Penetrationtesting

Inhaltsverzeichnis

Probleme mit der Applikation

Am Tag des Austausches hatten wir grosse Probleme die VueJs Applikation zum Laufen zu bringen. Bei Julian schlug schon der Download der einzelnen NodeJs Modulen fehl. Bei Noel funktionierte dies, jedoch hatte ich 92 Vulnerabilities (“Sicherheitslücken”). Trotzdem versuchte er die Applikation zu starten. Der Start wurde durch eine Fehlermeldung unterbrochen. Noel hatte schnell realisiert das bei ihm ein grösseres Problem vorhanden war, da man selbst normale Tasks (“Aufgaben”) nicht mehr ausführen konnte. Beispielsweise `sudo npm install -g npm@latest` . Daraufhin hatte auch Herrn Kemper probiert die VueJs Applikation bei sich zu installieren und zu starten, ohne Erfolg.

Es ist nun der letzte Tag, an dem wir unseren Penetrationtesting durchführen können. Noel hatte das Problem mit meinem NPM lösen können und wollte noch einen aller letzten Versuch starten die Applikation doch noch zum Laufen zu kriegen. Trotz seinen Verbesserungen brachte er die Applikation nicht zum Laufen.

Plan B

Da nun wir mehrmals probiert hatten die VueJs Applikation zum Laufen zu bringen jedoch ohne Erfolg mussten wir einen Plan B finden. Dabei hatte Noel mit Dillan Kontakt aufgenommen, welcher die Seite auf einen Server für uns hostete und uns die Login Daten zur Verfügung stellte. Während Noel dann mit dem Penetratiobtesting begann, arbeitete Julian noch and der Dokumentation.

Penetrationtesting in Aktion

Noel hatte also nun alle nötigen Ressourcen und Daten beisammen, um mit den Penetrationtesting zu beginnen. Für das effiziente Testing nutzen wir OWASP ZAP, um spezifisch zu sein benutzten wir das Autotesting. Wir arbeiteten uns Seite für Seite durch, doch OWASP ZAP konnte keine extremen Sicherheitslücken feststellen. Lediglich ein paar Low Weight (“Leicht Gewicht”) und Medium Weight (“Medium Weight”) Warnungen tauchten auf. Als zweiter Schritt hatte Noel den Source Code analysiert, dies sowohl Client- als auch Serverseitig. Dabei fiel Noel eine fatale Lücke im System auf.

Testprotokoll

Alle der folgenden Tests sind nur mit dem OWASP ZAP Tool getestet worden.
Dieses Testprotokoll beinhaltet keine genaueren Analysen des Source Codes.

Uri:	/
Wer:	Kein Benutzer
Sicherheitslücken:	Nein
Status:	Ok

Uri:	/login
Wer:	Kein Benutzer
Sicherheitslücken	Nein
Status:	Ok

Uri:	/dashboard
Wer:	Normaler Benutzer
Sicherheitslücken	Nein
Status:	Ok

Sicherheitslücken

Firebase Sicherheitslücke

Beschreibung
Diese Lücke ermöglichte es einem ohne Registration oder Admin Rechte sich Zugang zu der Datenback (“Firebase”) zu verschaffen und dort Daten zu ändern.

Wie einfach ist es die Lücke zu schliessen?
Die Lücke erfordert eine grundgehende Neuentwicklung der Seite.

Wie gross sind die Auswirkungen?
Mit der Sicherheitslücke ist es möglich ohne Admin Rechte und ohne Registration User zu erstellen, bearbeiten und sogar zu löschen. Kurzgesagt man kann die volle Kontrolle über die Nutzer erlangen.

Wie einfach kann die Lücke ausgenutzt werden, um Schaden anzurichten?
Sehr einfach, wie erwähnt benötigt man keine Admin Rechte oder eine Registration. Jeder kann ohne weiteres diese Lücke ausnützen und erheblichen Schaden anrichten.

Wie wahrscheinlich ist es, dass die Lücke von Angreifern gefunden und ausgenutz wird?
Die Wahrscheinlichkeit ist sehr hoch da man diese schon bei etwas genauerem betrachten des Clients (“Browser”) finden kann.

Schliessung der Sicherheitslücke

Firebase Sicherheitslücke

Theoretisch könnte man eine API zwischen User und Firebase setzen welche eine gewisse Abgrenzung bietet. Diese Methode wäre nach meiner Meinung die günstigste zur Realisierung. Die API würde dann ein Authentifizierungssystem beinhalten welche die Aktionen des Users genaustens überwacht. Auch ein Logging + Monitoring System könnte man als zusätzliche Sicherheit mit einbauen.