	http://192.168.16.143 /xss/xss_simple_test.php	comment	1--%3E %3Csome_dangerous_input_5feceb66ffc86f38d952786c6d696c79c2dbc239dd4e91b46729d %2F%3E%3C%21--

Abb. 2: Eingabewerte zur Erkennung von XSS-Schwachstellen

Neben den von einem Proxy mitgeschnittenen Datenverkehr, bieten Web-App Scanner auch Informationen über die genutzten Daten zur Schwachstellenerkennung an. So zeigt beispielsweise Arachni deutlich, dass es eine einfache XSS-Schwachstelle durch ein Pattern Matching ausfindig macht.

Response

Proof is highlighted in red and scroll-centered.

Render the response

```
Output: 1--><some_dangerous_input_5feceb66ffc86f38d952786c6d696c79c2dbc239dd4e91b46729d73a27fb57e9/><!--<br><a
href="../index.html">Back to Main</a>
```

Abb. 3: Erfolgreiches Pattern Matching zur Erkennung einer XSS-Schwachstelle

Die Problemstellung bei einer XSS Multistep-Schwachstelle liegt nun darin, dass zwischen dem Speichern der Benutzereingabe und der ungeprüften Ausgabe, mindestens eine Weiterleitung genutzt wird. Die Zuordnung zwischen Ein- und Ausgabe, welche bei einem Pattern Matching unabdingbar ist, kann nicht hergestellt werden.

Ein Lösungsansatz, um solche Zuordnungen herzustellen, sind eindeutige Identifikationsnummern. Jede Benutzereingabe, die ein Web-App Scanner simuliert, sollte mit einer eindeutig zu identifizierenden Nummer versehen werden. So kann unabhängig von Codefluss und Applikations-Aufbau eine Zuordnung zwischen Ein- und Ausgabe hergestellt werden.

Neben einer solchen ID beinhaltet eine Benutzereingabe die herkömmlichen Zeichen zur Identifizierung von XSS-Schwachstellen. Eine mögliche Benutzereingabe könnte wie folgt konstruiert werden:

„<script>alert(„ID:123456789“)</script>“.

Die verwendeten spitzen Klammern und weitere Sonderzeichen können ebenfalls, wie in Abbildung 2 gezeigt, in kodierter Form vorliegen. Die Dekodierung übernimmt der Web-Browser.

Damit ein Zusammenhang zwischen Ein- und Ausgabe hergestellt werden kann, muss nach absolvierter Kommunikation mit der Web-Applikation, jede Antwort mit den verwendeten Benutzereingaben verglichen werden. Dabei wird zum einen natürlich die ID verglichen, um den

Zusammenhang selbst herzustellen, zum anderen werden auch XSS-spezifische Erkennungsmerkmale verglichen, um eine Eingabevalidierung zu erkennen.

Liegt keine Eingabevalidierung vor, kann anhand der ausgelesenen ID der Zusammenhang zwischen Ein- und Ausgabe hergestellt werden und eine XSS Multistep-Schwachstelle identifiziert werden.

6 Fazit und Ausblick

Im Rahmen dieser Arbeit wurde sowohl eine Infrastruktur zur Evaluation unsicherer Web-Applikationen und Web-App Scannern geschaffen, als auch eine eigene unsichere Web-Applikation als Prüfelement implementiert und verwendet. Frei erhältliche Web-App Scanner zeigten dabei Defizite in der Erkennung bestimmter Schwachstellenkategorien. Die implementierte Web-Applikation beinhaltet 25 Schwachstellen und ist modular aufgebaut, um die Möglichkeit zur Erweiterung sicherzustellen. Die geschaffene Analyseinfrastruktur ermöglicht eine Analyse des Datenverkehrs und Scanner-Verhaltens und zeigt mögliches Verbesserungspotential in der Schwachstellenerkennung auf. Für drei der nicht gefundenen Schwachstellen konnte unter Verwendung des Frameworks ein Lösungsansatz gefunden werden, der die Erkennungsleistung von Web-App Scannern signifikant steigern könnte. Um die Leistungsfähigkeit des entwickelten Lösungsansatzes real zu erproben, müsste dieser technisch umgesetzt werden.

Für die Umsetzung der erarbeiteten Konzepte bieten sich Web-App Scanner mit modularem Aufbau und die Implementierung eines Plug-Ins an. Beispielsweise bietet w3af die Möglichkeit, verschiedene Plug-Ins zu benutzen. Ein solches open-source Projekt könnte Potential für die Realisierung eines Lösungskonzeptes in Form eines Plug-Ins bieten. Die gebotene Modularität bietet ebenfalls Vorteile bei der Evaluation der implementierten Lösung, da nicht benötigte Module abgeschaltet werden können und jegliche Zugriffe über einen Monitor-Dienst wie beispielsweise Wireshark oder einen geeigneten Proxy analysiert und zugeordnet werden können. Auch die Umsetzung als eigenständiges Skript, unabhängig von Betriebssystem und Web-App Scannern, ist in Zusammenhang mit einem geeigneten Proxy vorstellbar.

Literatur

[BBGM10] J. Bau, E. Burzstein, D. Gupta, J. Mitchell: State of the Art: Automated Black-Box Web Application Vulnerability Testing, Stanford University (2010). IEEE Symposium on Security and Privacy 2010: 332-345

[DoCV10] A. Doupé, M. Cova, G. Vigna: Why Johnny Can't Pentest: An Analysis of Black-box Web Vulnerability Scanners, University of California (2010). DIMVA 2010: 111-131

[Owas13] The Open Web Application Security Project: The Open Web Application Security Project: Top 10 – 2013, The Ten Most Critical Web Application Security Risks, 2013.