securitum

Security report

SUBJECT

Simple Login Mobile Application (Android)

DATE

07.03.2022 – 11.03.2022

RETEST DATE

N/A

LOCATION

Cracow (Poland)

AUDITOR

Dariusz Tytko

VERSION

1.0

Executive summary

This document is a summary of work conducted by Securitum company. The subject of the test was the mobile application Single Login for Android system available at:

- https://play.google.com/store/apps/details?id=io.simplelogin.android

Tests were conducted using the following roles: user with an account and anonymous user (without an account).

The most severe vulnerability identified during the assessment was:

- Brute-force login mechanism

During the tests, particular emphasis was placed on vulnerabilities that might in a negative way affect confidentiality, integrity or availability of processed data.

The security tests were carried out in accordance with generally accepted methodologies, including: OWASP TOP10, (in a selected range) OWASP ASVS, OWASP MASVS as well as internal good practices of conducting security tests developed by Securitum.

An approach based on manual tests (using the above-mentioned methodologies), supported by a number of automatic tools (i.a. Burp Suite Professional, ffuf, MobSF), was used during the assessment.

The vulnerabilities are described in detail in further parts of the report.

Risk classification

Vulnerabilities are classified in a five-point scale, that is reflecting both the probability of exploitation of the vulnerability and the business risk of its exploitation. Below, there is a short description of meaning of each of severity levels:

- CRITICAL – exploitation of the vulnerability makes it possible to compromise the server or network device, or makes it possible to access (in read and/or write mode) data with a high degree of confidentiality and significance. The exploitation is usually straightforward, i.e. an attacker does not need to gain access to the systems that are difficult to reach and does not need to perform any kind of social engineering. Vulnerabilities marked as 'CRITICAL' must be fixed without delay, especially if they occur in production environment.

- HIGH – exploitation of the vulnerability makes it possible to access sensitive data (similar to 'CRITICAL' level), however the prerequisites for the attack (e.g. possession of a user account in an internal system) makes it slightly less likely. Alternatively, the vulnerability is easy to exploit, but the effects are somehow limited.

- MEDIUM – exploitation of the vulnerability might depend on external factors (e.g. convincing the user to click on a hyperlink) or other conditions that are difficult to achieve. Furthermore,

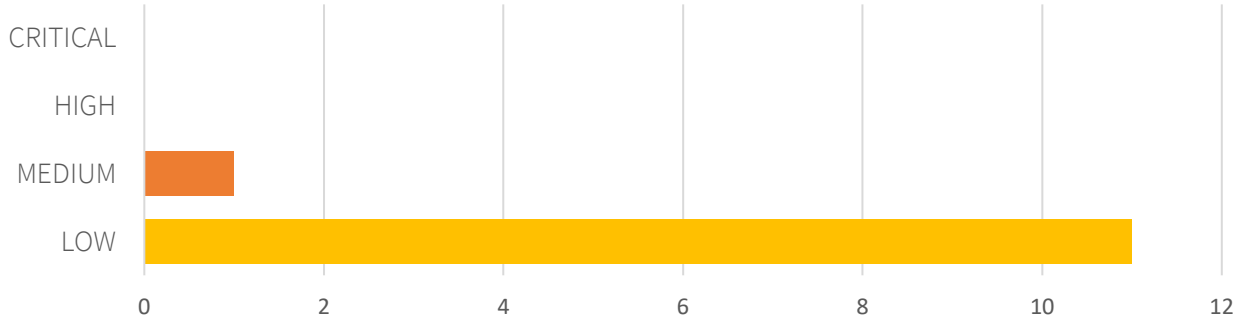
exploitation of the vulnerability usually allows access only to a limited set of data or to data of a lesser degree of significance.

- LOW – exploitation of the vulnerability results in minor direct impact on the security of the test subject or depends on conditions that are very difficult to achieve in practical manner (e.g. physical access to the server).

- INFO – issues marked as 'INFO' are not security vulnerabilities per se. Their aim is to point out good practices, the implementation of which will lead to the overall increase of the system security level. Alternatively, the issues point out some solutions in the system (e.g. from an architectural perspective) that might limit the negative effects of other vulnerabilities.

Statistical overview

Below, a statistical overview of vulnerabilities is shown:



Additionally, 4 INFO issues were reported.

Contents

Change history

Document date	Version	Change description
14.03.2022	1.0	Final version.

Vulnerabilities

[MEDIUM] SECURITUM-221794-001: Brute-force login mechanism

SUMMARY

Login mechanism is vulnerable to brute-force attack. Password checking mechanism implements a protection in the form of API Rate Limiting but it is still possible to check ~24 passwords per minute (1440 password per hour) from the same IP address. Additionally, attack can be distributed – e.g. carried out from N VPSs (different IP addresses) what allows to check 1440 * N passwords per hour. In combination with username enumeration (see *SECURITUM-221794-002: Username enumeration*) and weak password policy (see *SECURITUM-221794-013: Weak password policy*) the vulnerability introduces risk of unauthorized access to the users' accounts.

Optional MFA mechanism is also vulnerable to brute-force attack. In this case there is no API Rate Limiting thus it is possible to check one time code with no limits.

More information:

- https://owasp.org/www-project-top-ten/2017/A2_2017-Broken_Authentication

PREREQUISITES FOR THE ATTACK

Attacker has to know user's login (e-mail).

TECHNICAL DETAILS (PROOF OF CONCEPT)

To perform the attack on password checking mechanism the Burp Suite Professional (Intruder module) tool was used. The following request was sent automatically:

```
POST /api/auth/login HTTP/1.1
Content-Type: application/json; charset=utf-8
Content-Length: 75
Host: app.simplelogin.io
Connection: close
Accept-Encoding: gzip, deflate
User-Agent: okhttp/4.8.0

{"email":"dt1+sl01@securitum.pl","password":"<password candidate>","device":"walleye"}
```

The tool was configured to send one request per 2500 milliseconds (~24 requests per second). In the result, API Rate Limiting protection was not activated (429 response code), and after sending the 150 requests password was guessed (200 response code):

Request	Payload	Status	Error	Timeout	Length
133	gocougs	400			486
134	good-luck	400			486
135	graymail	400			486
136	guinness	400			486
137	hilbert	400			486
138	hola	400			486
139	home	400			486
140	homebrew	400			486
141	hotdog	400			486
142	indian	400			486
143	protel	400			486
144	psalms	400			486
145	qwaszx	400			486
146	walker	400			486
147	watson	400			486
148	young	400			486
149	zhongguo	400			486
150	[redacted]	200			1472

The second (optional) authentication step – MFA one-time code validation request was possible to send with no limits:

```
POST /api/auth/mfa HTTP/1.1
Content-Type: application/json; charset=utf-8
Content-Length: 88
Host: app.simplelogin.io
Connection: close
Accept-Encoding: gzip, deflate
User-Agent: okhttp/4.8.0

{"mfa_token":"309523","mfa_key":"113778.A72g[…]nAeM","device":"walleye"}
```

LOCATION

- POST /api/auth/login
- POST /api/auth/mfa

RECOMMENDATION

It is recommended to eliminate the problems reported in the following report points:

- SECURITUM-221794-002: Username enumeration,
- SECURITUM-221794-013: Weak password policy.

Additionally, it is recommended to implement the following changes:

- CAPTCHA protection (per account) activated after a few failed login attempts,
- Decreasing API Rate Limiting threshold for password checking API endpoint (`POST /api/auth/login`),
- API Rate Limiting for MFA API endpoint (`POST /api/auth/mfa`).

More information:

- https://owasp.org/www-community/controls/Blocking_Brute_Force_Attacks

[LOW] SECURITUM-221794-002: Username enumeration

SUMMARY

Attacker is able to check if the given username (e-mail address) is used in the application. Lists of the valid email addresses can be used to perform further attacks e.g. sending phishing e-mails or brute-force (see *SECURITUM-221794-001: Brute-force login mechanism*).

More information:

- https://owasp.org/www-project-top-ten/2017/A2_2017-Broken_Authentication

PREREQUISITES FOR THE ATTACK

None – anonymous access to the application.

TECHNICAL DETAILS (PROOF OF CONCEPT)

The following variants of the username vulnerability were identified:

#1 The following login request was sent to check if the given email address is valid (notice sending null as a password):

```
POST /api/auth/login HTTP/1.1
Content-Type: application/json; charset=utf-8
Content-Length: 68
Host: app.simplelogin.io
Connection: close
Accept-Encoding: gzip, deflate
User-Agent: okhttp/4.8.0

{"email":"dt1+sl01@securitum.pl","password":null,"device":"walleye"}
```

If the username was valid the following error was returned:

```
HTTP/1.1 500 INTERNAL SERVER ERROR
Server: nginx/1.18.0 (Ubuntu)
Date: Tue, 08 Mar 2022 10:25:45 GMT
Content-Type: application/json
Content-Length: 27
Connection: close
Access-Control-Allow-Origin: *
Vary: Cookie
Set-Cookie: slapp=[…]; Expires=Tue, 15-Mar-2022 10:25:45 GMT; Secure; HttpOnly; Path=/;
SameSite=Lax
Strict-Transport-Security: max-age=63072000; includeSubDomains; preload

{"error":"Internal error"}
```

If the username was not valid, returned error message was different:

```
HTTP/1.1 400 BAD REQUEST
Server: nginx/1.18.0 (Ubuntu)
Date: Tue, 08 Mar 2022 10:26:37 GMT
Content-Type: application/json
```

```
Content-Length: 40
Connection: close
Access-Control-Allow-Origin: *
Vary: Cookie
Set-Cookie: slapp=[…]; Expires=Tue, 15-Mar-2022 10:26:37 GMT; Secure; HttpOnly; Path=/;
SameSite=Lax
Strict-Transport-Security: max-age=63072000; includeSubDomains; preload

{"error":"Email or password incorrect"}
```

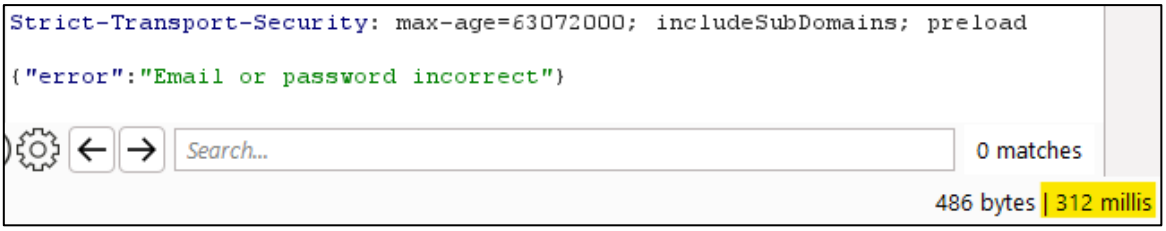
It seems that password was processed only for the valid username and null value caused an exception.

#2 Time-based enumeration vulnerability was identified in sign in mechanism. The following request was sent to check if the given username is valid (incorrect password was used):

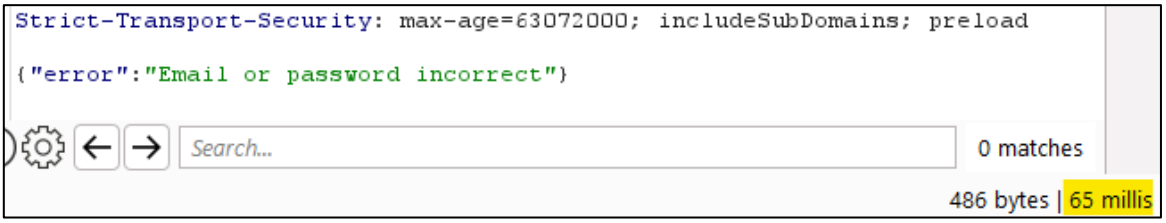
```
POST /api/auth/login HTTP/1.1
Content-Type: application/json; charset=utf-8
Content-Length: 84
Host: app.simplelogin.io
Connection: close
Accept-Encoding: gzip, deflate
User-Agent: okhttp/4.8.0

{"email":"dt1+sl05@securitum.pl","password":"incorrect-password","device":"walleye"}
```

For the valid username, response was returned after 312 milliseconds:



For an invalid username, response was return after 65 milliseconds:



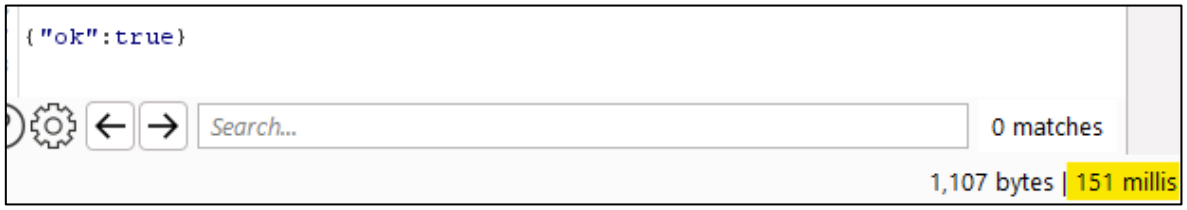
Response time difference allows an attacker to deduce if the username is used in the application.

#3 Time-based enumeration vulnerability was identified in forgot password mechanism. The following request was sent to check if the given username is valid:

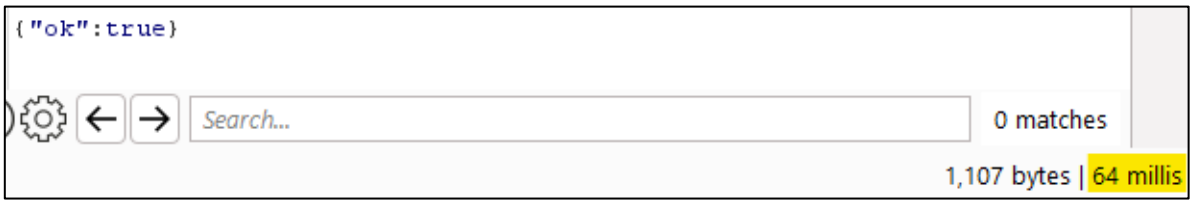
```
POST /api/auth/forgot_password HTTP/1.1
Content-Type: application/json; charset=utf-8
Content-Length: 33
Host: app.simplelogin.io
Connection: close
Accept-Encoding: gzip, deflate
User-Agent: okhttp/4.8.0

{"email":"dt1+sl01@securitum.pl"}
```

For the valid username, response was returned after 151 milliseconds:



For an invalid username, response was return after 64 milliseconds:



Response time difference allows an attacker to deduce if the username is used in the application.

#4 The following registration request was sent to check if the given email address is valid:

```
POST /api/auth/register HTTP/1.1
Content-Type: application/json; charset=utf-8
Content-Length: 56
Host: app.simplelogin.io
Connection: close
Accept-Encoding: gzip, deflate
User-Agent: okhttp/4.8.0

{"email":"dt1+sl06@securitum.pl","password":"[…]"}
```

If the username was valid the following error was returned:

```
HTTP/1.1 400 BAD REQUEST
[…]

{"error":"cannot use dt1+sl06@securitum.pl as personal inbox"}
```

If the username was not valid, returned response was different:

```
HTTP/1.1 200 OK
[…]

{"msg":"User needs to confirm their account"}
```

LOCATION

- POST /api/auth/login
- POST /api/auth/forgot_password
- POST /api/auth/register

RECOMMENDATION

There should be no difference (content and time) in the response for valid and invalid username.

[LOW] SECURITUM-221794-003: Omitting the first authentication step

SUMMARY

It is possible to omit the first authentication step (providing login and password) if the MFA is enabled. It increases risk of unauthorized access to the user's account.

PREREQUISITES FOR THE ATTACK

Attacker has to know signed user id and one-time code from the authentication application.

TECHNICAL DETAILS (PROOF OF CONCEPT)

During the tests the MFA was enabled for pentester's account. Due to that, the following requests were sent during the authentication:

#1 Login request:

```
POST /api/auth/login HTTP/1.1
Content-Type: application/json; charset=utf-8
Content-Length: 75
Host: app.simplelogin.io
Connection: close
Accept-Encoding: gzip, deflate
User-Agent: okhttp/4.8.0

{"email":"dt1+sl02@securitum.pl","password":"[…]","device":"walleye"}
```

In response, signed user id was returned:

```
HTTP/1.1 200 OK
[…]

{"api_key":null,"email":"dt1+sl02@securitum.pl","mfa_enabled":true,"mfa_key":"113778.A72g[…]nAeM"
,"name":""}
```

#2 MFA request containing one-time code and signed user id:

```
POST /api/auth/mfa HTTP/1.1
Content-Type: application/json; charset=utf-8
Content-Length: 88
Host: app.simplelogin.io
Connection: close
Accept-Encoding: gzip, deflate
User-Agent: okhttp/4.8.0

{"mfa_token":"558765","mfa_key":"113778.A72g[…]nAeM","device":"walleye"}
```

Response containing API key:

```
HTTP/1.1 200 OK
[…]

{"api_key":"mdcu[…]wbya","email":"dt1+sl02@securitum.pl","name":""}
```

It was observed that it is possible to login by sending only the second request (MFA), the first one is not necessary. Due to that attacker who knows the signed user id and has access to the user's authentication application will be able to login to the account without knowing the user's password.

LOCATION

Authentication mechanism.

RECOMMENDATION

Both steps – providing login/password and providing one-time code should be required during the authentication for MFA-enabled accounts.

[LOW] SECURITUM-221794-004: Sending reset password token to plausible service

SUMMARY

It was found that reset password token is sent to plausible service (plausible.simplelogin.io). It increases risk of unauthorized access to the sensitive data by plausible operator.

PREREQUISITES FOR THE ATTACK

Access to https://plausible.simplelogin.io.

TECHNICAL DETAILS (PROOF OF CONCEPT)

The following request was observed while using the reset password mechanism:

```
POST /api/event HTTP/1.1
Host: plausible.simplelogin.io
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:98.0) Gecko/20100101 Firefox/98.0
Accept: */*
Accept-Language: pl,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: text/plain
Content-Length: 178
Origin: https://app.simplelogin.io
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-site
Pragma: no-cache
Cache-Control: no-cache
Te: trailers
Connection: close

{"n":"pageview","u":"https://app.simplelogin.io/auth/reset_password?code=ujtl[...]mfhu","d":"app.
simplelogin.io","r":null,"w":1920}
```

LOCATION

Reset password mechanism.

RECOMMENDATION

No sensitive data should be sent to plausible service.

[LOW] SECURITUM-221794-005: Blocking e-mail address for registration

SUMMARY

It was noticed that during the registration an active code is sent only once. After that there is no possibility to resend the activation code again. Due to that, an attacker can start registration process for many e-mail addresses, and in the future the owners of these addresses may have a problem with registration. The only way to get access to such accounts is using forgot password functionality but it is not stated outright and may not be obvious for person who tries to register in the application.

PREREQUISITES FOR THE ATTACK

None – attack can be conducted by anonymous user.

TECHNICAL DETAILS (PROOF OF CONCEPT)

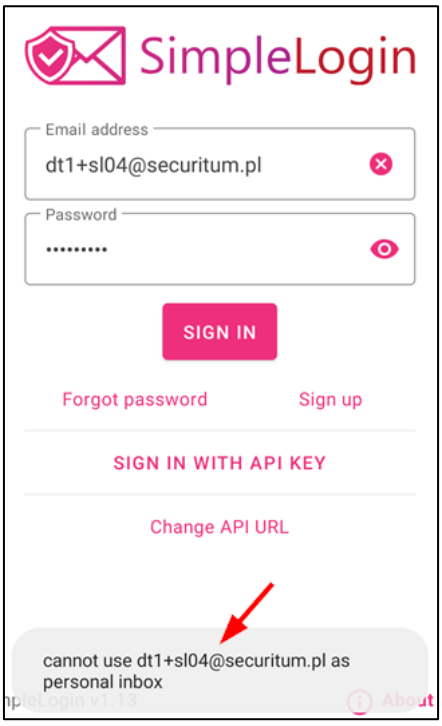
The following request was sent to start registration process:

```
POST /api/auth/register HTTP/1.1
Content-Type: application/json; charset=utf-8
Content-Length: 55
Host: app.simplelogin.io
Connection: close
Accept-Encoding: gzip, deflate
User-Agent: okhttp/4.8.0

{"email":"dt1+sl04@securitum.pl","password":"[…]"}
```

Activation code was sent to dt1+sl04@securitum.pl but activation process has not been finished.

As a result, it was not possible to register an account for this e-mail address. There was also no possibility to resend an activation code, no information about using forgot password functionality was displayed either.



LOCATION

Registration mechanism.

RECOMMENDATION

It recommended to add a possibility to resend activation code during the registration process or add clear instruction that forgot password functionality can be used to get an access to the account that was registered by other person using user's e-mail address.

[LOW] SECURITUM-221794-006: Automatic resources creation

SUMMARY

It was found that is possible to automate creation of some resources (e.g. account registration). There is API Rate Limiting protection, but it is still possible to register ~24 accounts per minute. It may allow to perform a Denial of Service attack.

Analogous problem exists for other resources like mailboxes and API keys.

PREREQUISITES FOR THE ATTACK

None – attack can be conducted by anonymous user.

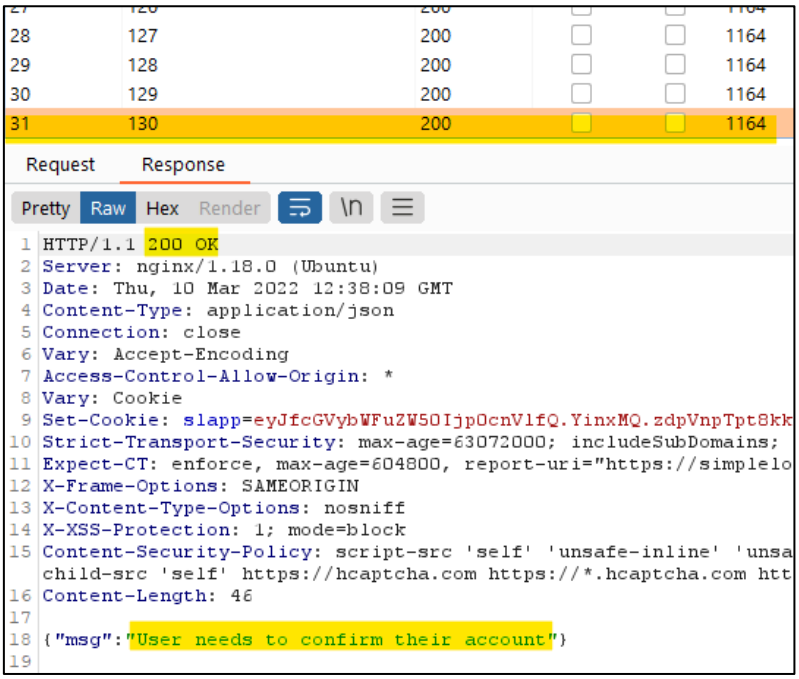
TECHNICAL DETAILS (PROOF OF CONCEPT)

#1 The following, registration request was sent automatically using Burp Suite Professional (module Intruder) tool:

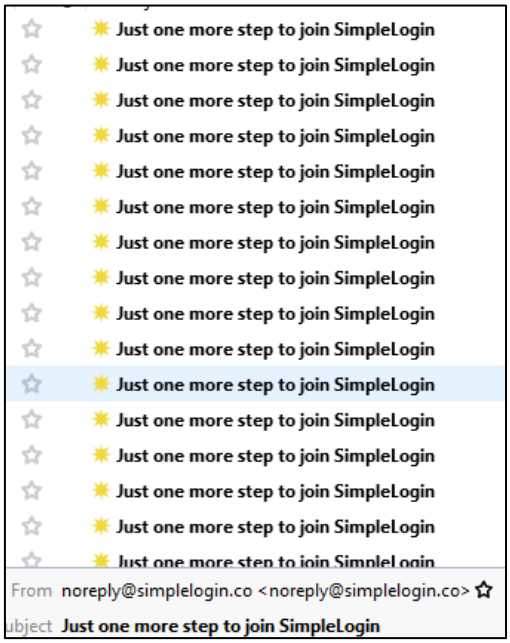
```
POST /api/auth/register HTTP/1.1
Content-Type: application/json; charset=utf-8
Content-Length: 56
Host: app.simplelogin.io
Connection: close
Accept-Encoding: gzip, deflate
User-Agent: okhttp/4.8.0

{"email":"random email","password":"radom password"}
```

The tool was configured to send one request per 2500 milliseconds (~24 requests per second). In the result, API Rate Limiting protection was not activated (429 response code), and in a short period of time dozens of account were crated:



It is important to add that many “spam” email messages was also sent (auditor’s email box view):

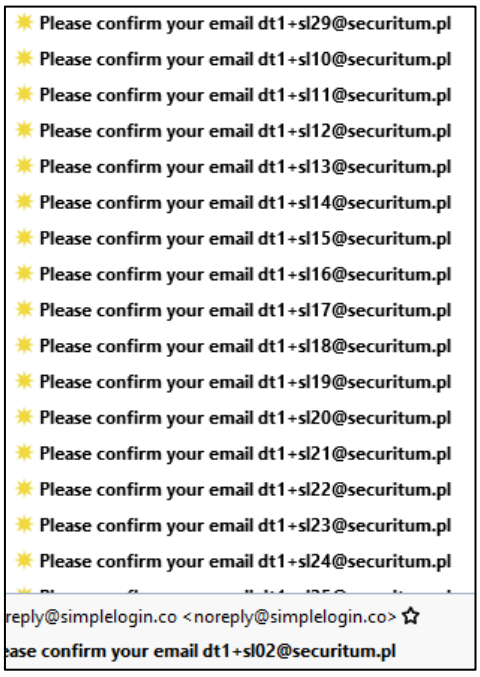


#2 The following request was used to create many mailboxes. In this case there is no API Rate Limiting at all:

```
POST /api/mailboxes HTTP/1.1
Authentication: […]
Content-Type: application/json; charset=utf-8
Content-Length: 33
Host: app.simplelogin.io
Connection: close
Accept-Encoding: gzip, deflate
User-Agent: okhttp/4.8.0

{"email":"arbitrary email address"}
```

As a result many mailboxes were created, and a lot of spam email messages were sent:

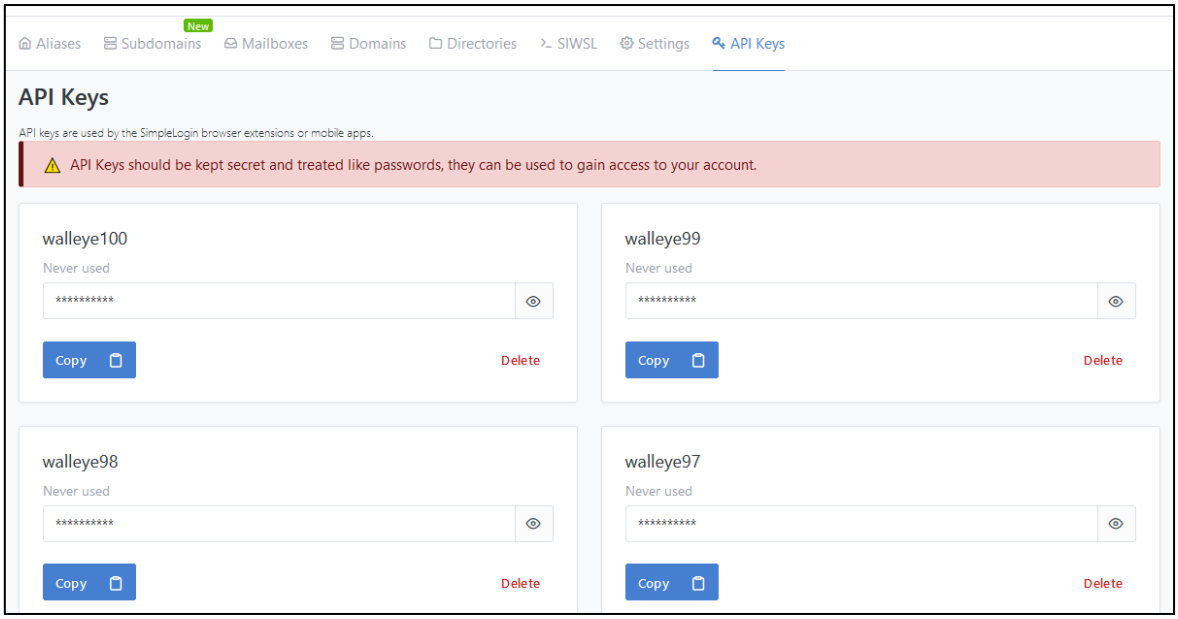


#3 The login requests with unique device names were send automatically, 100 times using Burp Suite Professional (module Intruder):

```
POST /api/auth/login HTTP/1.1
Content-Type: application/json; charset=utf-8
Content-Length: 76
Host: app.simplelogin.io
Connection: close
Accept-Encoding: gzip, deflate
User-Agent: okhttp/4.8.0

{"email":"correct username","password":"correct password","device":"random unique device name"}
```

As a result, 100 API keys were generated (web application view):



It was found that it is also possible to create the API keys directly using the following request:

```
POST /api/api_key HTTP/1.1
Authentication: codm[…]kshd
Content-Type: application/json; charset=utf-8
Host: app.simplelogin.io
Connection: close
Accept-Encoding: gzip, deflate
User-Agent: okhttp/4.8.0
Content-Length: 13

{"device":""}
```

LOCATION

- POST /api/auth/register
- POST /api/mailboxes
- POST /api/auth/login
- POST /api/api_key

RECOMMENDATION

It is recommended to implement CAPTCHA protection. It is also recommended to decrease API Rate Limiting threshold for registration API endpoint (`POST /api/auth/register`) and provide API Rate Limiting with low threshold for other API endpoints creating resources.

This point should be treated as a general recommendation to provide a protection against automatic creation of any resources.

[LOW] SECURITUM-221794-007: Sending spam emails

SUMMARY

API endpoints were detected that allow to send spam emails to the email addresses not registered in the application. The API endpoint are not protected by API Rate Limiting. Due to that it is possible to send a large amount of emails with no limits.

SECURITUM-221794-006: Automatic resources creation vulnerability describes other examples of sending spam email messages.

PREREQUISITES FOR THE ATTACK

None – attack can be conducted by anonymous user.

TECHNICAL DETAILS (PROOF OF CONCEPT)

During the test a registration request was sent:

```
POST /api/auth/register HTTP/1.1
Content-Type: application/json; charset=utf-8
Content-Length: 56
Host: app.simplelogin.io
Connection: close
Accept-Encoding: gzip, deflate
User-Agent: okhttp/4.8.0

{"email":"dt1+sl09@securitum.pl","password":"[…]"}
```

Then the following request was sent automatically using Burp Suite Professional (module Intruder) tool:

```
POST /api/auth/reactivate HTTP/1.1
Content-Type: application/json; charset=utf-8
Content-Length: 33
Host: app.simplelogin.io
Connection: close
Accept-Encoding: gzip, deflate
User-Agent: okhttp/4.8.0

{"email":"dt1+sl09@securitum.pl"}
```

No protection mechanism was enabled, and many unwanted email messages were received (auditor's email box view):

Analogous problem exists for the following API endpoint:

```
PUT /api/mailboxes/{id} HTTP/1.1
Authentication: codm[…]kshd
Content-Type: application/json; charset=utf-8
Content-Length: 33
Host: app.simplelogin.io
Connection: close
Accept-Encoding: gzip, deflate
User-Agent: okhttp/4.8.0

{"email":"arbitrary email address"}
```

LOCATION

- POST /api/auth/reactivate
- PUT /api/mailboxes/{id}

RECOMMENDATION

It is recommended to implement CAPTCHA protection, it is also recommended to implemented API Rate Limiting with low threshold for any API endpoint that allow to send email messages.

This point should be treated as a general recommendation to provide a protection against automatic sending of large amount of unwanted/spam email messages.

[LOW] SECURITUM-221794-008: Uploading an arbitrary file into the application s3 bucket

SUMMARY

It is possible to upload an arbitrary file into the application s3 bucket used to storage the profile photos. An attacker can use this vulnerability to storage any files e.g. malware files and use them to conduct any other attacks.

PREREQUISITES FOR THE ATTACK

Account in the application.

TECHNICAL DETAILS (PROOF OF CONCEPT)

The following request was used to put EICAR[1] file on the s3 bucket (file content is base64-encoded):

```
PATCH /api/user_info HTTP/1.1
Authentication: codm[…]kshd
Content-Type: application/json; charset=utf-8
Content-Length: 114
Host: app.simplelogin.io
Connection: close
Accept-Encoding: gzip, deflate
User-Agent: okhttp/4.8.0

{"profile_picture":"WDVPIVAlQEFQWzRcUFpYNTQoUF4pN0NDKTd9JEVJQ0FSLVNUQU5EQVJELUFOVElWSVJVUy1URVNULUZJTEUhJEgrSCo="}
```

Response contained URL to the uploaded file:

```
HTTP/1.1 200 OK
[…]

{"email":"dt1+sl01@securitum.pl","in_trial":true,"is_premium":true,"name":"<i>test</i>","profile_picture_url":"https://s3.eu-west-3.amazonaws.com/prod.sl/wigmeieiqwueifnhllnyquhhkiymxr?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAYA7QYVHRAOSRW4I6%2F20220310%2Feu-west-3%2Fs3%2Faws4_request&X-Amz-Date=20220310T202103Z&X-Amz-Expires=3600&X-Amz-SignedHeaders=host&X-Amz-Signature=f22fa079d7765781f13cf96460e11a9cfc97054bfb9275582ff282df7613a230"}
```

Request was sent twice and both URLs were working. It indicates that previous file (profile's image) was not deleted from the s3 bucket. It may expose the application's owner to costs by sending a large amount of files.

LOCATION

PATCH /api/user_info

[1] https://www.eicar.org/?page_id=3950

RECOMMENDATION

It is recommended to implement the following improvements:

- Validation if the uploading file is a correct image,
- Maximum size of the uploading file,
- Removing previous profile's picture.

[LOW] SECURITUM-221794-009: Technical information disclosure in HTTP headers

SUMMARY

Redundant information leakage has been detected. HTTP response headers contain information about web server – nginx/1.18.0 (Ubuntu). This kind of information can be used to prepare further, platform specific attacks.

PREREQUISITES FOR THE ATTACK

Access to the application.

TECHNICAL DETAILS (PROOF OF CONCEPT)

The following, example HTTP response contains redundant information:

```
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Tue, 08 Mar 2022 09:57:11 GMT
Content-Type: application/json
Connection: close
Vary: Accept-Encoding
Access-Control-Allow-Origin: *
Vary: Cookie
[…]
```

LOCATION

https://app.simplelogin.io/*

RECOMMENDATION

No redundant information should be returned in the HTTP responses.

[LOW] SECURITUM-221794-010: Changing API URL

SUMMARY

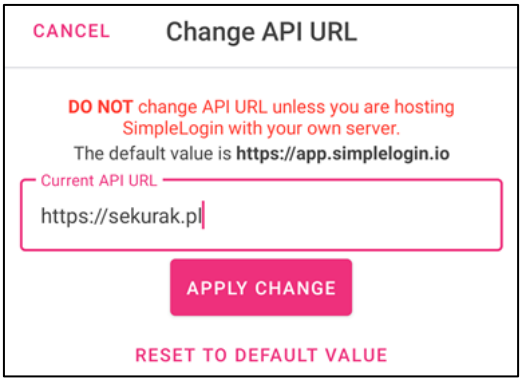
The mobile application allows to change the API URL on the login panel. At the same time the login panel does not present on what URL the application connects to. Due to that there is a risk that an attacker who has temporary access to the phone change the API URL, and unaware user will be using the application while network traffic will be sent through the attacker's server.

PREREQUISITES FOR THE ATTACK

Access to an unblocked phone.

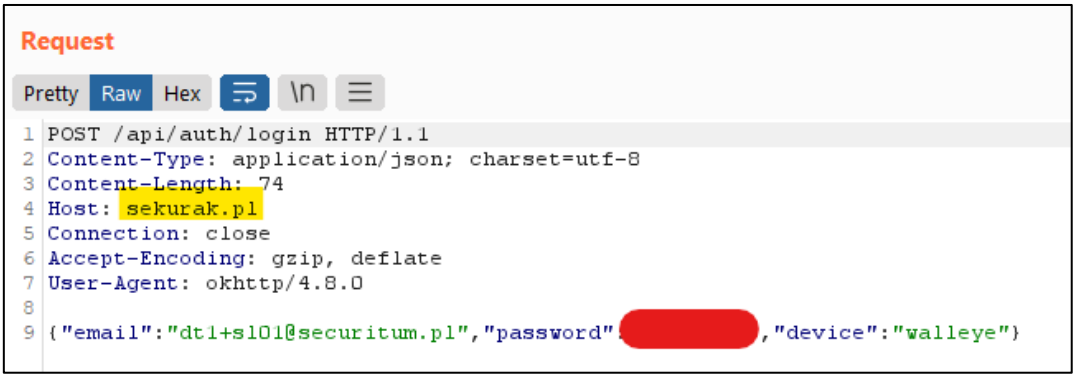
TECHNICAL DETAILS (PROOF OF CONCEPT)

API URL was changed to https://sekurak.pl:



On the login panel there was no information that API URL was changed:

When logging in, data was sent to sekurak.pl server (request was captured using Burp Suite Professional (Proxy module)):



LOCATION

Login panel on the mobile application.

RECOMMENDATION

Login panel should countian the information that default API URL was changed.

[LOW] SECURITUM-221794-011: API key stored in unencrypted form in the filesystem

SUMMARY

API key is stored in unencrypted form in the file system. It exposes the user to the risk of the sensitive data leakage. It is worth to mention that API key leakage can gain an attacker permanent access to the account (see *SECURITUM-221794-015: Permanent session identifier*).

PREREQUISITES FOR THE ATTACK

Attacker needs an access to the application files.

TECHNICAL DETAILS (PROOF OF CONCEPT)

`/data/data/io.simplelogin.android/shared_prefs` file contains API key:

```
walleye:/data/data/io.simplelogin.android/shared_prefs # cat io.simplelogin.android.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="API_URL">https://app.simplelogin.io</string>
    <string name="API_KEY">cod                                                    xshd</string>
    <boolean name="RATED" value="true" />
    <boolean name="FORCE_DARK_MODE" value="true" />
    <boolean name="SHOULD_LOCALLY_AUTHENTICATE" value="false" />
</map>
walleye:/data/data/io.simplelogin.android/shared_prefs #
```

LOCATION

/data/data/io.simplelogin.android/shared_prefs

RECOMMENDATION

Sensitive data should be stored in encrypted form. Encryption key should be stored in the system keystore protected by device PIN or biometry.

[LOW] SECURITUM-221794-012: Extracting API key from backup

SUMMARY

Application allows to make backup of its files. This greatly helps in accessing application files – root access is not necessary. During the tests the backup mechanism was used to get an access to API key.

PREREQUISITES FOR THE ATTACK

Access to the unblocked user's phone.

TECHNICAL DETAILS (PROOF OF CONCEPT)

Fragment of `AndroidManifest.xml` file:

```
"@mipmap/ic_launcher" android:allowBackup="true" android:
```

The following steps were done to extract an API key using backup:

```
$ adb backup -f backup.ab io.simplelogin.android
$ dd if=backup.ab bs=24 skip=1 > backup.zlib
$ zlib-flate -uncompress < backup.zlib > backup.tar
$ tar xf backup.tar
$ cat apps/io.simplelogin.android/sp/io.simplelogin.android.xml
```

```
kali@kali:~/project2/backup$ cat apps/io.simplelogin.android/sp/io.simplelogin.android.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="API_URL">https://app.simplelogin.io</string>
    <string name="API_KEY">co[redacted]hd</string>
    <boolean name="RATED" value="true" />
    <boolean name="FORCE_DARK_MODE" value="false" />
    <boolean name="SHOULD_LOCALLY_AUTHENTICATE" value="false" />
</map>
```

LOCATION

Application for Android system.

RECOMMENDATION

It is recommended to disable the backup functionality by changing the `AndroidManifest.xml` file to:

```
android:allowBackup="false"
```

Informational issues

[INFO] SECURITUM-221794-013: Weak password policy

SUMMARY

Application does not enforce using strong passwords. During the tests, it was possible to set "12345678" password. Lack of the strong password policy increases risk of the unauthorized access to the accounts (see *SECURITUM-221794-001: Brute-force login mechanism*).

TECHNICAL DETAILS (PROOF OF CONCEPT)

At this moment application requires only minimum password length – 8 characters.

LOCATION

Password policy.

RECOMMENDATION

It is recommended to implement the requirements regarding password complexity, in particular:

a) Enforcing a minimum password length of at least 12 characters and a maximum length of up to 128 characters (length limitation should be introduced due to potential DoS attacks in the absence of it);
b) Checking if the password is not present in at least 10,000 of the most popular passwords from database leaks and other sources, as well as in publicly available password dictionaries (most commonly used for brute-force attacks);
c) Checking if the password does not contain phrases related to the application or user (e.g. application name, user name etc.);

More information:

- https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html
- https://github.com/danielmiessler/SecLists/tree/master/Passwords/Common-Credentials

[INFO] SECURITUM-221794-014: Missing change password mechanism

SUMMARY

Application does not provide change password mechanism. It is important part of the authentication mechanism. Without it user has to use reset-password mechanism to change password (e.g. in case of credentials leakage).

More information:

- https://owasp.org/www-project-top-ten/2017/A2_2017-Broken_Authentication

TECHNICAL DETAILS (PROOF OF CONCEPT)

N/A

LOCATION

Application for Android system.

RECOMMENDATION

Application should provide change password mechanism. Implementation may be based on reset-password mechanism but there should be a dedicated option for this functionality in the account settings.

[INFO] SECURITUM-221794-015: Permanent session identifier

SUMMARY

Application uses permanent API key as a session identifier. Such API key is generated once, after the first successful login, and does not change for a user name, device name pair. In the case of a leak of a session identifier, attacker will gain permanent access to the user's account (even if the user enables MFA).

TECHNICAL DETAILS (PROOF OF CONCEPT)

During the tests the same session token (API key) was returned for any successful login request:

```
POST /api/auth/login HTTP/1.1
Content-Type: application/json; charset=utf-8
Content-Length: 75
Host: app.simplelogin.io
Connection: close
Accept-Encoding: gzip, deflate
User-Agent: okhttp/4.8.0

{"email":"dt1+sl01@securitum.pl","password":"[…]","device":"walleye"}
```

Response:

```
HTTP/1.1 200 OK
[…]

{"api_key":"codm[…]kshd","email":"dt1+sl01@securitum.pl","mfa_enabled":false,"mfa_key":null,"name
":"test"}
```

LOCATION

Session management.

RECOMMENDATION

It is recommended to use random, expiring values as a session token. Cryptography-secured generator should be used for this purpose. After logout, session token should be invalidated.

[INFO] SECURITUM-221794-016: Missing rooted device detection

SUMMARY

Tested application does not verify whether device on which app is running was rooted. Good security practices related to Android environment suggest to verify whether such device was rooted, and at least notify user about that fact.

TECHNICAL DETAILS (PROOF OF CONCEPT)

N/A

LOCATION

Application for Android system.

RECOMMENDATION

Application should verify whether it is running on secure (not rooted) device, and at least notify user, that unsecure device is used.