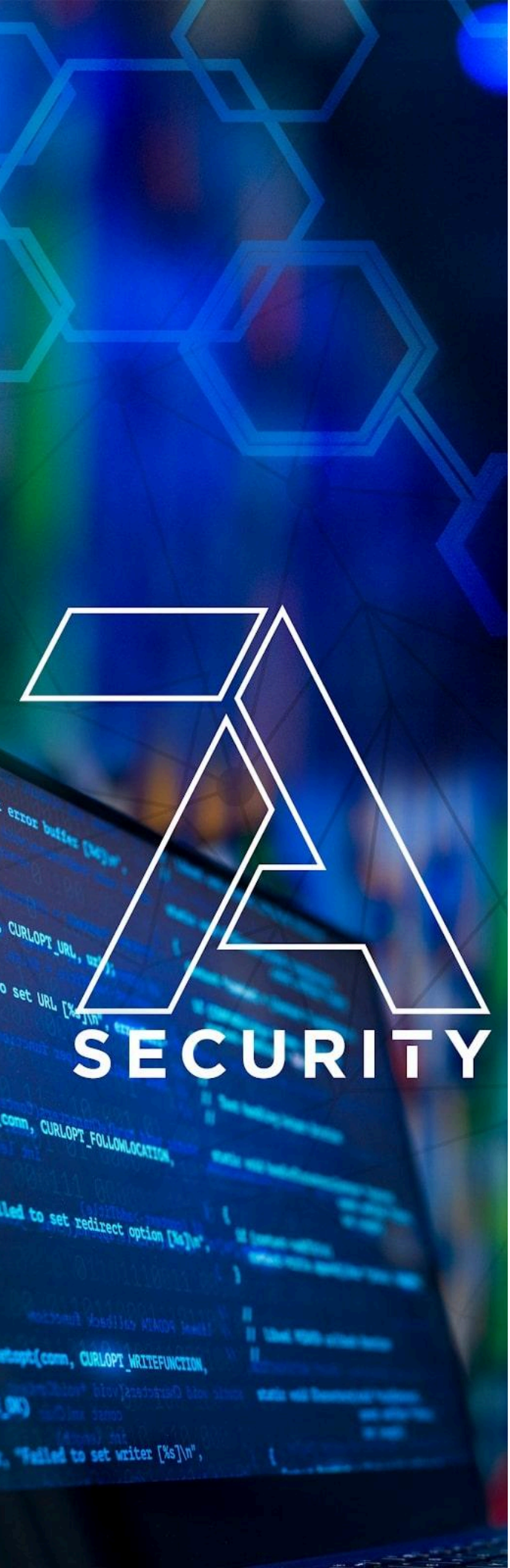



## CoverDrop Test Targets:

Protocol & Library
Mobile apps
Backend Services
Servers
AWS Infrastructure
Kubernetes Infrastructure
Threat Model
Supply Chain

# Pentest Report

---

Client:

*The Guardian*

*Guardian News & Media Limited*

**7ASecurity Test Team:**

- Abraham Aranguren, MSc.
- Daniel Ortiz, MSc.
- Dariusz Jastrzębski
- Harsh Bothra, BTech.
- Miroslav Štampar, PhD.
- Szymon Grzybowski, MSc.

7ASecurity

*Protect Your Site & Apps*

*From Attackers*

sales@7asecurity.com

7asecurity.com

# INDEX

# Introduction

*"CoverDrop can be integrated into existing news apps and enables news app users to contact journalists at that news organisation, reducing the risk of insecure communication from the start. It uses cover traffic generated by all the regular users of the news app to hide whistleblowers' communication, where traffic is passed through one or more mixes at the newspaper. Thus, every news app user acts as a potential whistleblower and becomes inconspicuous in the crowd. We fortify this by ensuring that the news app ordinarily sends a small amount of constant cover traffic to the mix (Cover-Node) hosted in the news organisation's infrastructure; the cover traffic is replaced with message contents when a whistleblower communicates with a reporter."*

From https://petsymposium.org/2022/files/papers/issue2/popets-2022-0035.pdf

This document outlines the results of a penetration test and *whitebox* security review conducted against CoverDrop, a novel approach for whistleblowers implemented by the CoverDrop team, within *Guardian News & Media Limited*, more commonly known as *The Guardian* newspaper in the UK[1].

It should be noted that *The Guardian* Android[2] and iOS[3] apps already implement the CoverDrop protocol at the time of writing. However, this assignment focused on the reference CoverDrop apps, which will become open source for other newspapers to fork and integrate, while *The Guardian* apps were out of scope during this exercise.

Another notable exception was that the journalist app, for journalists to interact with newspaper readers, was not available during this assignment, as it is still in development, hence an interim Signal implementation was assessed instead. It is strongly advised to extrapolate all findings, recommendations and guidance in this report to harden the upcoming journalist app prior to releasing it.

The project was solicited by the CoverDrop team, funded by the Open Technology Fund (OTF), and executed by 7ASecurity in January and February 2024. The audit team dedicated 44 working days to complete this assignment. Please note that this is the first penetration test for this project. Consequently, the identification of security weaknesses was initially expected to be easier during this assignment, as more vulnerabilities are typically identified and resolved after each testing cycle.

During this iteration the goal was to review the CoverDrop project as thoroughly as possible, to ensure users can be provided with the best possible security and privacy.

[1] https://www.theguardian.com/gmg
[2] https://play.google.com/store/apps/details?id=com.guardian&hl=en&gl=US
[3] https://apps.apple.com/us/app/the-guardian-live-world-news/id409128287

The methodology implemented was *whitebox*: 7ASecurity was provided with access to reference Android and iOS builds, a reference server, the AWS and Kubernetes infrastructure, documentation and source code. 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 necessary arrangements were in place by December 2023 and January 2024, 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 Slack channel. The CoverDrop team was helpful and responsive at all times, which facilitated the test for 7ASecurity, without introducing any unnecessary delays. 7ASecurity provided regular updates regarding the audit status and its interim findings during the engagement.

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

- WP1: Whitebox Tests against CoverDrop Protocol and Library Implementation
- WP2: Mobile Security tests against CoverDrop Implementation on Android & iOS
- WP3: Whitebox Tests against CoverDrop Implementation on Backend Services
- WP4: Whitebox Tests against CoverDrop Servers & Configuration via SSH
- WP5: Whitebox Tests against CoverDrop AWS & Kubernetes Infrastructure
- WP6: CoverDrop Lightweight Threat Model documentation
- WP7: Whitebox Tests against CoverDrop Supply Chain Implementation

The findings of the security audit (WP1-5) can be summarized as follows:

| *Identified Vulnerabilities* | *Hardening Recommendations* | *Total Issues* |
|---|---|---|
| *7* | *26* | *33* |

Please note that the analysis of the remaining work packages (WP6-7) is provided separately, in the following sections of this report:

- WP6: CoverDrop Lightweight Threat Model
- WP7: CoverDrop Supply Chain Implementation

Moving forward, the scope section elaborates on the items under review, while 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. Additionally, it provides mitigation or fix advice for follow-up actions by the development team.

Finally, the report culminates with a conclusion providing detailed commentary, analysis, and guidance. This includes insights related to the context, preparation, and general

impressions gained throughout this test. Additionally, it offers a summary of the perceived security posture of the CoverDrop framework.

# Scope

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

- **WP1: Whitebox Tests against CoverDrop Protocol and Library Implementation**
  - Audited Source Code: https://github.com/guardian/coverdrop
- **WP2: Mobile Security tests against CoverDrop Implementation on Android & iOS apps**
  - Audited Android Version: 1.0 - *com.theguardian.coverdrop*
  - Audited iOS Version: 1.0 - *uk.co.guardian.securemessaging*
- **WP3: Whitebox Tests against CoverDrop Implementation on Backend Services**
  - Audited URLs:
    - https://secure-messaging-msg-audit.guardianapis.com
    - https://secure-messaging.code.dev-guardianapis.com
    - https://secure-messaging-api-audit.guardianapis.com
- **WP4: Whitebox Tests against CoverDrop Servers, Infrastructure & Configuration via SSH**
  - Audited Reference Server: *host-01 (100.127.4.93)*
- **WP5: Whitebox Tests against CoverDrop AWS and on-prem Kubernetes Infrastructure**
  - Audited AWS Account: *648583952313*
  - On-premise Kubernetes Cluster
- **WP6: CoverDrop Lightweight Threat Model documentation**
  - As above
- **WP7: Whitebox Tests against CoverDrop Supply Chain Implementation**
  - 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. *COV-01-001*) for ease of reference and offers an estimated severity in brackets alongside the title.

## COV-01-001 WP2: Possible Phishing via StrandHogg 2.0 on Android *(Medium)*

**Retest Notes:** The CoverDrop team resolved this issue during the test and 7ASecurity confirmed that the fix is valid.

Testing confirmed that the CoverDrop reference Android app is currently vulnerable to a number of Task Hijacking attacks. The *launchMode* for the app-launcher activity is currently not set and hence defaults to *standard*[4], which mitigates Task Hijacking via *StrandHogg*[5] and other older techniques documented since 2015[6], while leaving the app vulnerable to *StrandHogg 2.0*[7]*.* This vulnerability affects Android versions 3-9.0[8] but was only patched by Google on Android 8-9[9]. Since the app supports devices from Android 8.x (API level 26), this leaves all users running unpatched Android 8.x-9.0 devices vulnerable.

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[10].

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 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*,

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

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.

However, in the case of StrandHogg 2.0, all exported activities **without** a *launchMode* of *singleTask* or *singleInstance* are affected on vulnerable Android versions[11].

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

**Affected File:**
*AndroidManifest.xml*

**Affected Code:**

```
<activity android:name="com.theguardian.coverdrop.MainActivity"
android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
</activity>
```

As can be seen above, the *launchMode* is not set and hence defaults to *standard*.

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

**PoC Demo:**
https://7as.es/CoverDrop_LGs11RcVNB2/Task_Hijacking_PoC.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

[11] https://www.xda-developers.com/strandhogg-2-0.../

*singleTask*). This will ensure continuous mitigation in *StrandHogg 2.0*[12] while improving security strength against older Task Hijacking techniques[13].

- 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[14].

**Affected File:**
*AndroidManifest.xml*

**Proposed Fix:**

```
<application android:theme="@style/Theme.CoverDropSample"
android:label="@string/app_name" android:icon="@mipmap/ic_launcher"
android:name="com.theguardian.coverdrop.CoverDropApplication" [...]
android:taskAffinity="">
<activity android:name="com.theguardian.coverdrop.MainActivity" android:exported="true"
android:launchMode="singleInstance">
          <intent-filter>
              <action android:name="android.intent.action.MAIN"/>
              <category android:name="android.intent.category.LAUNCHER"/>
          </intent-filter>
</activity>
```

[12] https://www.xda-developers.com/strandhogg-2-0-android-vulnerability-explained.../
[13] http://blog.takemyhand.xyz/2021/02/android-task-hijacking-with.html
[14] https://www.slideshare.net/phdays/android-task-hijacking

### COV-01-013 WP2: Messaging DoS via DNS Spoofing *(Medium)*

**Retest Notes:** The CoverDrop team resolved this issue during the test and 7ASecurity confirmed that the fix is valid.

It was found that the reference CoverDrop Android and iOS apps are vulnerable to DoS attacks via spoofing of the DNS domains used for sending messages. Malicious attackers, able to modify clear-text network communications (i.e. via open Wi-Fi without guest isolation, DNS rebinding, ISP MiTM, BGP hijacking), could leverage this weakness to prevent legitimate app users from accessing the system. Please note that the same issue applies to potential censorship or user detection attempts on a wider scale.

This issue was confirmed by changing the DNS settings on the test iOS and Android devices (i.e. to simulate DNS spoofing), so that they point to an attacker-controlled DNS server using *dnschef*[15], spoofing the domains as follows:

**Command:**

```
dnschef -i 192.168.0.15 --fakeip 127.0.0.1 --fakedomains
secure-messaging-api-audit.guardianapis.com,secure-messaging-msg-audit.guardianapis.com
--logfile dns_traffic.txt
```

**Output:**

```
[...]
(13:53:59) [*] DNSChef started on interface: 192.168.0.15
(13:53:59) [*] Using the following nameservers: 8.8.8.8
(13:53:59) [*] 192.168.0.225: proxying the response of type 'AAAA' for
secure-messaging-msg-audit.guardianapis.com
(13:53:59) [*] 192.168.0.225: cooking the response of type 'A' for
secure-messaging-msg-audit.guardianapis.com to 127.0.0.1
[...]
(14:06:11) [*] 192.168.0.225: proxying the response of type 'AAAA' for
dualstack.guardian.map.fastly.net
(14:06:11) [*] 192.168.0.225: proxying the response of type 'A' for
dualstack.guardian.map.fastly.net
```

As a consequence, the application becomes incapable of sending messages, thus leaving the user with a perpetual pending state:

[15] https://github.com/iphelix/dnschef

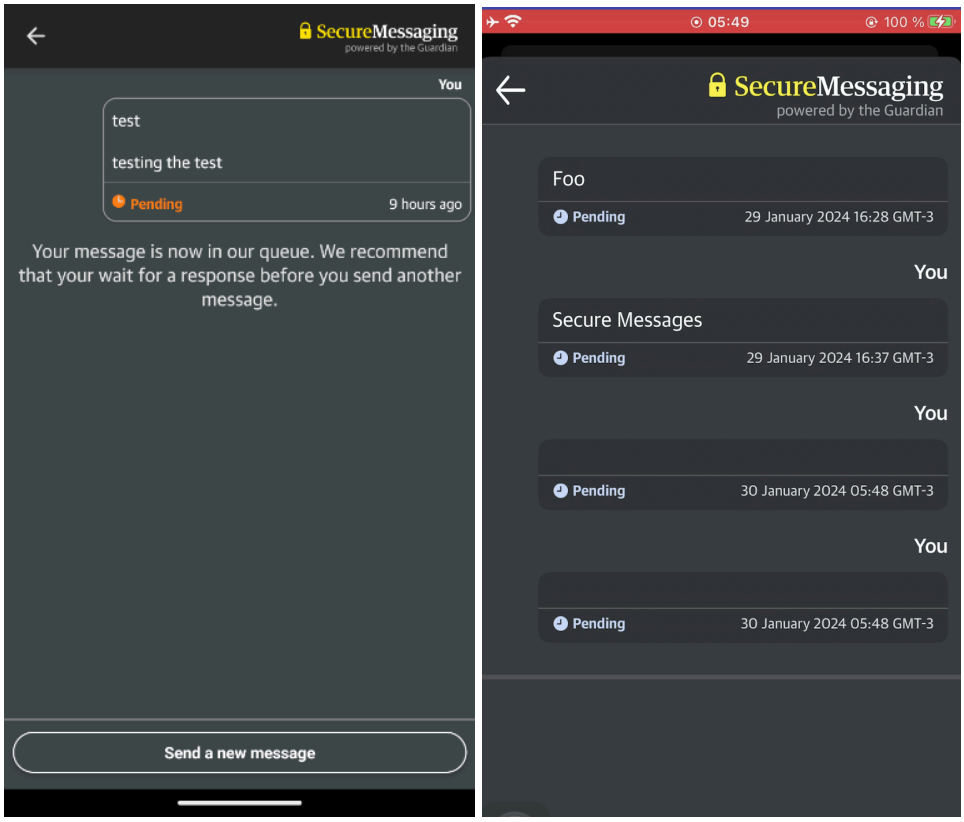



*Fig.: Pending message in Android (left) and iOS (right) via DNS spoofing*

This issue occurs due to building the messaging functionality on top of the insecure DNS protocol. An example of this can be seen in the output of the command below, which confirms the latest iOS protections available since iOS 14 are not being leveraged at the moment:

**Command:**

```
egrep -Ir '(NWParameters|PrivacyContext)' * | wc -l
```

**Output:**

```
0
```

It is recommended to switch over to *DNS over HTTPS* (*DoH*)[16] or *DNS over TLS* (*DoT*)[17] to mitigate these types of attacks. This will automatically remove all clear-text DNS resolution with its associated privacy and security problems, and instead encrypt all DNS traffic over HTTPS, this ensures DNS traffic will have the confidentiality and integrity protections offered by the HTTPS protocol thereafter. On iOS, since iOS 14 Apple allows developers to specify DoH connection parameters via the

[16] https://en.wikipedia.org/wiki/DNS_over_HTTPS
[17] https://www.cloudflare.com/learning/dns/dns-over-tls/

*NWParameters.PrivacyContext*[18][19] class. However, it is also possible to implement DoH at the app level in a compatible way with older iOS versions[20]. On Android, DoH can be easily deployed via the *okhttp-dnsoverhttps*[21] module, which has a Kotlin implementation available[22] and may, alternatively, be used as a reference.

Additionally, *Domain Fronting*[23] could be considered, this is a popular technique for Internet censorship circumvention that uses different domain names, employing different HTTPS communication layers. However, please note that Fastly, the content delivery network (CDN) used by CoverDrop services at the moment, plans to block domain fronting in February 2024[24].

### COV-01-015 WP2: Message Access via lack of Passphrase Prompt *(Medium)*

**Retest Notes:** The CoverDrop team resolved this issue during the test and 7ASecurity confirmed that the fix is valid.

It was discovered that the CoverDrop reference applications do not always prompt users for a passphrase prior to granting access to messages. Specifically, the reference iOS application will prompt the user for the passphrase, prior to showing the messages, after ~5 minutes in the background. Whereas the reference Android app does never appear to prompt users, if they background the app with the messaging screen open. A malicious attacker with access to an unlocked device could leverage this weakness to gain access to all user messages, particularly on Android, while the attack window is only ~5 minutes on iOS.

This issue can be confirmed by opening the messaging screen, sending the app into the background, and then opening the app again after a few hours (on Android):

[18] https://developer.apple.com/documentation/network/nwparameters/privacycontext
[19] https://gist.github.com/sschizas/ed03571ed129a227947f482b51ffabc5
[20] https://www.wwdcnotes.com/notes/wwdc20/10047/
[21] https://github.com/square/okhttp/tree/master/okhttp-dnsoverhttps
[22] https://github.com/square/okhttp/blob/master/okhttp-dnsoverhttps/.../dnsoverhttps/DnsOverHttps.kt
[23] https://en.wikipedia.org/wiki/Domain_fronting
[24] https://github.com/net4people/bbs/issues/309

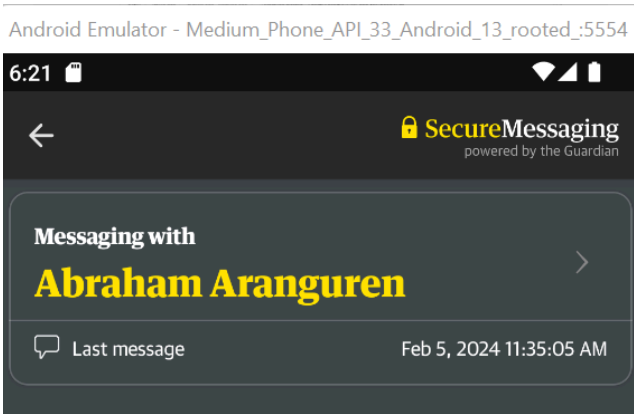



*Fig.: Access to Android messages via missing passphrase prompt*

It is recommended to leverage all the available platform protections to protect sensitive information, like confidential messages at rest. Furthermore, biometric authentication such as Face or Touch ID has not been implemented. Such mechanisms could be in place immediately after the user sends the application into the background, while the passphrase might still be required once a certain time threshold is reached (i.e. ~5 minutes). In turn, this would better protect user messages with multi-factor authentication (MFA), whereby, on the client-side, messages could be protected by both biometrics, as well as the passphrase.

### COV-01-021 WP4: File Access & Tampering via Insecure Permissions *(Medium)*

**Retest Notes:** The CoverDrop team resolved this issue and 7ASecurity confirmed that the fix is valid.

During the host hardening review, it was discovered that a number of files and directories have lax permissions in place. In the event that the CoverDrop server-side components are deployed into an environment with multiple users, such as shared hosting, malicious attackers might leverage this weakness to read and write many sensitive CoverDrop files. Please note that some of these files contain sensitive data in their filenames, such as the first and last names of the journalists. These issues can be confirmed as follows:

**Affected Host:**
*host-01 (100.127.4.93)*

**Issue 1: Globally Writable & Readable Files**

**Affected Files:**
`/data/coverdrop/covernode/keys/organization-b579e12e.pub.json`
`/data/coverdrop/covernode/keys/covernode_id-27b4e917.keypair.json`

```
/data/coverdrop/covernode/keys/covernode_msg-b081920f.keypair.json
/data/coverdrop/covernode/checkpoints/user_checkpoint.json
/data/coverdrop/covernode/checkpoints/journalist_checkpoint.json
/data/coverdrop/identity-api/keys/organization-b579e12e.pub.json
/data/coverdrop/identity-api/keys/covernode_provisioning-a98fad5c.keypair.json
/data/coverdrop/identity-api/keys/journalist_provisioning-6209d296.keypair.json
/data/coverdrop/signal-cli/accounts.json
/data/coverdrop/signal-cli/243359
/data/coverdrop/signal-bridge/vaults/abraham_aranguren.password
/data/coverdrop/signal-bridge/vaults/daniel_ortiz.password
/data/coverdrop/signal-bridge/vaults/harsh_bothra.password
/data/coverdrop/signal-bridge/vaults/miroslav_stampar.password
/data/coverdrop/signal-bridge/vaults/sam_cutler.password
/data/coverdrop/signal-bridge/vaults/szymon_grzybowski.password
/data/coverdrop/signal-bridge/vaults/abraham_aranguren.vault
/data/coverdrop/signal-bridge/vaults/daniel_ortiz.vault
/data/coverdrop/signal-bridge/vaults/harsh_bothra.vault
/data/coverdrop/signal-bridge/vaults/miroslav_stampar.vault
/data/coverdrop/signal-bridge/vaults/sam_cutler.vault
/data/coverdrop/signal-bridge/vaults/szymon_grzybowski.vault
```

### Example permissions:

### Command:

```
ls -altr /data/coverdrop/covernode/keys/organization-b579e12e.pub.json
```

### Output:

```
-rwxrwxrwx 1 user user 281 Dec 20 15:25
/data/coverdrop/covernode/keys/organization-b579e12e.pub.json
```

### Issue 2: Globally Writable & Readable Directories

### Affected Directories:

```
/data
/data/coverdrop
/data/coverdrop/covernode
/data/coverdrop/covernode/checkpoints
/data/coverdrop/covernode/keys
/data/coverdrop/identity-api/keys
/data/coverdrop/signal-cli
/data/coverdrop/signal-cli/243359.d
/data/coverdrop/signal-bridge
/data/coverdrop/signal-bridge/keys
/data/coverdrop/signal-bridge/account-data
/data/coverdrop/signal-bridge/account-data/243359.d
/data/coverdrop/signal-bridge/vaults
```

### Command:

```
ls -altrd /data/coverdrop
```

**Output:**

```
drwxrwxrwx 6 user user 4096 Dec 20 15:09 /data/coverdrop
```

**Issue 3: Globally Writable & Readable Database Files**

**Affected Files:**

```
/data/coverdrop/signal-bridge/signal.db
/data/coverdrop/signal-bridge/account-data/243359.d/account.db
/data/coverdrop/signal-cli/243359.d/account.db
```

**Command:**

```
ls -altr /data/coverdrop/signal-bridge/signal.db
```

**Output:**

```
-rwxrwxrwx 1 user user 57344 Feb  2 17:08 /data/coverdrop/signal-bridge/signal.db
```

**Issue 4: Bootloader with insecure Permissions**

The *grub.cfg* boot loader configuration file may contain security-relevant information, like the encrypted password for unlocking boot options, and should be restricted so that only the super user (*root*) can read it.

**Affected File:**

```
/boot/grub/grub.cfg
```

**Command:**

```
stat /boot/grub/grub.cfg
```

**Output:**

```
File: /boot/grub/grub.cfg
Size: 8990            Blocks: 24         IO Block: 4096   regular file
Device: 802h/2050d      Inode: 23          Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
[...]
```

Files and directories used to control jobs by the *cron* service are world-readable. Read access to the following files and directories could provide users with the ability to gain insight on system jobs.

**Issue 5: World-readable crontab File**

**Affected File:**

```
/etc/crontab
```

**Command:**

```
stat /etc/crontab
```

**Output:**

```
File: /etc/crontab
Size: 1136            Blocks: 8          IO Block: 4096   regular file
Device: fd01h/64769d    Inode: 1573551     Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
[...]
```

### Issue 6: World-readable cron.hourly Directory

**Affected Directory:**

```
/etc/cron.hourly/
```

**Command:**

```
stat /etc/cron.hourly/
```

**Output:**

```
File: /etc/cron.hourly/
Size: 4096            Blocks: 8          IO Block: 4096   directory
Device: fd01h/64769d    Inode: 1572888     Links: 2
Access: (0755/drwxr-xr-x)  Uid: (    0/    root)   Gid: (    0/    root)
[...]
```

### Issue 7: World-readable cron.daily Directory

**Affected Directory:**

```
/etc/cron.daily/
```

**Command:**

```
stat /etc/cron.daily/
```

**Output:**

```
File: /etc/cron.daily/
Size: 4096            Blocks: 8          IO Block: 4096   directory
Device: fd01h/64769d    Inode: 1572887     Links: 2
Access: (0755/drwxr-xr-x)  Uid: (    0/    root)   Gid: (    0/    root)
[...]
```

### Issue 8: World-readable cron.d Directory

**Affected Directory:**

```
/etc/cron.d/
```

**Command:**

```
stat /etc/cron.d/
```

**Output:**

```
File: /etc/cron.d/
Size: 4096            Blocks: 8          IO Block: 4096   directory
Device: fd01h/64769d    Inode: 1572886     Links: 2
Access: (0755/drwxr-xr-x)  Uid: (    0/    root)   Gid: (    0/    root)
[...]
```

### Issue 9: sshd_config File

The SSH *sshd_config* file contains the configuration of *ssh* service and should be protected from unauthorized access from non-privileged users.

**Affected File:**

```
/etc/ssh/sshd_config
```

**Command:**

```
stat /etc/ssh/sshd_config
```

**Output:**

```
File: /etc/ssh/sshd_config
Size: 3254            Blocks: 8          IO Block: 4096   regular file
Device: fd01h/64769d    Inode: 1574253     Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
[...]
```

It is recommended to implement the minimum possible permissions for the application to work. Specifically, CoverDrop files and directories should not be readable, writable or executable to unprivileged users on the same server. For the operating system files and directories, it is advised to change permissions as follows:

**Proposed Fix:**

```
chmod 400 /boot/grub/grub.cfg
chmod 600 /etc/crontab
chmod 700 /etc/cron.hourly/
chmod 700 /etc/cron.daily/
chmod 700 /etc/cron.d/
chmod 600 /etc/ssh/sshd_config
```

## COV-01-022 WP4: Data Access via Missing Encryption at Rest *(Medium)*

**Retest Notes:** The CoverDrop team resolved this issue and 7ASecurity confirmed that the fix is valid.

During the host review of the *host-01 (100.127.4.93)* server, it was found that some security-relevant data is unencrypted. The affected files contain Personally Identifiable Information (PII) of journalists, like their phone numbers and names. Furthermore, due to the insecure permissions reported on COV-01-021, this information may leak not only via server backups or to highly privileged users, but also to any malicious attacker who has an unprivileged user on the same server (i.e. in a shared hosting deployment scenario).

**Example 1: Leaks via Unencrypted Database**

**Command:**

```
# Connect to the SQLite signal.db database
sqlite3 /data/coverdrop/signal-bridge/signal.db
```

**Output:**

```
SQLite version 3.37.2 2022-01-06 13:25:41
Enter ".help" for usage hints.
sqlite>
```

**Command:**

```
# Select all data from the coverdrop_journalist_to_signal_journalist table
select * from coverdrop_journalist_to_signal_journalist;
```

**Output:**

```
s[...]r|+44[...]00
h[...]a|+91[...]22
a[...]n|+48[...]85
d[...]z|+54[...]96
m[...]r|+38[...]06
s[...]i|+48[...]60
```

Please note that, while chat information is encrypted, journalist PII is not:

**Command:**

```
sqlite> select * from group_chats limit 5;
```

**Output:**

```
|deVXiTZS1ts2REFIlgZB1fLfwPnXlNX[...]+UNELYk=|abraham_aranguren|0
[...],>|Pb3PjO9kjBMHqt9qR6lo9A==|s[...]|0
[...]p5IcVOAWpShoYcr3hCV4=|daniel_ortiz|0
[...]}
[...]|RbDIwftWvB5ElC1lpwWaMyywXRG/MGK0geeO2uL5ts4=|abraham_aranguren|0
```

```
[...]|JQ3yOJiv4N1C2Z4H8x8fnjmrQjm4HfVQBOkC78VUEG0=|daniel_ortiz|0
sqlite>
```

It is recommended to encrypt SQLite databases. This may be achieved using *SQLite Encryption Extensions (SEE)*[25], such as *SQLCipher*[26].

**Issue 2: Leaks via other Unencrypted Files**

A less significant finding was the leakage of passphrases in clear-text:

**Commands:**

```
user@host-01:/$ cat /data/coverdrop/signal-bridge/vaults/miroslav_stampar.password
user@host-01:/$ cat /data/coverdrop/signal-bridge/vaults/daniel_ortiz.password
```

**Output:**

```
crablike sash clothes regretful aside
marine raking plated flattop camper
```

However, this finding is less security-relevant as it pertains to an interim solution to get around the not-yet-implemented journalist client. This is perhaps best explained in the banner that is shown upon opening the *journalist.vault* file:

**Command:**

```
sqlcipher journalist.vault
```

**Output:**

*[...]*
*Because each vault is encrypted, a passphrase is needed to decrypt it. This is normally stored in a .password file upon vault creation. The password file is a simple txt file with the passphrase stored in cleartext.* ***When the Signal Bridge is eventually replaced by a journalist client, the .password file will no longer be used, and journalists will have to remember their passphrase.***

Nevertheless, it is recommended to ensure all files that contain sensitive information are encrypted at rest to avoid unintended leaks.

Regarding the storage of encryption keys on the server-side, it is crucial to avoid hard-coding them in the source code. While using environment variables is better than hard-coding secrets, they still have downsides[27]. Therefore, it is recommended to employ a dedicated secret management tool. Ideally, applications should retrieve

[25] https://www.sqlite.org/see/doc/trunk/www/readme.wiki
[26] https://www.zetetic.net/sqlcipher/
[27] https://security.stackexchange.com/questions/197784/is-it-unsafe-to-use-env…

credentials from secure vaults such as *AWS Secrets Manager*[28], *HashiCorp Vault*[29] or an equivalent secure vault that provides the application with credentials at runtime, while they remain encrypted at rest. In this particular case, a self-hosted password vault might increase resilience against high profile attackers, such as a government, able to ask cloud providers to hand over information. This approach allows applications to use credentials without exposing them to potential adversaries who may have access to leaked source code, developer machines, or other vulnerabilities. Additionally, it is advised to generate credentials, secrets, and API keys randomly to mitigate the potential for brute force or password-guessing attacks. For further mitigation guidance, please refer to the *OWASP Cryptographic Storage Cheat Sheet*[30] and the *CWE-798: Use of Hard-coded Credentials* page[31].

More broadly, it is important to emphasize the importance of having appropriate processes in place to:

- Regularly rotate credentials
- Revoke and replace credentials in the event of a compromise

### COV-01-023 WP2: Passphrase Access via Memory Leak *(Medium)*

**Retest Notes:** The CoverDrop team resolved this issue during the test and 7ASecurity confirmed that the fix is valid.

It was found that the reference CoverDrop Android app keeps the user passphrase in memory. This approach is insecure because that information could be accessed by a malicious attacker with physical access or memory access. Furthermore, given the large volume of publicly known Android kernel vulnerabilities[32] 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.

To confirm this issue, filesystem usage was reviewed but no sensitive information was found. Hence, subsequently the app process memory was dumped and the contents reviewed for possible leaks. In particular, a search for the user passphrase discovered an occurrence in memory.

**Command:**

```
grep "mace tropical ripening safari campus" android_mem.dump.strings | wc -l
```

**Output:**

[28] https://aws.amazon.com/.../aws-secrets-manager-store-distribute-and-rotate-credentials.../
[29] https://www.vaultproject.io/
[30] https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html
[31] https://cwe.mitre.org/data/definitions/798.html
[32] https://www.cvedetails.com/vulnerability-list.php?vendor_id=1224&product_id=19997...

```
1
```

This was further confirmed, using *Eclipse Memory Analyzer*[33] to inspect the Memory Dump as follows:

**Command:**

```
SELECT * FROM com.theguardian.coverdrop.core.crypto.Passphrase
```

**Output:**

```
com.theguardian.coverdrop.core.crypto.Passphrase @ 0x13a40278
|- <class>, shadow$_klass_ class com.theguardian.coverdrop.core.crypto.Passphrase @
0x14315298
|- passphrase java.lang.String @ 0x13c1b048  \u7261\u7567\u6261\u656c
|  |- <class>, shadow$_klass_ class java.lang.String @ 0x6fe8ddf8 Unknown, System
Class, JNI Global
|  |- value byte[8] @ 0x13c1b058  arguable
|  '- Total: 2 entries
'- Total: 2 entries
```

The root cause for this issue appears to be in the following code path, which uses a *java.lang.String* object to store the passphrase in memory:

**Affected Code:**

```
data class Passphrase(val passphrase: String) {
    fun getWords(): List<String> = passphrase.split(" ")

    companion object {
        fun fromWords(words: List<String>): Passphrase =
Passphrase(words.joinToString(" "))
    }
}
```

To resolve this issue, at a minimum, the user passphrase should be regularly wiped from memory to avoid potential leaks. Additionally, sensitive data like encryption keys, should not be retained in RAM longer than necessary. Variables storing keys ought to be nullified after use. Immutable objects like *java.lang.String* should be avoided for sensitive information, opting for char arrays instead. Even after removing or nullifying references to immutable objects, they might persist in memory until garbage collection, which apps are unable to enforce. For additional mitigation guidance, please see the *Testing Memory for Sensitive Data* section of the *Mobile Application Security Testing Guide (MASTG)*[34].

[33] https://eclipse.dev/mat/downloads.php
[34] https://mas.owasp.org/MASTG/tests/android/MASVS-STORAGE/MASTG-TEST-0011/

### COV-01-028 WP3/5: Server DoS via Insecure Signal-CLI Configuration *(Medium)*

**Retest notes:** The CoverDrop team resolved this issue and 7ASecurity confirmed that the fix is valid.

The on-premise Kubernetes environment hosts a *signal-cli*[35] container to headlessly communicate with Signal[36] infrastructure. It was discovered that the current configuration allows attachments. Hence a malicious actor, with knowledge of the number associated with the Signal account in the backend, may trivially send multiple max size attachments (*100MB*), which will be saved within the ephemeral *rootfs* volume of the container. Exhausting disk space may in turn trigger the kubernetes eviction policy, and termination of the container, thus disrupting the availability of the messaging bus.

**Affected Resources:**
*Kubernetes Cluster (pod signal-cli)*
*git: coverdrop/infra/on-premises/base/signal-cli-deployment.yaml*

This issue can be confirmed by observing the arguments passed to the *signal-cli*:

**Command:**

```
cat signal-cli-deployment.yaml
```

**Output:**

```
apiVersion: apps/v1
kind: Deployment
[...]
      containers:
        - name: signal-cli
          command: ["./signal-cli"]
          args:
            - "--output=json"
            - "--verbose"
            - daemon
            - "--tcp=$(SIGNAL_CLI_ADDRESS)"
            - "--receive-mode=manual"
            - "--no-receive-stdout"[...]
```

It is recommended to add the *--ignore-attachments* and *--ignore-stories* arguments to prevent saving unnecessary data in the backend, as described in the *signal-cli* documentation[37]. Furthermore, the production environment ought to remove verbose logging and handle potential rate-limiting conditions gracefully.

[35] https://github.com/AsamK/signal-cli
[36] https://signal.org/
[37] https://github.com/AsamK/signal-cli/blob/master/man/signal-cli.1.adoc

Alternatively, if attachments are perceived as a necessary feature, their maximum size should be limited and the signal bridge may monitor disk usage to detect and prevent potential attacks via excessive disk consumption.

# Hardening Recommendations

This report area 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.

## COV-01-002 WP2: iOS Binary Hardening Recommendations *(Info)*

**Retest Notes:** The CoverDrop team resolved this issue during the test and 7ASecurity confirmed that the fix is valid.

It was found that the iOS binary in the reference CoverDrop iOS app fails to leverage some code injection, privilege escalation, and anti-tampering flags available. While this is not a serious issue on its own, it could facilitate code injection, privilege escalation, and application tampering attacks in edge-case scenarios.

**Issue: Missing *__RESTRICTED* segment**

The binary does not have a restricted segment that prevents dynamic loading of *DYLIB* for arbitrary code injection.

**Command:**

```
size -x -l -m referenceAudit  |  grep __RESTRICT  | wc -l
```

**Output:**

```
0
```

It is recommended to use the following compiler options to enable the restricted segment feature:

**Proposed fix (compiler options):**

```
-Wl,-sectcreate,__RESTRICT,__restrict,/dev/null
```

### COV-01-003 WP1: Multiple Vulnerabilities in Rust Crates *(Low)*

**Retest Notes:** The CoverDrop team resolved this issue during the test and 7ASecurity confirmed that the fix is valid.

It was found that the CoverDrop library makes use of Rust crates with publicly known vulnerabilities. While most of these weaknesses are likely not exploitable under the current implementation, this is still a bad practice that could result in unwanted security issues. The following table summarizes publicly known weaknesses affecting Rust files:

| Library | Details |
|---|---|
| h2@0.3.19 | **Affected by:** Resource exhaustion vulnerability which may lead to Denial of Service (DoS)[38].<br>**Affected File**: *Cargo.lock* |
| libsqlite3-sys@0.24.2 | **Affected by:** libsqlite3-sys via C SQLite CVE-2022-35737[39].<br>**Affected File**: *Cargo.lock* |
| time@0.1.45 | **Affected by:** Potential segfault[40].<br>**Affected File**: *Cargo.lock* |
| sodiumoxide@0.2.7 | **Affected by:** sodiumoxide is deprecated[41].<br>**Affected File**: *Cargo.lock* |
| atty@0.2.14 | **Affected by:** Potential unaligned read[42].<br>**Affected File**: *Cargo.lock* |

In addition to upgrading outdated dependencies to the current versions, it is recommended to implement an automated task and/or commit hook to regularly check for vulnerabilities in dependencies. Some solutions that could help in this area are the *cargo update*[43] and *cargo audit fix*[44] commands, the *Snyk* tool[45], and the *OWASP Dependency Check* project[46]. Ideally, such tools should be run regularly by an automated job that alerts a lead developer or administrator about known vulnerabilities in dependencies so that the patching process can start in a timely manner.

[38] https://rustsec.org/advisories/RUSTSEC-2024-0003
[39] https://rustsec.org/advisories/RUSTSEC-2022-0090
[40] https://rustsec.org/advisories/RUSTSEC-2020-0071
[41] https://rustsec.org/advisories/RUSTSEC-2021-0137
[42] https://rustsec.org/advisories/RUSTSEC-2021-0145
[43] https://doc.rust-lang.org/cargo/commands/cargo-update.html
[44] https://crates.io/crates/cargo-audit
[45] https://snyk.io/
[46] https://owasp.org/www-project-dependency-check/

### COV-01-004 WP1: Potential for MitM via Downgraded TLS Version *(Info)*

**Retest note:** The CoverDrop team resolved this issue and 7ASecurity confirmed that the fix is valid.

During the code review, while inspecting the Rust toolchain installation, it was found that the CoverDrop library setup scripts restrict the TLS version to v1.2, which is considered to be adequately safe at the time of writing. Although some theoretical attacks exist against v.1.2[47][48], TLS v1.3 is considered to be faster and more secure[49], and hence it should be used instead. This was confirmed in the following code locations:

**Affected Files:**
*scripts/setup.sh*

*docker/dev/signal-cli/Dockerfile*

**Affected Code:**

```
# Install Rust toolchain
echo "👉 Checking if Rust is installed..."
if ! [[ -x "$(command -v cargo)" ]]; then
    echo "❌ Rust not found."
    if ! [[ -x "$(command -v curl)" ]]; then
        echo "❌ curl not found. Please check https://www.rust-lang.org/tools/install"
        exit 1
    else
        curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
        source "$HOME/.cargo/env"
        echo "✅ Finished installing Rust."
    fi
[...]
fi
```

It is recommended to switch directly to TLSv1.3, which is now widely supported. Additionally, the backend server supports TLS1.3[50], while *curl* supports TLSv1.3 since 2016[51].

[47] https://www.ssl.com/blogs/raccoon-attack-targets-tls-1-2-and-earlier-but-is-difficult-to-exploit/
[48] https://access.redhat.com/articles/2112261
[49] https://www.cloudflare.com/learning/ssl/why-use-tls-1.3/
[50] https://domsignal.com/test/jeed5c8r08o2eoewf3v7nsd3ngn44hwm
[51] https://curl.se/mail/lib-2016-12/0091.html

### COV-01-005 WP2: Multiple Vulnerabilities in Android packages *(Low)*

**Retest Notes:** The CoverDrop team resolved this issue during the test and 7ASecurity confirmed that the fix is valid.

It was found that the CoverDrop Android module makes use of Android packages with publicly known vulnerabilities. While most of these weaknesses are likely not exploitable under the current implementation, this is still a bad practice that could result in unwanted security issues. The following table summarizes publicly known weaknesses affecting Android packages:

| Library | Details |
|---|---|
| com.google.dagger:hilt-compiler@2.48.1 | **Affected by:** Creation of Temporary File in Directory with Insecure Permissions [52]<br>**Affected File:** *android/gradle/libs.versions.toml* |
| org.jetbrains.kotlin:kotlin-stdlib@1.8.10 | **Affected by:** Information Exposure [53]<br>**Affected File:** *android/build.gradle* |
| libsqlite3-sys@0.24.2 | **Affected by:** Information Exposure [54]<br>**Affected File:** *android/gradle/libs.versions.toml* |

The following command can be used to obtain a summary of vulnerable packages:

**Command:**

```
➜  android git:(main) ✗ snyk test .  --all-sub-projects
```

It is recommended to extrapolate the mitigation guidance offered under COV-01-003 to resolve this issue.

[52] https://security.snyk.io/vuln/SNYK-JAVA-COMGOOGLEGUAVA-5710356
[53] https://security.snyk.io/vuln/SNYK-JAVA-ORGJETBRAINSKOTLIN-2393744
[54] https://security.snyk.io/vuln/SNYK-JAVA-ORGJETBRAINSKOTLIN-2393744

### COV-01-006 WP3: TLS Hardening Recommendations (*Info*)

**Note:** The CoverDrop team plans to address this issue in the next release.

It was found that the TLS configuration of the CoverDrop API servers has minor weaknesses that could be resolved. While these issues do not constitute any significant security finding at present, they might become serious as new attacks continue to be discovered and fall into the public domain. Furthermore, these misconfigurations may facilitate Man-In-The-Middle (MitM) attacks against outdated clients.

The TLS configuration was found to support a number of weak TLS1.2 ciphers:

- *TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256*
- *TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA*
- *TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA*
- *TLS_RSA_WITH_AES_128_GCM_SHA256*
- *TLS_RSA_WITH_AES_256_GCM_SHA384*
- *TLS_RSA_WITH_AES_256_CBC_SHA*
- *TLS_RSA_WITH_AES_128_CBC_SHA*
- *TLS_RSA_WITH_3DES_EDE_CBC_SHA*

**PoC URLs:**
https://www.ssllabs.com/ssltest/analyze.html?d=secure%2dmessaging%2dapi.guardianapis.com&s=151.101.1.111&hideResults=on
https://www.ssllabs.com/ssltest/analyze.html?d=secure%2dmessaging%2dapi%2daudit.guardianapis.com&s=151.101.1.111&hideResults=on
https://www.ssllabs.com/ssltest/analyze.html?d=secure%2dmessaging%2dmsg%2daudit.guardianapis.com&s=151.101.1.111&hideResults=on
https://www.ssllabs.com/ssltest/analyze.html?d=secure%2dmessaging.code.dev%2dguardianapis.com&s=151.101.1.111&hideResults=on

It is recommended to deploy TLS correctly to solve these problems[55]. This should be done on all servers, including those that were out of scope during this assignment. The *OWASP TLS Cheat Sheet*[56] is a valuable resource to do this. Ultimately, the *SSL Labs* website[57] can be helpful to verify the configuration when the website is reachable online. Alternatively, the *OWASP O-Saft* tool[58] may facilitate testing the TLS configuration of servers that are not reachable via the internet.

[55] https://docs.fastly.com/en/guides/working-with-hosts#specifying-acceptable-tls-cipher-suites
[56] https://cheatsheetseries.owasp.org/.../Transport_Layer_Protection_Cheat_Sheet.html
[57] https://www.ssllabs.com/ssltest/
[58] https://owasp.org/www-project-o-saft/

### COV-01-007 WP5: ELB Hardening Recommendations (*Low*)

**Retest Notes:** The CoverDrop team resolved this issue and 7ASecurity confirmed that the fix is valid.

The AWS infrastructure makes use of an *Elastic Load Balancer* (ELB) to expose services. It was found that minor hardening improvements can be applied to improve its configuration. Specifically, the load balancer does not remove invalid headers, fails to leverage the AWS WAF, and has no logging configuration.

**Affected Resources:**
*AWS Account 648583952313*

**Issue: Security attributes and integrations disabled in all ELBs**

This issue can be confirmed reviewing the ELB security settings as follows:

1. Navigate to the *Elastic Load Balancers* view, on the *AWS Management Console* under the *EC2* category:
   **URL:**
   https://eu-west-1.console.aws.amazon.com/ec2/home?region=eu-west-1#LoadBalancer:loadBalancerArn=arn:aws:elasticloadbalancing:eu-west-1:648583952313:loadbalancer/app/secure-LoadB-5GQZ2lZ2MagZ/8c558a216e1999bc;tab=attributes
2. Click on the *Integrations* (*WAF* and *Config* settings) and then the *Attributes* (*logging* and *drop invalid headers*) tabs.

**Result:**
The *Attributes* and *Integrations* tabs fail to enable *Access logs*, the *Drop Invalid Headers* option, as well as the *AWS WAF* and *AWS Config* integration:

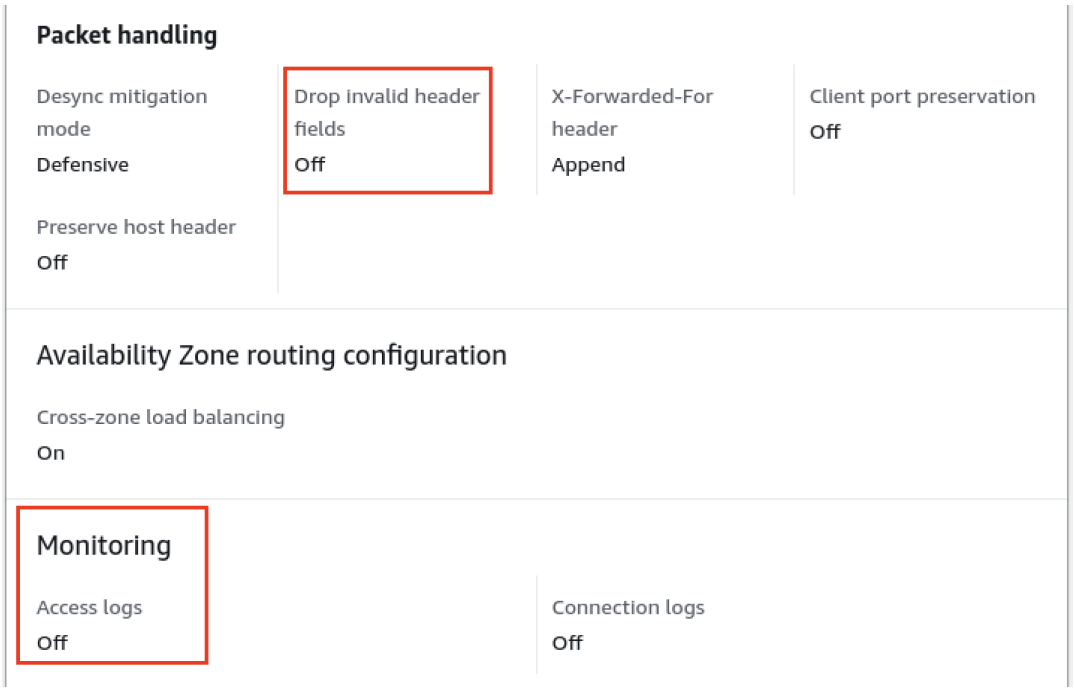



*Fig.: "Drop Invalid Header" & "Access logs" options disabled*

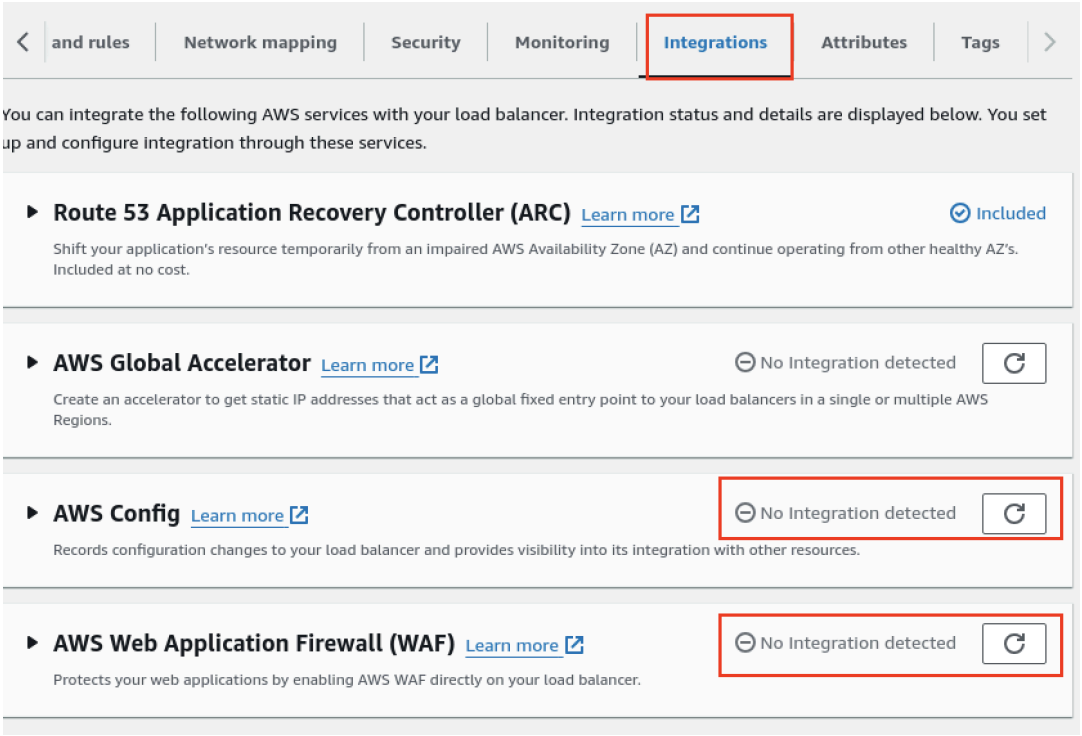



*Fig.: AWS Config & AWS WAF integrations disabled for sample ELB*

It is recommended to enable the *Drop Invalid Headers* option[59], enable the *AWS Config*[60] integration and consider deploying the AWS WAF[61]. Please note that implementing the *Drop Invalid Headers* and *AWS WAF* options will provide some protection against *HTTP Smuggling* and other web-based attacks. It is also advised to collect access logs at the ELB-level, if more comprehensive insight is needed to analyze HTTP requests coming from the Internet, however in most cases application-level logs should be sufficient.

### COV-01-008 WP5: AWS Weaknesses in Vuln Management Processes *(Low)*

**Retest Notes:** The CoverDrop team resolved this issue and 7ASecurity confirmed that the fix is valid.

During the configuration audit of the AWS account, it was discovered that some AWS security-relevant services are not configured correctly. Failure to leverage these services can leave the infrastructure open to attacks due to insufficient hardening.

**Affected Resources:**
*AWS Account 648583952313*

Please note that, as most of the AWS services are region-based, it is important to determine which regions are used first, to focus the analysis on the regions that are actually in use. The regions with defined resources in the analyzed environment are: *eu-west-1* and *us-east-1*. The latter however seems to be potentially unused and was described separately.

**Issue 1: *AWS Config* is not enabled in all used regions**

*AWS Config*[62] is a service that maintains the configuration history for AWS resources and evaluates best practices. The following command can be used to confirm *AWS Config* is not enabled on the used region:

**Command:**

```
aws configservice get-status
```

**Output:**

```
Configuration Recorders:
Delivery Channels:
```

[59] https://docs.aws.amazon.com/config/latest/developerguide/alb-http-drop-invalid-header(...).html
[60] https://docs.aws.amazon.com/config/latest/developerguide/elb-logging-enabled.html
[61] https://docs.aws.amazon.com/waf/
[62] https://aws.amazon.com/blogs/mt/aws-config-best-practices/

**Issue 2: *Security Hub* is not enabled in *us-east-1***

*Security Hub*[63] is a region-based service that provides a comprehensive view of security issues from regions where it is enabled. The following command describes the status of *Security Hub* for the regions in use:

**Command:**

```
aws securityhub --region us-east-1 describe-hub
```

**Output:**

```
An error occurred (InvalidAccessException) when calling the DescribeHub operation:
Account 648583952313 is not subscribed to AWS Security Hub
```

On the other hand, in the main region (*eu-west-1*), the service is enabled.

**Command:**

```
aws securityhub --region eu-west-1 describe-hub
```

**Output:**

```
{
    "HubArn": "arn:aws:securityhub:eu-west-1:648583952313:hub/default",
    "SubscribedAt": "2022-10-24T14:44:37.989Z",
    "AutoEnableControls": true,
    "ControlFindingGenerator": "STANDARD_CONTROL"
}
```

It is recommended to implement as many AWS Security related services as possible. This should include tools like *Security Hub*[64], *Config*[65], *Guard Duty*[66] and if possible *Macie*[67] and *Inspector*[68]. After this, the infrastructure team should ensure that all relevant services, and equivalent products, are enabled for the whole environment in all used regions. Furthermore, any reported issues should be regularly reviewed and remediated. This should ideally be accomplished by leveraging an *infrastructure-as-code* approach such as *Terraform*[69], which would significantly simplify applying the same settings across all AWS accounts and regions. Please note that cloud-native security tools are not perfect, however they provide a solid baseline for each environment. Special

[63] https://docs.aws.amazon.com/securityhub/latest/userguide/securityhub-get-started.html
[64] https://aws.amazon.com/security-hub/
[65] https://docs.aws.amazon.com/config/latest/developerguide/security-best-practices.html
[66] https://docs.aws.amazon.com/guardduty/latest/ug/what-is-guardduty.html
[67] https://aws.amazon.com/macie/
[68] https://docs.aws.amazon.com/inspector/v1/userguide/inspector_introduction.html
[69] https://www.terraform.io/use-cases/infrastructure-as-code

consideration should be given to *Security Hub* and *Config*, as they allow to streamline and discover common misconfigurations.

### COV-01-009 WP5: Possible risks via unused AWS Region *(Info)*

**Retest Notes:** The CoverDrop team resolved this issue and 7ASecurity confirmed that the fix is valid.

The AWS account mainly uses the *eu-west-1* region. However, it was discovered that some resources like *VPC*, *Security Groups and CloudFormation Stacks*, *RDS SubnetGroups* and an *S3 bucket* are present in the *us-east-1* region. If the region is actively used, multiple monitoring options should be enabled, i.e. *Security Hub*, *VPC Flow Logs*, etc. Failure to do this, may result in some security breaches going unnoticed.

**Affected Resources:**
*AWS Account 648583952313*

**Example 1: VPC defined in us-east-1**

**Command:**

```
# List VPCs in a given region
aws ec2 --region us-east-1 describe-vpcs
```

**Output:**


```
{
    "Vpcs": [
        {
            "CidrBlock": "172.31.0.0/16",
            "DhcpOptionsId": "dopt-0bcd10b8ca6e2f646",
            "State": "available",
            "VpcId": "vpc-04761d6f9341bb24a",
            "OwnerId": "648583952313",
            "InstanceTenancy": "default",
            "CidrBlockAssociationSet": [
                {
                    "AssociationId": "vpc-cidr-assoc-00453aa0a7d1582d8",
                    "CidrBlock": "172.31.0.0/16",
                    "CidrBlockState": {
                        "State": "associated"
                    }
                }
            ],
            "IsDefault": true
        }
    ]
}
```

**Example 2: CloudFormation Stacks in us-east-1**

**Command:**

```
# List cloudformation stacks in a given region
aws cloudformation list-stacks
    --region us-east-1
    --query StackSummaries[*].StackName
```

**Output:**

```
[
    "secure-collaboration-INFRA-aws-cost-monitor",
    "CoverDrop-CODE",
    "login-federation"
]
```

It is advised to monitor resources in all used regions. This way, in the event of a compromise, this may be a useful indicator when attackers successfully breach into the environment, as they might attempt to remain hidden in rarely used regions. In general, it is further recommended to remove all unnecessary regions and resources, to reduce the attack surface as much as possible.

### COV-01-010 WP5: Insufficient AWS Logging & Monitoring *(Low)*

**Retest Notes:** The CoverDrop team resolved this issue and 7ASecurity confirmed that the fix is valid.

It was found that the AWS account fails to log some important events, and does not implement robust security alerts for early attack detection. Without adequate logging and alerting, it may be impossible to monitor and detect malicious activities, or use integrated tools that analyze logs for anomalies, all of which may be critical in the event of a security breach.

**Affected Resources:**
*AWS Account 648583952313*

**Issue 1: No *VPC flow logs* defined**

No VPC flow logs were found to be defined. At a minimum, these should be listed for the VPCs with the main workloads. This can be confirmed by reviewing the VPC flow logs like so:

1. Open the *AWS Management Console*
2. Navigate to the *VPC Settings* and select a VPC to check.

**PoC URL:**
https://eu-west-1.console.aws.amazon.com/vpcconsole/home?region=eu-west-1#vpcs:

3. Review the *Flow Logs* tab.

The following command confirms there are no flow logs defined in the regions for the AWS accounts provided during this assignment:

**Command:**

```
# eu-west-1
aws ec2 describe-flow-logs
```

**Output:**

```
{ "FlowLogs": [] }
```

**Issue 2: No *CloudWatch* and *CloudTrail* integration and no security-related alerts**

The trail created in *CloudTrail* for the account does not have associated *CloudWatch* log groups. Effectively, there are no security-related metrics, as well as no security-related alerts based on the missing metrics. The only metrics and alerts configured in the system are related to typical daily operations of the administrative teams regarding the operational status of the application. Lack of *CloudTrail* and *CloudWatch* integration can be confirmed by reviewing the appropriate sections as follows:

1. Open the *AWS Management Console*
2. Navigate to the *AWS CloudTrail* and select the main trail.
   **PoC URL:**
   https://eu-west-1.console.aws.amazon.com/cloudtrail/home?region=eu-west-1#/trails/arn:aws:cloudtrail:eu-west-1:648583952313:trail/cloudtrail-Trail-Gn8iT6KGJps0
3. Review the *CloudWatch Logs* tab.

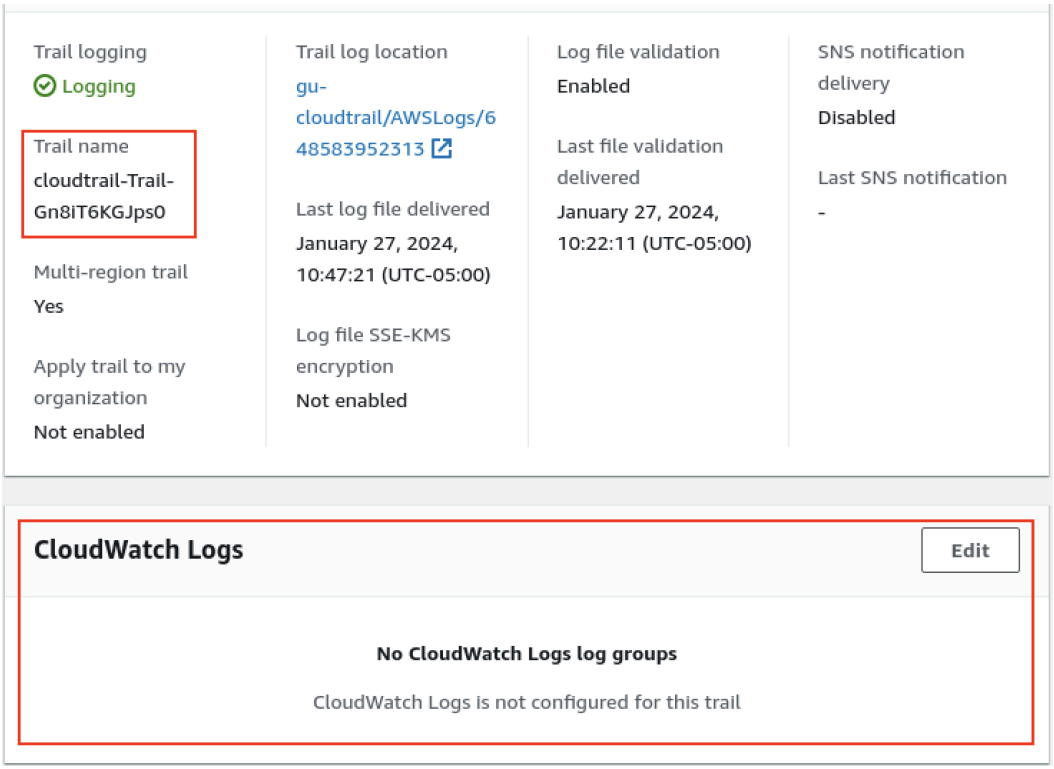



*Fig.: No associated log groups in CloudWatch*

It is recommended to enable *VPC Flow Logs* and ensure *CloudTrail* is integrated with *CloudWatch*. Metrics, alerts and notifications should then be defined[70] to detect anomalies early.

In general, all logging and monitoring settings should be adjusted depending on the threat model, compliance requirements and volume of generated data. Excessively verbose logs may increase the overall infrastructure cost significantly, however, lack of appropriate logging and monitoring decreases the chances of successful threat detection and analysis in case of a breach. It is advised to review and improve the logging and monitoring configuration in the context of a potential incident response case, rather than just regular daily operations of the infrastructure[71].

[70] https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudwatch-alarms-for-cloudtrail.html
[71] https://docs.aws.amazon.com/whitepapers/.../aws-security-incident-response...html

## COV-01-011 WP5: Insecure GitHub Token Storage in Parameter Store *(Low)*

**Retest Notes:** The CoverDrop team resolved this issue and 7ASecurity confirmed that the fix is valid.

It was found that a *GitHub Personal Token* belonging to a user in the organization is stored in the *AWS Parameter Store*, without sufficiently strong encryption properties. The parameter uses a default *KMS encryption key*, meaning users with read access can easily extract the token, and try to reach private resources in GitHub. While the token was found to have limited *read:packages* permissions attached, it can still be leveraged to download private docker containers from the *GitHub Container Registry* belonging to the organization. Please note the severity of this finding is reduced, as no containers were found to have sensitive data stored in docker layers. Nevertheless, attackers might be able to extract this type of information in the future, if it is added by mistake to some private container.

**Affected Resources:**
*AWS Account 648583952313*

Users with attached *ReadOnlyAccess* and *SecurityAudit* policies, can fetch the unencrypted token from the *AWS Parameter Store*, despite the *SecureString* property being assigned to the parameter, due to the default *KMS* key usage. This would not be possible if keys were managed with fine-grained policies instead.

**Issue 1: Token encrypted using the default KMS key in the AWS Parameter Store**

**Command:**

```
# Get the value of the parameter encrypted using the default KMS (alias/aws/ssm)
aws ssm get-parameter --name "/PROD/coverdrop/ghcr-token" --with-decryption
```

**Output:**

```
# Decrypted GitHub Personal Token
{
    "Parameter": {
        "Name": "/PROD/coverdrop/ghcr-token",
        "Type": "SecureString",
        "Value": "ghp_SpnZd8oo[...]",
        "Version": 1,
        "LastModifiedDate": "2023-11-13T10:38:16.423000-05:00",
        "ARN":
"arn:aws:ssm:eu-west-1:648583952313:parameter/PROD/coverdrop/ghcr-token",
        "DataType": "text"
    }
}
```

### Issue 2: Personal GitHub token used in the production environment

#### Command 1: GitHub Token Recon

```
# GitHub token verification
curl -sS -f -H "Authorization: token ghp_SpnZd[...]" https://api.github.com/user
```

#### Output:

```
# Object from GitHub API
{
  "login": "it[...]",
  [...]
  "name": "Sa[...]",
  "company": "The Guardian",
  "blog": "https://twitter.com/it[...]",
  [...]
  "created_at": "2013-09-27T18:43:22Z",
  [...]
}
```

#### Command 2: List Private Containers from ghcr.io

```
# GitHub API list containers for Guardian organization
curl -L -H "Authorization: token ghp_SpnZd[...]"
"https://api.github.com/orgs/guardian/packages?package_type=container" | jq '.[] |
select(.repository.private==true) | .name'
```

#### Output:

```
# Containers accessible to the user including some test resources
"coverdrop_kinesis"
"coverdrop_covernode"
"coverdrop_api"
"containers-experiment"
"container-experiments"
"coverdrop_key-expiry"
"coverdrop_journalist-identity-api"
"coverdrop_identity-api"
"coverdrop_cover-traffic"
"coverdrop_autoconfigure-api-ingress"
"test_coverdrop_kinesis"
"test_coverdrop_identity-api"
"test_coverdrop_covernode"
"test_coverdrop_api"
"test_coverdrop_fastly-edge"
"coverdrop_fastly-edge"
"coverdrop_signal-cli"
"coverdrop_signal-bridge"
```

It is recommended to invalidate the identified credentials, and use a separate machine account to access the *GitHub Container Registry* (GHCR). Limited permissions should be attached to expose only the necessary containers.

More broadly, secrets should be stored securely either in *AWS Secrets Manager*[72] or in the *AWS Parameters Store*[73], but in both cases sensitive data should additionally be protected utilizing a managed encryption key with fine-grained permissions. Preferably *CMK KMS*[74] should be used for full control over the encryption keys, and in case of cross-account data sharing.

### COV-01-012 WP5: Lack of S3 Bucket Hardening *(Low)*

**Client Note**: The CoverDrop team made some adjustments in this area, e.g. to enable bucket versioning. However to date they have not enabled MFA delete. They are planning before CoverDrop launches publicly to enable KMS encryption using service-specific keys to encrypt new objects in the bucket.

It was found that the *S3* buckets in scope use default encryption settings and lack adequate hardening options, such as *MFA Delete* and *Versioning*. Malicious attackers, with read access to the account, may leverage this weakness to download data from the buckets. If attackers gain write access, it may be possible to modify the content of the objects, which might be unnoticed by IT Staff. Furthermore, modification of binaries and scripts could lead to the compromise of *EC2* instances: As instances download scripts and binaries during *cloud-init* via *User-Data*, any malicious modifications to those files in *S3* may lead to the compromise of virtual machines.

**Affected Resources:**
*AWS Account 648583952313*

This issue can be confirmed navigating to the *S3 buckets* view on the *AWS Management Console*:

**Issue 1: Default AWS-managed KMS encryption key**

This issue can be confirmed reviewing any S3 bucket as follows:

1. Navigate to the *S3 buckets* view, on the *AWS Management Console*:
   **URL:** https://s3.console.aws.amazon.com/s3/buckets
2. Click on e.g. *coverdrop-dist-audit* S3 bucket to review the properties.

[72] https://aws.amazon.com/secrets-manager/features/
[73] https://docs.aws.amazon.com/kms/latest/developerguide/services-parameter-store.html
[74] https://docs.aws.amazon.com/systems-manager/latest/userguide/sysman-paramstore-access.html

**Result:**
The *Encryption type* section reveals the default *KMS* key:

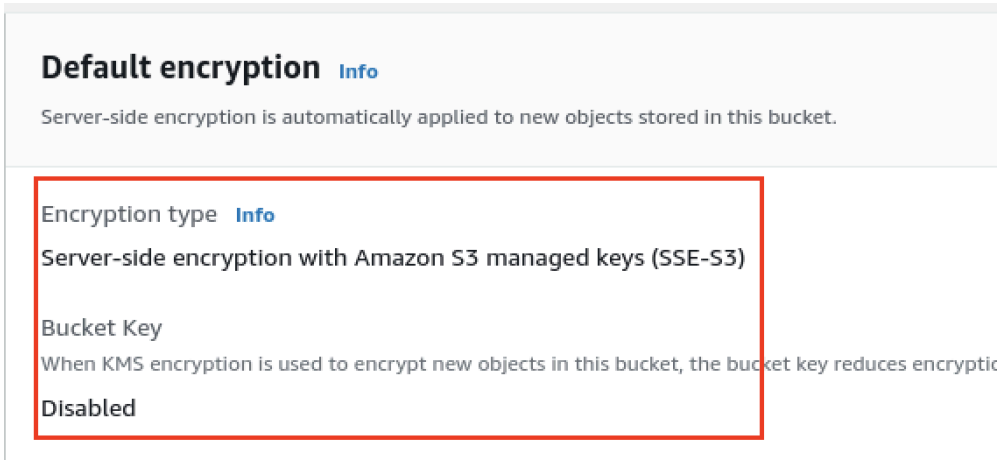



*Fig.: Default Amazon-managed encryption key*

**Issue 2: Lack of MFA delete and Bucket Versioning**

The *Bucket versioning* section reveals disabled options:

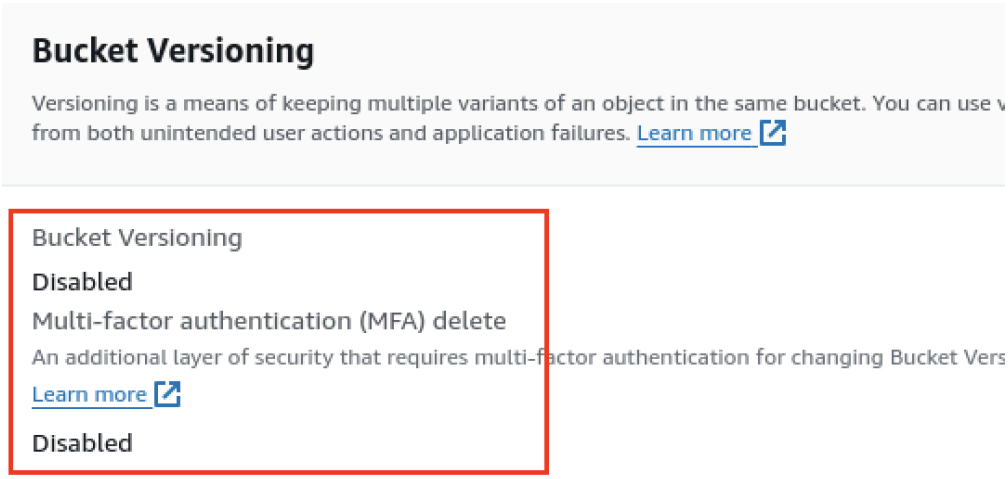



*Fig.: MFA delete and versioning options disabled*

It is recommended to use custom keys in the *AWS Key Management Service (AWS KMS)*[75] for bucket encryption. Ideally *CMK KMS* should be used for best control, and in case of cross-account sharing scenarios. Additionally options like *Versioning* and *MFA delete* should be enabled, especially for important buckets, like the ones containing binaries and scripts used by *EC2* instances. Finally, if possible, robust *object-level*

[75] https://docs.aws.amazon.com/AmazonS3/latest/userguide/UsingKMSEncryption.html

*logging*[76] should be implemented, to monitor and notify administrators if any unauthorized modifications of important data are detected, as this will contain potential security breaches early in the kill chain.

### COV-01-014 WP2: Android Config Hardening Recommendations *(Info)*

**Retest Notes:** The CoverDrop team resolved this issue during the test and 7ASecurity confirmed that the fix is valid.

It was found that the reference CoverDrop Android app fails to leverage optimal values for a number of security-related settings. While the CoverDrop team diligently mitigates the potential for backup leaks via backup exceptions, and no backup leaks could be found during this assignment, these settings increase the potential for leaks once the CoverDrop project becomes open source, and less educated developers fork and modify the CoverDrop applications in the future. These weaknesses are documented in more detail next.

**Issue 1: Undefined *android:hasFragileUserData***

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

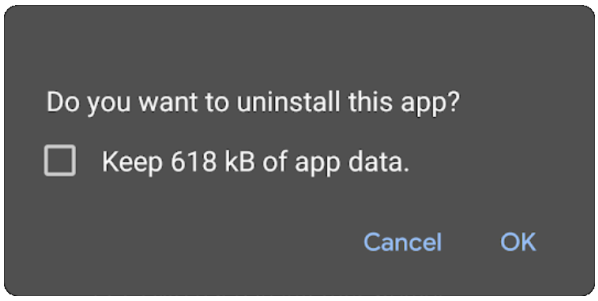



*Fig.: Uninstall prompt with check box for keeping the app data*

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.

[76] https://docs.aws.amazon.com/AmazonS3/latest/userguide/enable-cloudtrail-logging-for-s3.html

**Issue 2: Usage of *android:allowBackup=”true”***

**Note:** CoverDrop was not found to leak any sensitive information in backups and correctly sets backup exceptions. Nevertheless, this hardening setting is presented here as an informational hardening recommendation for third parties to be careful about, once they fork and modify the upcoming open source CoverDrop version.

The application currently allows backups which might result in local attackers with access to an unlocked device able to enable USB debugging and access application secrets:

**Affected File:**
*AndroidManifest.xml*

**Affected code:**

```
<application android:theme="@style/Theme.CoverDropSample"
[...]
android:taskAffinity="" android:allowBackup="true" android:largeHeap="true"
android:supportsRtl="true" android:extractNativeLibs="false"
[...]
android:appComponentFactory="androidx.core.app.CoreComponentFactory"
android:dataExtractionRules="@xml/data_extraction_rules">
```

It is recommended to use a value of *false* for *android:allowBackup*. If backups must be allowed, the *android:fullBackupContent* directive could be used to specify an XML file[77] with full backup rules for auto backup[78].

### COV-01-016 WP1: Insecure Zero-Padding in *PaddedCompressedString* *(Info)*

**Retest Notes:** The CoverDrop team resolved this issue during the test and 7ASecurity confirmed that the fix is valid.

It was found that the CoverDrop protocol implementation is using zero-padding[79] for message encapsulation. In cryptography, Zero padding refers to the practice of padding a message with zeros, to meet a specific block size requirement before encryption. While zero padding is a simple and widely used technique, it has some security vulnerabilities, especially in certain cryptographic scenarios[80].

Zero padding is deterministic, meaning that the same input will always result in the same padded output. This lack of variability can be exploited by attackers who might observe

[77] https://developer.android.com/guide/topics/manifest/application-element#fullBackupContent
[78] https://developer.android.com/guide/topics/data/autobackup
[79] https://en.wikipedia.org/wiki/Padding_(cryptography)#Zero_padding
[80] https://www.iacr.org/cryptodb/archive/2002/EUROCRYPT/2850/2850.pdf

repeated patterns in the ciphertext, potentially revealing information about the plaintext. Furthermore, zero padding does not provide any integrity protection for the padded message. Malicious attackers capable of modifying the ciphertext may be able to tamper the padding, leading to potential security issues. Even though this does not appear to be an exploitable vulnerability at the moment, it represents a bad security practice which should be addressed appropriately. This issue can be confirmed observing the following code paths:

**Affected Files:**
*common/src/padded_compressed_string.rs*
*ios/reference/CoverDropCore/Sources/CoverDropCore/PaddedCompressedString.swift*
*android/core/src/main/java/com/theguardian/coverdrop/core/models/PaddedCompressedString.kt*

**Affected Code:**

```
impl<const PAD_TO: LengthHeader> PaddedCompressedString<PAD_TO> {
const HEADER_SIZE: usize = size_of::<LengthHeader>();

/// The total length of this buffer, used when allocating arrays which need `usize`.
pub const TOTAL_LEN: usize = PAD_TO as usize;

pub fn new(text: &str) -> Result<PaddedCompressedString<PAD_TO>, Error> {
    // Allocate at the expected size up front.
    let mut buf: Vec<u8> = Vec::with_capacity(Self::TOTAL_LEN);

    // Reserve bytes which we'll need later for writing the size
    // This saves us from prepending later on, O(n), horrifying!
    buf.resize(Self::HEADER_SIZE, 0x0);

    let mut compression = GzEncoder::new(buf, Compression::default());
    compression.write_all(text.as_bytes())?;

    let mut buf = compression.finish()?;

    // Write the size of the compressed bytes into the header we reserved previously
    let compressed_size = LengthHeader::try_from(buf.len() - Self::HEADER_SIZE)
       .map_err(|_| Error::CompressedStringTooLong(buf.len() as f32 / PAD_TO as f32))?;
    // The compressed string is >2^16 bytes
    let compressed_size_bytes = compressed_size.to_be_bytes();

    buf[..Self::HEADER_SIZE].copy_from_slice(&compressed_size_bytes[..
        Self:: HEADER_SIZE]);

    if buf.len() > Self::TOTAL_LEN {
        Err(Error::CompressedStringTooLong(
            buf.len() as f32 / PAD_TO as f32,
        ))
    } else {
```

```
            buf.resize(Self::TOTAL_LEN, 0x0);  // NOTE: zero-padding happens here
            Ok(PaddedCompressedString(buf))
        }
    }
```

It is recommended to employ a modern cryptographic standard that implements more secure padding schemes, such as PKCS#7[81] or simple randomized padding. These provide better security properties and resistance to a broader range of cryptography attacks.

### COV-01-017 WP5: Possible Improvements to IAM Policies *(Low)*

**Retest Notes:** The CoverDrop team resolved this issue and 7ASecurity confirmed that the fix is valid.

While performing the AWS audit, It was discovered that some of the IAM policies fail to restrict access to a number of resources in the environment. As the environment mixes staging and production workloads, it is crucial to ensure boundaries are enforced to prevent pivoting between environments, in the event of a compromise.

**Affected Resources:**
*AWS Account 648583952313*

**Issue 1: Cross-Env Param Store Access via AmazonSSMManagedInstanceCore**

The following command using *PMapper*[82] reveals which entities have permissions to perform actions on a given resource. In this case, all EC2 instance roles, and the *cloudquery-access* role, were found to be authorized to read data from the *AWS Parameter Store*, regardless of the environment the parameter belongs to, based on the naming convention. Furthermore, for EC2 instance roles, it was found that the *AmazonSSMManagedInstanceCore* AWS-managed policy is attached to the role, granting broad access.

**Command:**

```
# Query to PMapper to retrieve who has access to sample sensitive parameter
who can do ssm:GetParameter with
arn:aws:ssm:eu-west-1:648583952313:parameter/CODE/secure-collaboration/signal-bridge/ac
count.pin
```

**Output:**

```
# Highlighted roles potentially should not be able to read any parameter
```

[81] https://node-security.com/posts/cryptography-pkcs-7-padding/
[82] https://github.com/nccgroup/PMapper

```
user/7ASecurity
role/AWSCloudFormationStackSetExecutionRole
role/cloudquery-access
role/janus
role/secure-collaboration-AUDI-InstanceRoleCovertrafficC-wew3fxOVjoe3
role/secure-collaboration-AUDIT--InstanceRoleApi6D8EAD0D-sOyg3BaZnuzX
role/secure-collaboration-CODE-c-InstanceRoleApi6D8EAD0D-mTe6oxCNcVxT
role/secure-collaboration-CODE-InstanceRoleCovernodeE7FC-0qSsjIVICPAE
role/secure-collaboration-CODE-InstanceRoleCovertrafficC-joP4PRfWPlks
role/secure-collaboration-CODE-InstanceRoleIdentityapi5B-6rmUNSDbeJJL
role/secure-collaboration-CODE-InstanceRoleSignalbridgeB-VLHitlW8xdTs
role/secure-collaboration-DEMO-c-InstanceRoleApi6D8EAD0D-t8lZ3WtNWalX
role/secure-collaboration-DEMO-InstanceRoleCovernodeE7FC-26LwY8S6ZLNR
role/secure-collaboration-DEMO-InstanceRoleCovertrafficC-QBjdHO2RuKu3
role/secure-collaboration-DEMO-InstanceRoleIdentityapi5B-BwDk3qEqNIOQ
role/secure-collaboration-DEMO-InstanceRoleSignalbridgeB-iKbEjZwHKVur
role/secure-collaboration-PROD-c-InstanceRoleApi6D8EAD0D-cqUpyAdIr17d
role/StackSet-RiffRaffAccess-3-RiffRaffCloudformationEx-P6ZILT0HFUT3
role/StackSet-RiffRaffAccess-3828f6-RiffRaffAccountRole-HEG6110C54G0
role/stacksets-exec-af8836dbce8a429fb93c2b7e225a10f0
```

**Issue 2: Broad Read Access for cloudquery-access role**

The *cloudquery-access* role, used by an external entity to query the environment, was found to have a full *ReadOnlyAccess* policy attached, with some minor restrictions introduced by an inline policy. Despite the restrictions, as listed in the previous issue, this allows any user assuming *cloudquery-access*, to retrieve values from the *AWS Parameter Store*, including those marked as *SecretString*. The following command can be used to list the attached policies:

**Command:**

```
# List policies attached to the role
aws iam list-attached-role-policies --role-name cloudquery-access
```

**Output:**

```
# ReadOnlyAccess policy attached to the role assumed from other aws
# account: 095768028460
{
    "AttachedPolicies": [
        {
            "PolicyName": "ReadOnlyAccess",
            "PolicyArn": "arn:aws:iam::aws:policy/ReadOnlyAccess"
        }
    ]
}
```

It is recommended to consider removing roles granting broad read access, and use restricted *KMS* policies, or customer managed *KMS* keys, for better control over

encrypted data. Tools such as *IAM Access Analyzer*[83] can then be used, to determine the smallest set of permissions needed to comply with the least privilege principle.

### COV-01-018 WP5: Insecure Cross-Account Integration *(Low)*

**Client note**: The CoverDrop team plans to address this issue in the next release.

The CoverDrop *648583952313* cloud environment is integrated with multiple AWS accounts. This includes, *717331877981 - Fastly*, an external one, while *100378408918 - Guardian Security, 095768028460 - Guardian deployTools*, and others are internal and part of the Guardian organization. It was found that multiple roles, used by the internal account integrated via IAM roles, do not prevent a confused deputy issue or can be leveraged to obtain administrative access. These weaknesses are described in more detail next:

**Affected Resources:**
*AWS Account 648583952313*

**Issue: Lack of ExternalID for role from CI/CD pipelines AWS account (*deployTools*)**

The *StackSet-RiffRaffAccess-3828f6-RiffRaffAccountRole-HEG6110C54G0* role can be assumed from *095768028460*, a shared AWS account used by multiple development teams. Such an account with shared services can be potentially used in *Confused Deputy Problem*[84] scenarios. For example, an attacker with access to another development team, might gain administrative access to the analyzed AWS account. The lack of *ExternalID* can be confirmed reviewing the trust policy settings as follows:

**Command:**

```
# List trust policy
aws iam get-role --role-name
StackSet-RiffRaffAccess-3828f6-RiffRaffAccountRole-HEG6110C54G0
```

**Output:**

```
# Trusted roles from shared CI/CD AWS account 095768028460
{   "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": [

"arn:aws:iam::095768028460:role/riff-raff-PROD-InstanceRole-1PZTOOPUBA63P",
```

[83] https://aws.amazon.com/iam/access-analyzer/
[84] https://cwe.mitre.org/data/definitions/441.html

```
"arn:aws:iam::095768028460:role/riff-raff-CODE-InstanceRole-1X7SE4IXFE6MU"
                ]
            },
            "Action": "sts:AssumeRole" }]}
```

It is worth mentioning that the external Fastly account integrated with Kinesis streams is correctly protected against this type of attack as follows:

**Command:**

```
# List trust policy for Fastly
aws iam get-role --role-name userToJournalistKinesisFastlyRole-PROD
```

**Output:**

```
# Limited Fastly role with ExternalId
{   "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::717331877981:root"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "sts:ExternalId": "Ga3[...]" }}}]}
```

Please note that other roles are also missing confused deputy protection. Hence, this also similarly weakens services deployed in other AWS accounts, which might be leveraged to gain access via *648583952313*, namely:

- *arn:aws:iam::648583952313:role/grafana-access*
- *arn:aws:iam::648583952313:role/cloudquery-access*

It is recommended to remediate the confused deputy problem by using the aforementioned *ExternalID* approach. It is important to do this, if possible, on all systems deployed in integrated AWS accounts. For additional mitigation and guidance, please see the *Confused Deputy* section of the AWS documentation[85].

[85] https://docs.aws.amazon.com/IAM/latest/UserGuide/confused-deputy.html

### COV-01-019 WP2: Android Binary Hardening Recommendations *(Info)*

**Client Note:** The CoverDrop team will not apply this hardening recommendation for the time being.

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

**Issue 1: Binaries missing usage of -D_FORTIFY_SOURCE=2**

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

**Example binaries (from decompiled production app):**
*x86/libsodium.so*
*x86_64/libjnidispatch.so*
*arm64-v8a/libjnidispatch.so*
*arm64-v8a/libtoolChecker.so*
*armeabi/libsodium.so*
*armeabi-v7a/libjnidispatch.so*
*[...]*

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

**Issue 2: Binaries missing usage of Stack Canary**

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

**Example binaries (from decompiled app):**
*lib/mips/libjnidispatch.so*
*lib/armeabi-v7a/libjnidispatch.so*
*lib/mips64/libjnidispatch.so*
*lib/x86/libjnidispatch.so*
*lib/arm64-v8a/libjnidispatch.so*
*lib/x86_64/libjnidispatch.so*
*[...]*

Regarding stack canaries, the *-fstack-protector-all* option can be used to allow the detection of overflows by verifying the integrity of the canary before function returns.

### COV-01-020 WP4: Boot Loader Password Not Set *(Low)*

**Retest Notes:** The CoverDrop team resolved this issue and 7ASecurity confirmed that the fix is valid.

It was found that no boot loader password is set on the *host-01 (100.127.4.93)* server. Having no boot loader password set, an unauthorized user with physical access to the server, may set command line boot parameters when booting the host. These parameters could be used to subvert the security of the system.

**Affected File:**
*/boot/grub/grub.cfg*

**Command:**
```
grep "password" /boot/grub/grub.cfg
```

**Output:**
```
(empty)
```

It is recommended to create an encrypted password by using the *grub-mkpasswd-pbkdf2*[86] command.

### COV-01-024 WP4: Weaknesses in Network Stack Configuration *(Low)*

**Retest Notes:** The CoverDrop team resolved this issue and 7ASecurity confirmed that the fix is valid.

During the host hardening audit, it was discovered that the network stack on *host-01 (100.127.4.93)* is configured with insecure settings. Specifically:

- *send_redirects* is enabled. An attacker could use a compromised host, to send invalid ICMP redirects to other router devices, corrupt routing, and have users access an attacker-controlled system instead of the intended one.
- *accept_redirects* is enabled. This enables processing of ICMP Redirect packets. ICMP redirects should not be needed in correctly operating networks and unnecessarily increase the attack surface.
- *secure_redirects* is enabled. This enables processing of ICMP Redirect packets, if the source address is a router.

[86] https://manpages.ubuntu.com/manpages/focal/en/man1/grub-mkpasswd-pbkdf2.1.html

- *log_martians* is not enabled. Enabling this feature and logging these packets allows administrators to investigate the possibility of attackers sending spoofed packets to the system.

This can be confirmed as follows:

**Affected File:**
*/etc/sysctl.conf*

**Commands:**
```
sysctl net.ipv4.conf.all.send_redirects
sysctl net.ipv4.conf.default.send_redirects
sysctl net.ipv4.conf.default.accept_redirects
sysctl net.ipv4.conf.all.secure_redirects
sysctl net.ipv4.conf.default.secure_redirects
sysctl net.ipv4.conf.all.log_martians
sysctl net.ipv4.conf.default.log_martians
```

**Output:**
```
net.ipv4.conf.all.send_redirects = 1
net.ipv4.conf.default.send_redirects = 1
net.ipv4.conf.default.accept_redirects = 1
net.ipv4.conf.all.secure_redirects = 1
net.ipv4.conf.default.secure_redirects = 1
net.ipv4.conf.all.log_martians = 0
net.ipv4.conf.default.log_martians = 0
```

It is recommended to disable the *send_redirects, accept_redirects* and *secure_redirects* settings in the */etc/sysctl.conf* file, and enable the *log_martians* setting. To set the runtime status of the aforementioned kernel parameters, it is recommended to run the following commands:

**Proposed Fix:**
```
sysctl -w net.ipv4.conf.all.send_redirects=0
sysctl -w net.ipv4.conf.default.send_redirects=0
sysctl -w net.ipv4.conf.default.accept_redirects=0
sysctl -w net.ipv4.conf.all.secure_redirects=0
sysctl -w net.ipv4.conf.default.secure_redirects=0
sysctl -w net.ipv4.conf.all.log_martians=1
sysctl -w net.ipv4.conf.default.log_martians=1
```

## COV-01-025 WP4: Weaknesses in SSH Server Access *(Low)*

**Retest Notes:** The CoverDrop team resolved this issue and 7ASecurity confirmed that the fix is valid.

During the host hardening audit, it was found that the configuration for a number of SSH options was not improved from the default settings. Additionally, the password policy has not been hardened. This makes the operating system more prone to brute force and resource exhaustion attacks. Other weaknesses include the usage of shared user accounts for access to the server, as well as missing Multi-Factor Authentication (MFA), both of which complicate the auditability of possible server compromises and increase the potential for unauthorized access to the server.

**Affected Host:**
*host-01 (100.127.4.93)*

**Issue 1: Default MaxAuthTries**

The *MaxAuthTries* parameter specifies the maximum number of authentication attempts permitted per connection. If the number of login failures reaches half the number, error messages will be added to the *syslog* file. Setting the value of *MaxAuthTries* parameter to a low number will minimize the risk of successful brute force attacks to the SSH server. The setting is currently not set, resulting in the default value of 6.

**Command:**
```
grep --color MaxAuthTries /etc/ssh/sshd_config
```

**Output:**
```
#MaxAuthTries 6
```

**Issue 2: ClientAliveInterval not set**

The *ClientAliveInterval* and *ClientAliveCountMax* settings control the timeout of SSH sessions. When the *ClientAliveInterval* variable is set, SSH sessions that have no activity for the specified length of time are terminated. When the *ClientAliveCountMax* variable is set, the SSH daemon will send alive messages at every *ClientAliveInterval* interval to the client. When the number of consecutive client alive messages are sent without response from the client, the SSH session is terminated. The *ClientAliveInterval* parameter is currently set to default value of 0, indicating the setting is disabled.

**Command:**
```
grep --color ClientAliveInterval /etc/ssh/sshd_config
```

**Output:**

```
#ClientAliveInterval 0
```

It is recommended to configure the *MaxAuthTries, ClientAliveInterval* and *ClientAliveCountMax* settings. This can be done by running the commands listed below. If the value of *ClientAliveCountMax* is left at the default, and the *ClientAliveInterval* is set to 15, unresponsive SSH clients will be disconnected after approximately 45 seconds.

**Proposed Fix:**

```
echo "MaxAuthTries 4" >> /etc/ssh/sshd_config
echo "ClientAliveInterval 15" >> /etc/ssh/sshd_config
```

**Issue 3: Lax Password Policy**

The password policy configuration is configured with unhardened default settings:

- *PASS_MAX_DAYS* is set to *99999*. The *PASS_MAX_DAYS* setting allows to force passwords to expire once they reach a defined age. The ability to leverage a compromised password is limited by the age of the password. Therefore, reducing the maximum age of the password *(PASS_MAX_DAYS)* also reduces the window of opportunity.
- *PASS_MIN_DAYS* is set to *0*. The *PASS_MIN_DAYS* setting Prevents users from changing their password until a minimum number of days have passed since the last time the user changed their password.

**Command:**

```
egrep "(PASS_MAX_DAYS|PASS_MIN_DAYS)" /etc/login.defs
```

**Output:**

```
PASS_MAX_DAYS   99999
PASS_MIN_DAYS   0
```

By restricting the frequency of password changes, administrators can prevent users from repeatedly changing their password in an attempt to circumvent password reuse controls.

For the SSH configuration, it is recommended to set the *PASS_MAX_DAYS* to *90* and *PASS_MIN_DAYS* to *7* and update the settings for existing account with the following commands:

**Proposed Fix:**

```
chage --maxdays 90 $user
chage --mindays 7 $user
```

**Issue 4: Non-personal User Account**

A non-personal account appears to be in use for daily tasks on the server. User accounts should be either service accounts or personal accounts. The primary reasons for this are:

- **Accountability**: When a security incident occurs and log files are examined it is important to be able to link activities to individuals instead of only a shared user account.
- **Shared Password Risks**: Shared user accounts require shared passwords, which often leads to various risks such as insecure passwords, passwords that are hard to change, and/or passwords that get stored in plain text files.

Non-personal and shared accounts should not be used by any user. It is recommended to use personal accounts instead.

**Issue 5: Missing MFA for SSH Access**

The reference host is currently missing *Multi Factor Authentication (MFA)* for SSH access.

It is recommended to implement MFA for SSH access. A possible way to accomplish this is by installing and configuring the *google-authenticator*[87] package.

### COV-01-026 WP4: Weaknesses in Auditing and OS-level Logging *(Low)*

**Retest Notes:** The CoverDrop team resolved this issue and 7ASecurity confirmed that the fix is valid.

During the host hardening audit, it was discovered that the server does not have *auditd* installed. When the Linux Auditing System *(auditd)* is installed and configured, it may capture security related events. It was also found that the server fails to implement an external log sink, although the *rsyslog* logging service is active. These weaknesses make the investigation of possible security breaches more difficult and might allow attackers to evade detection.

**Issue 1: Linux Audit System (auditd) not installed**

The following command shows that the *auditd* and *audispd-plugins* packages are not installed on the operating system.

**Command:**

[87] https://ubuntu.com/tutorials/configure-ssh-2fa

```
dpkg-query -W -f='${binary:Package}\t${Status}\t${db:Status-Status}\n' auditd
audispd-plugins
```

**Output:**

```
dpkg-query: no packages found matching auditd
dpkg-query: no packages found matching audispd-plugins
```

**Issue 2: Missing External OS-level Logging**

Currently, system logs are stored on the local disk. In the event of a compromise, an attacker with elevated privileges is likely to delete all logs from disk and disable logging to prevent effective forensics, thus logs stored locally cannot be trusted. Therefore, it is a good practice to designate dedicated log servers to collect logs from the host in an isolated network.

**Affected File:**

```
/etc/rsyslog.conf
```

**Command:**

```
grep -E '^\s*([^#]+\s+)?action\(([^#]+\s+)?\btarget=\"?[^#"]+\"?\b' /etc/rsyslog.conf
/etc/rsyslog.d/*.conf
```

**Output:**

```
(empty)
```

It is recommended to install and configure the *auditd* package, as well as enable remote logging[88] to a secure logging server. Please note log events should be transferred in real time, so attackers have no time to modify them before they are transferred.

### COV-01-027 WP5: Lack of Commit Signatures in Git Repository *(Low)*

**Retest Notes:** The CoverDrop team resolved this issue and 7ASecurity confirmed that the fix is valid.

The CoverDrop Git repository was found to have multiple unsigned commits. If an attacker compromises a machine belonging to a developer, or gains access to the SSH key used by a developer, it might be possible to commit malicious code to the repository on behalf of that individual. Therefore, signature verification should be part of the code review workflow.

**Affected Resources:**
*guardian/coverdrop git repository*

[88] https://docs.fluentbit.io/manual/pipeline/inputs/syslog

### Example 1: Signed Git Commit

To show a valid commit signature the following command can be used:

#### Command:

```
# List signature for a commit
git show 4c62eb6f982b522a9136f87f327ee51131c7438e --show-signature
```

#### Output:

```
# Signed commit
commit 4c62eb6f982b522a9136f87f327ee51131c7438e (HEAD -> main, origin/main,
origin/HEAD)
gpg: Signature made Mon 05 Feb 2024 11:15:20 AM EST
gpg:                using RSA key B5690EEEBB952194
gpg: Can't check signature: No public key
Author: Mario Savarese <57295823+marsavar@users.noreply.github.com>
Date:   Mon Feb 5 16:15:20 2024 +0000
```

### Example 2: Unsigned Git Commit

In case of no signature nothing is returned in the output. For example:

#### Command:

```
# List signature for a commit
git show 4e93fc0f2e3f2c43307fe095434fca811617c139 --show-signature
```

#### Output:

```
# Missing signature part
commit 4e93fc0f2e3f2c43307fe095434fca811617c139
Author: D[...] H[...] <d[...].h[...]@gmail.com>
Date:   Mon Jul 31 16:51:08 2023 +0100
```

It is recommended to sign[89] all commits to easily verify the author of the changes and reject unsigned changes.

[89] https://docs.github.com/en/authentication/managing-commit-signature-verification/signing-commits

### COV-01-029 WP1/3: Weaknesses in Journalist Signal Chat *(Low)*

**Retest notes:** The CoverDrop team resolved this issue and 7ASecurity confirmed that the fix is valid.

The CoverDrop system relays messages posted by users to dynamically created *Signal*[90] group chats for communication with journalists. The *Signal* account is associated with *signal-bridge* and used via *signal-cli*[91], which has administrative privileges in chat groups, while journalists are invited as regular members. The absence of a human administrator hampers some management capabilities. For example, if an unwanted party joins a group chat, nobody is able to remove them. Additionally, all users can invite anyone to the chat without approval, and can modify various chat parameters, such as the *disappearing messages* setting.

**Affected Resources:**
*git: coverdrop/signal-bridge*

**Possible Attack 1: Unwanted participant eavesdropping conversation**

An invited user becomes an unwanted participant who cannot be removed from the group, as only the signal-bridge has administrator privileges, hence the CoverDrop IT administrator must remove them manually. In the meantime, the unwanted participant can observe the chat and capture all messages sent by the anonymous CoverDrop user.

**Possible Attack 2: Set disappearing messages to send invisible messages**

An unwanted user in the group chat sends an invisible message to the anonymous user, so disappearing messages are set to the lowest value (*5s*). After writing the message, it is immediately captured by the backend, and passed to the queue, which forwards it to the CoverDrop user. However, the message in the Signal group chat disappears, and nobody is aware something was sent to the anonymous user. It is worth remembering that all messages, including deleted messages, are also forwarded back to the CoverDrop user. Thus, users should not rely on the delete functionality in Signal.

It is recommended to grant administrative access to the first journalist who is invited to a group chat, so that initially there are two administrators (signal-bridge and the journalist). Additionally, it is worth enabling group links with approvals to add new users to the chat. This will reduce the likelihood of unwanted users joining the chat, and reading messages delivered through CoverDrop.

[90] https://signal.org/
[91] https://github.com/AsamK/signal-cli

### COV-01-030 WP1/3: Possible Impersonation via missing Signal Data *(Medium)*

**Retest notes:** The CoverDrop team resolved this issue and 7ASecurity confirmed that the fix is valid.

The CoverDrop application is integrated with Signal group chats via the *signal-bridge* and *signal-cli* components deployed in the on-premise infrastructure. According to the design outlined in the whitepaper, journalists can invite more users to collaborate on a case, with two types of users specified: individual journalists and desk teams. These desk teams comprise a group of journalists working on a single case and sharing a single Signal account.

In scenarios where a journalist invites third parties to the group, the anonymous CoverDrop user has no way to know who they are communicating with, hence this setup creates the illusion for the user that they are communicating with a single journalist. The reason for this is that any message posted in the chat group is sent to the anonymous user, without control messages -such as information about who joined the chat-, nor details about the message author.

Therefore, such a design might be exploited if a malicious party manages to join the group chat, i.e. via a SIM swapping[92] attack against any group participant. Despite the protections in place for the backend Signal account against account takeovers, invited users remain susceptible to targeting. Hence, an attacker with group access could impersonate the journalist and attempt to fool anonymous CoverDrop users to reveal their identity or other sensitive information.

**Affected Resource:**
*CoverDrop mobile applications*

This issue can be confirmed by having two users join the Signal group, making both send a message to the anonymous user, and observing how the CoverDrop anonymous user has no way to tell who is the message author:

[92] https://www.sec.gov/secgov-x-account

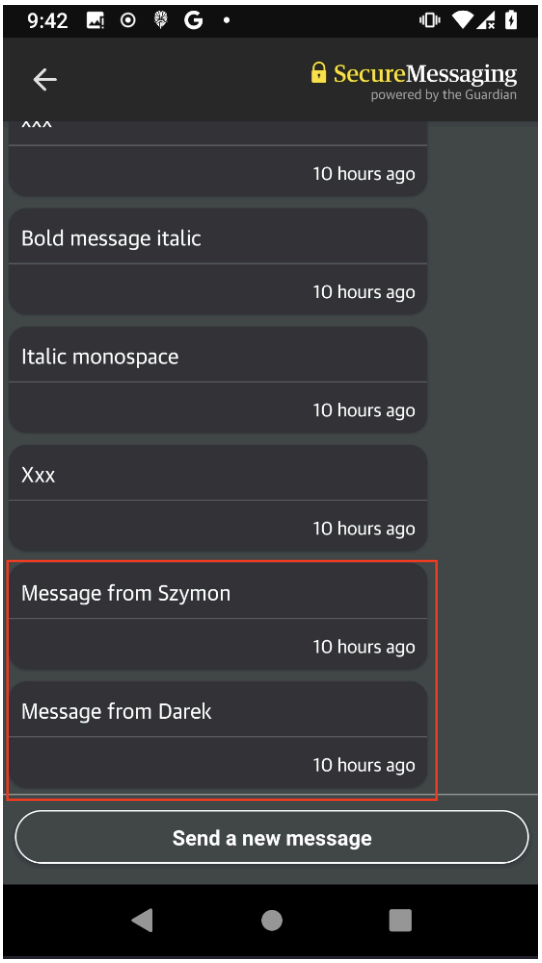



*Fig.: CoverDrop does not provide information about the message author*

The issue concerns the design of the application thus there is no clear single solution. It is recommended to review the requirements and perhaps poll possible application users to determine a desirable outcome. The following possible mitigations are provided for consideration:

- Author information could be included along with the message.
- Control messages about the Signal group chat participants could be forwarded for transparency.
- Messages from invited users, beside the first journalist account invited during the chat group creation, could be dropped, as all messages delivered to the anonymous user should originate from a single person.
- Group chat commands such as */block* and */unblock* may be implemented, to authorize the invited Signal participants to communicate directly with the user, along with some information when additional participants are added to the group, so it is possible to distinguish this in the CoverDrop application.
- The *signal-bridge* could monitor alerts when chat participants change devices, and require communication approval from the group chat admin, before

potentially compromised participants are allowed to communicate with the end-user. Ideally, in this scenario the backend ought to issue a suspicious action alert, and remove such users from the chat, requiring a new invitation to be issued.

- All parties meant to use Signal communications should be educated, to ensure they configure their Signal accounts[93] in a secure manner, are aware about possible phishing and SIM swapping attacks, and how to mitigate such risks.

### COV-01-031 WP5: Multiple Weaknesses in Kubernetes Cluster Config *(Medium)*

**Retest Notes:** The CoverDrop team addressed the principal concerns for this issue and 7ASecurity confirmed that those fixes were valid.

The Kubernetes cluster uses the K3S distribution, is installed on a single Ubuntu-based node, and is configured with default security settings. During the assignment, it was discovered that the K8s configuration is currently missing some hardening options. This may be verified by comparing it against the *CIS Benchmark* for Kubernetes[94]. Among the most important missing features, it is worth mentioning the absence of the *Pod Security Standard (PSS)*[95], which enforces runtime compliance, the lack of adequate cluster-wide auditing settings, and the absence of *Kernel Protection*[96]. Please note that the cluster has securely defined network policies for the *on-premise* namespace, significantly reducing potential attacks in the event of a compromised Pod.

**Affected Resources:**
*On-Premise Kubernetes Cluster*

**Issue 1: No PSS and audit settings**

There is no custom configuration file, nor arguments passed to the *k3s* process, related to the admission controller (PSS) or auditing, which results in suboptimal default settings[97] being applied to Kubernetes-related services.

**Command (k3s process arguments):**

```
ps auxww | grep "k3s server"
```

**Output:**

```
root       18141  7.2  2.1 5626656 709340 ?      Ssl   2023 5170:28 /usr/local/bin/k3s
server
```

[93] https://cert.europa.eu/publications/security-guidance/security-guidance-22-002---hardening-signal/
[94] https://www.cisecurity.org/benchmark/kubernetes
[95] https://kubernetes.io/docs/concepts/security/pod-security-standards/
[96] https://docs.datadoghq.com/security/default_rules/cis-kubernetes-1.5.1-4.2.6/
[97] https://docs.k3s.io/security/hardening-guide#control-plane-execution-and-arguments

**Command (configuration file):**

```
stat /etc/rancher/k3s/config.yaml
```

**Output:**

```
stat: cannot statx '/etc/rancher/k3s/config.yaml': No such file or directory
```

**Issue 2: No --protect-kernel-defaults argument passed to kubelet process**

**Commands:**

```
# CIS Benchmark verification tool with K3S profile
kube-bench | grep FAIL
```

**Output:**

```
[FAIL] 4.2.5 Ensure that the --streaming-connection-idle-timeout argument is not set to 0 (Automated)
[FAIL] 4.2.6 Ensure that the --protect-kernel-defaults argument is set to true (Automated)
[FAIL] 4.2.7 Ensure that the --make-iptables-util-chains argument is set to true (Automated)
```

It is recommended to review the current Kubernetes configuration against CIS benchmarks and apply all possible recommendations[98]. This should be done, regardless of the final Kubernetes distribution used in production, since benchmarks exist for all major variations. The cluster should undergo regular scanning, using tools such as *kube-bench*[99] and others. These tools seamlessly integrate well with the cluster, and can be scheduled as jobs within the cluster, to provide early notifications in case of potential misconfigurations. Consideration should additionally be given to scanning container images, stored in the local registry, to spot outdated artifacts. It ought to be ensured that all logs, including OS-level logs and cluster-wide auditing logs, are integrated with a centralized logging system. Furthermore, it is strongly advised to launch a multi-node deployment in production.

[98] https://docs.k3s.io/security/hardening-guide
[99] https://github.com/aquasecurity/kube-bench

### COV-01-032 WP5: Multiple Weaknesses in Pod Configurations *(Medium)*

**Retest Notes:** The CoverDrop team addressed the principal concerns for this issue and 7ASecurity confirmed that those fixes were valid.

The Kubernetes infrastructure audit revealed that multiple Pods lack a number of essential hardening features. Specifically, multiple containers were found to be running as *root*, without *runAsNonRoot* set to *true,* or without *allowPrivilegeEscalation* set to *false*. Additionally, almost no Pods drop *Linux* capabilities, or restrict capabilities to the bare essential ones. These weaknesses fail to implement the *Least Privilege*[100] security principle, and may provide attackers with access to resources in edge-case scenarios. Please note that, for the sake of brevity, only a few examples are listed below. The complete results should be reviewed using automated tools, such as for instance *checkov*[101].

**Affected Resources:**
*On-Premise Kubernetes Cluster*

**Example 1: Multiple weaknesses in container configurations**

The current failure to adhere to container hardening best practices can be confirmed using the following command, after fetching and saving deployment files in a directory:

**Commands:**

```
checkov --skip-download -d . | grep "FAILED" -B1 | grep "Check"
```

**Output:**

```
# Unique findings from all deployments from all namespaces
Check: CKV2_K8S_6: "Minimize the admission of pods which lack an associated
NetworkPolicy"
Check: CKV_K8S_11: "CPU limits should be set"
Check: CKV_K8S_13: "Memory limits should be set"
Check: CKV_K8S_15: "Image Pull Policy should be Always"
Check: CKV_K8S_20: "Containers should not run with allowPrivilegeEscalation"
Check: CKV_K8S_22: "Use read-only filesystem for containers where possible"
Check: CKV_K8S_23: "Minimize the admission of root containers"
Check: CKV_K8S_25: "Minimize the admission of containers with added capability"
Check: CKV_K8S_28: "Minimize the admission of containers with the NET_RAW capability"
Check: CKV_K8S_30: "Apply security context to your containers"
```

[100] https://www.cisa.gov/uscert/bsi/articles/knowledge/principles/least-privilege
[101] https://bridgecrew.io/blog/kubernetes-static-code-analysis-with-checkov/

```
Check: CKV_K8S_31: "Ensure that the seccomp profile is set to docker/default or
runtime/default"
Check: CKV_K8S_35: "Prefer using secrets as files over secrets as environment
variables"
Check: CKV_K8S_37: "Minimize the admission of containers with capabilities assigned"
Check: CKV_K8S_38: "Ensure that Service Account Tokens are only mounted where
necessary"
Check: CKV_K8S_40: "Containers should run as a high UID to avoid host conflict"
Check: CKV_K8S_43: "Image should use digest"
Check: CKV_K8S_8: "Liveness Probe Should be Configured"
Check: CKV_K8S_9: "Readiness Probe Should be Configured"
```

**Example 2: Pod running as root**

Only the *signal-cli* pod, in the *on-premises* namespace, was found to be running as *root*. The following commands execute the *id* command, on a container to confirm it is running as *root*.

**Command (*on-premises* namespace):**

```
# Executed inside a signal-cli-deployment-f5c88d855-rrgqx pod in on-premises namespace
id
```

**Output:**

```
uid=0(root) gid=0(root) groups=0(root)
```

**Example 3: Inconsistent settings on non on-premise deployments**

Options like with *digest*[102], or *deployed cosign*[103] verification, are applied to custom workloads launched in the *on-premise* namespace. However, they are missing in *kube-system* deployments:

**Command (*sealed-secrets-controller* image with tag instead of digest):**

```
kubectl get deployment sealed-secrets-controller --namespace kube-system
```

**Output:**

```
image: docker.io/bitnami/sealed-secrets-controller:v0.24.5
```

Sample comparison to a correct deployment with a digest defined for the image:

**Command (*signal-cli-deployment* image with digest):**

[102] https://docs.bridgecrew.io/docs/bc_k8s_39
[103] https://github.com/sigstore/cosign

```
kubectl get deployment signal-cli-deployment --namespace on-premises
```

**Output:**

```
image:ghcr.io/guardian/coverdrop_signal-cli@sha256:b07c5e96f3d10ada208641f2e6e0ee85985f
29d071922ab58b8f0eaf809f248a
```

It is recommended to employ static code analysis tools into the development pipelines, to scan and detect misconfigurations in Kubernetes Deployments. In this particular case, it is strongly advised to ensure no containers are running as *root*, and a minimal set of capabilities[104] is defined for each container, or all capabilities are dropped. Deployments should then utilize security context features[105], to reduce the permissions to the minimum necessary for the solution to work. Additionally, secrets should ideally be mounted as files instead of environment variables for better access control protection.

### COV-01-033 WP5: Unrestricted on-premise Outbound Traffic *(Medium)*

**Note:** The CoverDrop team plans to address this issue in the next release.

The Kubernetes server was found to have unrestricted outbound access to the Internet. In the event of a container compromise, a malicious attacker might leverage such weakness to fetch additional tools, escalate privileges, or easily exfiltrate data.

**Affected Resources:**
*On-Premise Kubernetes Cluster*

Unrestricted outbound connectivity can be confirmed running the following command:

**Command:**

```
curl ifconfig.co
```

**Output:**

```
HTTP/1.1 200 OK
Date: Wed, 07 Feb 2024 03:44:22 GMT
[...]
77.91.250.148
```

It is highly recommended to restrict outbound traffic. This should only be allowed in the minimum possible number of services, which truly require outbound traffic to be operational. All other traffic should be disallowed, or only temporarily enabled when administrators need it to perform maintenance procedures.

[104] https://docs.fugue.co/FG_R00493.html
[105] https://kubernetes.io/docs/tasks/configure-pod-container/security-context/

# WP6: CoverDrop Lightweight Threat Model

## Introduction

***Client note:*** *Some of the recommendations summarized below were addressed during the test period. For details please see the notes accompanying the issue descriptions in the "Identified Vulnerabilities" and "Hardening Recommendations" section of this report.*

The *CoverDrop* project aims to be a universal solution for whistleblowers that intend to contact journalists to share sensitive information, while they are potentially targeted by high profile attackers, such as government-sponsored adversaries. The project implements a communication protocol with multiple security countermeasures preventing powerful threat actors from disclosing the identity of application users, as well as features providing plausible deniability, in case the whistleblower or parts of the backend infrastructure are confiscated. As a result, CoverDrop is designed as a module which can be integrated with legitimate applications, and potentially adopted by multiple news organizations around the world, to protect the anonymity of individuals in the world of omnipresent surveillance.

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. Lightweight threat modeling refers to a simplified threat modeling process, loosely following the STRIDE[106] methodology, which does not involve workshops, but instead focuses on the analysis of the system, as performed by 7ASecurity, based on the documentation, specification and source code, with the assistance of a representative of the client.

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 mitigations. As the main target is a module, which may be integrated into any application to enable a covert communication channel, many threats from an external attacker perspective, common for web and mobile applications, have appropriate countermeasures designed and implemented, thus the threat model focuses on the general overview of the system, supply chain attacks, and deployment of the environment in which the application is going to operate in the near future. Taking into account such assumptions the main threats identified to be relevant to the environment can be categorized in the following groups**:**

1. Attacks against the supply chain (i.e. deployment and development).
2. Denial of service conditions rendering the system temporarily or permanently inoperational.

[106] https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats#stride-model

3. Incident response readiness in terms of logging, monitoring and anomaly detection.
4. User deanonymization attacks.
5. Sensitive data handling.

## Relevant assets and threat actors

The following assets are considered important for the CoverDrop project:

- CoverDrop Source Code (A01)
- CoverDrop Build Artifacts (A02)
- Backend Signal Account (A03)
- Journalist Vaults (A04)
- Conversation Chat History (A05)
- CoverNode Private Key (A06)
- Journalist Private Key (A07)
- AWS Secrets (A08)
- Authenticated Github Session (A09)
- Key Hierarchy (A10)
- Encrypted Messages (A11)
- Provisioning Keys (A12)

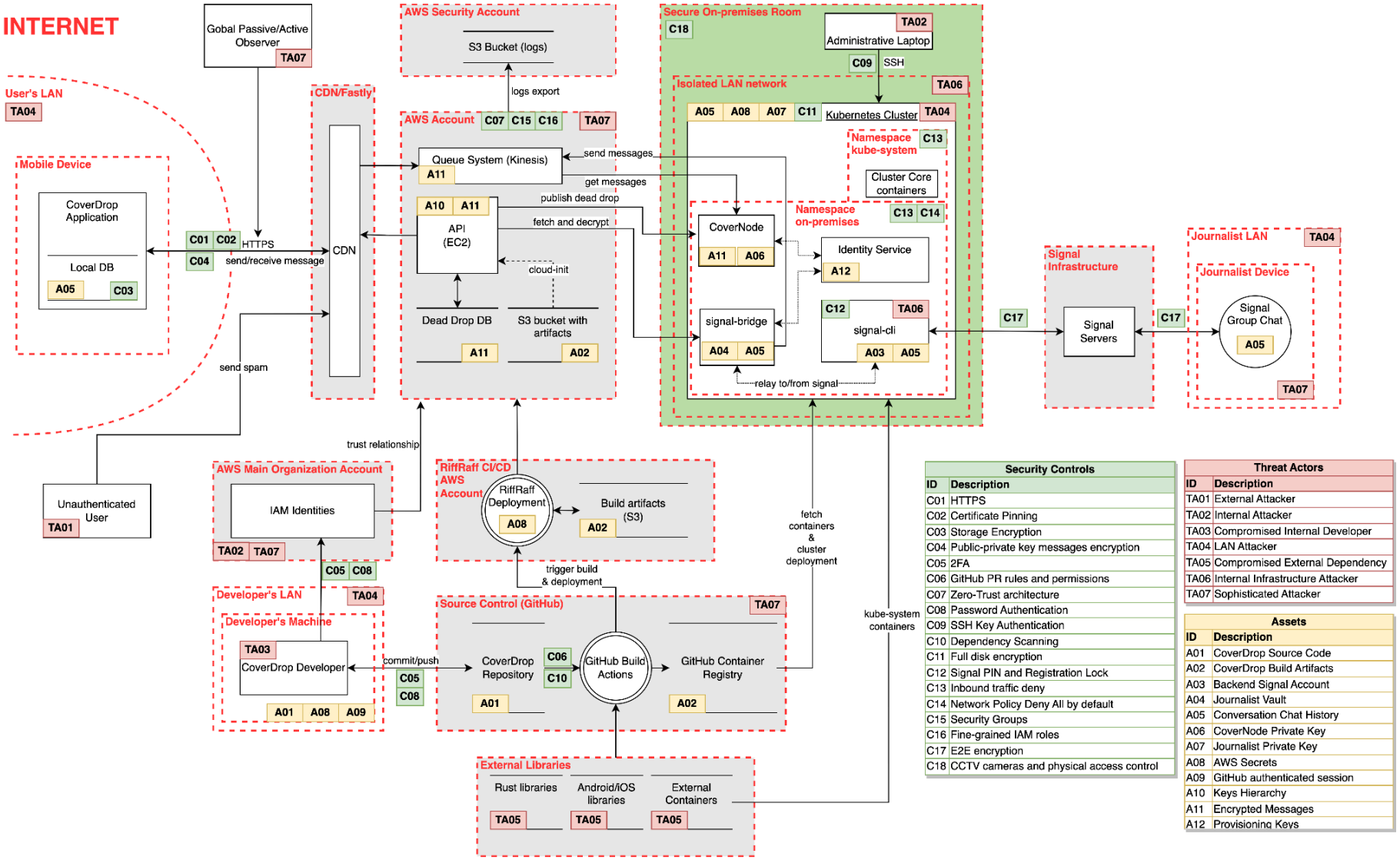
The following threat actors are considered relevant to the CoverDrop project:

- External Attacker (TA1)
- Internal Attacker (TA2)
- Compromised Internal Developer (TA3)
- Network/LAN Attacker (TA4)
- Compromised External Dependency (TA5)
- Internal Infrastructure Attacker (TA6)
- Sophisticated Attacker (TA7)

## Attack surface

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.

The following diagram provides an overview of potential attacks against the currently implemented deployment and development process as envisioned by 7ASecurity:




| Security Controls | |
|---|---|
| ID | Description |
| C01 | HTTPS |
| C02 | Certificate Pinning |
| C03 | Storage Encryption |
| C04 | Public-private key messages encryption |
| C05 | 2FA |
| C06 | GitHub PR rules and permissions |
| C07 | Zero-Trust architecture |
| C08 | Password Authentication |
| C09 | SSH Key Authentication |
| C10 | Dependency Scanning |
| C11 | Full disk encryption |
| C12 | Signal PIN and Registration Lock |
| C13 | Inbound traffic deny |
| C14 | Network Policy Deny All by default |
| C15 | Security Groups |
| C16 | Fine-grained IAM roles |
| C17 | E2E encryption |
| C18 | CCTV cameras and physical access control |

| Threat Actors | |
|---|---|
| ID | Description |
| TA01 | External Attacker |
| TA02 | Internal Attacker |
| TA03 | Compromised Internal Developer |
| TA04 | LAN Attacker |
| TA05 | Compromised External Dependency |
| TA06 | Internal Infrastructure Attacker |
| TA07 | Sophisticated Attacker |

| Assets | |
|---|---|
| ID | Description |
| A01 | CoverDrop Source Code |
| A02 | CoverDrop Build Artifacts |
| A03 | Backend Signal Account |
| A04 | Journalist Vault |
| A05 | Conversation Chat History |
| A06 | CoverNode Private Key |
| A07 | Journalist Private Key |
| A08 | AWS Secrets |
| A09 | GitHub authenticated session |
| A10 | Keys Hierarchy |
| A11 | Encrypted Messages |
| A12 | Provisioning Keys |

*Fig.: Data flow diagram for the CoverDrop environment, application, cloud, and infrastructure*

7ASecurity identified the following threat categories according to STRIDE, as relevant to the CoverDrop project, deployed infrastructure and the related DevOps processes:

**Threat 01: Identity or Data Spoofing**

**Overview:**
A spoofing attack allows an adversary to impersonate either a person or a program to gain advantage in the system. Spoofing may lead to various consequences such as data disclosure, if the receiver accepts the forged identity of the attacker, or to execute undesired actions on behalf of some entity, which the attacker pretends to be.

**Countermeasures:**
The whitepaper[107], as well as the CoverDrop team, correctly identified this threat as a serious issue. Hence the following mechanisms are already implemented to prevent spoofing attacks:

- Certificate pinning in mobile applications and transport-layer encryption
- Additional message encryption and signatures within the encrypted transport layer.
- Manual verification of journalists and monitoring of trusted Signal devices.
- Zero-trust cloud architecture.
- Container signatures.
- Public key hierarchy structure.

**Attack Scenarios:**
Despite the implementation of multiple countermeasures, room for improvement was identified in the following attack scenarios:

- Take-over of the *signal-bridge* Signal account, using a cloned account or linked devices.
- Take-over of journalist Signal accounts, i.e. via SIM swapping attacks.
- Impersonation of message authors to end users, due to the absence of authoring metadata, when messages from other participants in the Signal group chat are passed to the system.
- Impersonation of legitimate developers, due to the lack of enforced git signatures.

**Recommendation:**
It is advised to strengthen identity verification, and enhance monitoring, by implementing domain-relevant anomaly detection, similar to the IDS or IPS systems that have been utilized in IT security for years. Depending on the constraints, it may be necessary to develop and test processes for each attack scenario, and ensure that the team is trained

[107] https://petsymposium.org/2022/files/papers/issue2/popets-2022-0035.pdf

to respond effectively when an attack is detected. The following technical solutions could be explored to enhance defenses against the aforementioned attacks:

- Enforcement of signatures in the CI/CD process, adherence to the principle of least privilege, and ensuring a robust code review process is implemented.
- All accounts, particularly Signal accounts, ought to be adequately hardened at the account level (Signal properties) and used on secure and hardened devices (mobile/laptops). This also applies to the backend *signal-bridge* account, as well as all other Signal accounts involved in sensitive data processing. Consideration should be given to implementing an allow-list of accounts that can be invited to collaborate with journalists, or establish a verification process that participants must undergo to join. Invitations through a dedicated */invite* command could be a possible solution for more granular access control.
- Implementing anomaly detection, and procedures to automatically remove compromised assets (e.g. a journalist) from sensitive Signal groups whenever possible.
- For critical assets such as the *signal-bridge* account, appropriate anomaly monitoring and deactivation processes should be in place. This might include issuing a new Signal account number and reactivating all Signal groups.
- Integrity checks should be implemented and monitored for any modification to the public key hierarchy.
- All integrated systems and applications rendering user-supplied messages, such as the upcoming journalist app, should sanitize messages and utilize hardened views to reduce the potential for client-side attacks, such as phishing, XSS and premium number calls.

**Threat 02: Data Tampering or Unauthorized Modifications**

**Overview:**
Data tampering occurs when an attacker modifies information without the relevant authorization. This attack encompasses a broad range of assets, involving data modification on disk, in-memory, and in-transit. A successful attack may lead to multiple consequences, such as the modification of data exchanged between two parties, alteration of code, unauthorized access to the system, or compromise of the entire infrastructure.

**Countermeasures:**
This threat was found to be already taken into consideration by the CoverDrop team while designing the system. Hence, the following countermeasures are in place to mitigate this risk:

- Communications are protected by an encrypted transport layer between mobile devices and the API, including certificate pinning.
- Integrity checks are in place for messages between users and *CoverNodes*, as well as between *CoverNodes* and journalists.

- Utilization of end-to-end Signal communicator between the backend and journalists.
- Rigorous physical access control to on-premise infrastructure.
- Strict control over launched containers (using hashes) and signed build artifacts.
- Enforced two-factor authentication (2FA) for GitHub accounts.
- Mandatory code reviews and GitHub Pull Request rules with protected branches.
- Employment of fine-grained GitHub access tokens.
- Limited dependencies with pinned versions to minimize supply chain attacks.

Despite the implemented countermeasures, the following attack scenarios, primarily targeting the on-premise infrastructure, are deemed relevant for the environment and would benefit from remediation strategies.

**Attack Scenarios:**

- Unauthorized access and modifications to sensitive *CoverDrop* data in on-premise resources may occur due to e.g. lax server file system permissions and misconfigurations in Kubernetes cluster deployments, which are susceptible to container escapes.
- Potential kernel exploitation due to insufficient protection of Kubernetes clusters and inadequate logging and monitoring of OS-level anomalies.
- Attacks originating from CI/CD pipelines (e.g. RiffRaff) or other AWS accounts within the organization due to internal DevOps practices and shared artifact storage.
- Attacks on GitHub PR rules may occur due to the monorepo design, wherein compromised developers responsible for minor modules can introduce vulnerabilities into the infrastructure code or poison build artifacts.

**Recommendation:**

The described attack scenarios can be mitigated by adhering to the principle of least privilege on all components, both within the internal cluster and in the design of internal processes. As a baseline, CIS Benchmarks should be implemented for all assets in production environments. Shared CI/CD environments ought to undergo periodic audits, be treated as critical assets, and be included in holistic security reviews or internal red teaming engagements to test all implemented security controls. Alternatively, the environment hosting CoverDrop infrastructure may be isolated from other organizational components. Additionally, infrastructure-related source code in the Infrastructure as Code (IaaC) approach must always be handled with caution and ideally separated from less critical modules.

**Threat 03: Repudiation Attacks**

**Overview:**
Preventing attacks is equally important as logging and monitoring, which provide insight into all actions performed in the system, addressing the concept of repudiation threats. Comprehensive and robust logging and monitoring, though often underappreciated, serve as essential tools for forensic investigators to identify initial attack vectors. In modern security systems, they form the foundation of effective anomaly detection. If attackers can successfully cover their tracks or if there are insufficient controls to monitor activities, the company cannot be certain that the attack will not reoccur, what was the goal of the attacker and whether the attack was successful.

**Countermeasures:**
- Application-level logs collected by Fluentbit from containers in the on-premise Kubernetes cluster.
- AWS logs are shipped to a separate dedicated AWS account within the organization.
- Physical access security measures include physical access control, CCTV cameras, and a dedicated laptop for managing the on-premise cluster.

**Attack Scenarios:**
- Deletion of on-premise server logs due to the absence of OS-level log forwarding.
- Challenges in establishing accountability on on-premise servers due to shared OS-level accounts.
- Challenges in establishing accountability in the Kubernetes cluster due to inadequate cluster-level audit logging and the usage of a local cluster-level admin.

**Recommendation:**
The following technical solutions should be investigated to enhance defenses against the listed attacks:
- The infrastructure should improve logging and monitoring capabilities, both in the cloud and on-premise. Currently, numerous tools are available to implement centralized logging, monitoring, anomaly detection, and alerting effectively.
- At a minimum, adequate OS-level monitoring (including *auditd* rules) and audit logs for the Kubernetes cluster should be configured.
- All other services should be configured to collect as many logs as possible, detailing who performed actions and when. This approach leaves no room for attackers to cover their tracks in the event of a compromise.
- Systems should avoid the use of shared accounts. Instead, individual accounts for all staff members should be created in both the operating system and the

on-premise cluster. These identities should be centrally managed, facilitating credential invalidation and rotation.

**Threat 04: Information disclosure**

**Overview:**
Information disclosure occurs when an attacker gains access to sensitive data. This may occur due to inadequate security controls implemented in the system, incorrect encryption applied to data in transit or stored locally, or flaws in the design that can be exploited to extract sensitive information.

**Countermeasures:**
Information protection is one of the most critical components of the entire solution, and multiple security controls are detailed in the whitepaper as well as in the actual implementation. The following mechanisms have been implemented to enhance the resilience of the system to information disclosure threats:

- Encrypted transport layer between mobile and backend services,
- Encryption of messages using public-private key encryption.
- On-premise encrypted secrets loaded from repositories.
- Dedicated and isolated on-premise infrastructure hosting critical services, along with a zero-trust architecture for cloud deployment.
- Message mixing strategies that blend encrypted and dummy messages to protect communication parties and ensure anonymity.
- A list of allowed first points of contact for journalists.
- Encryption of end-user data storage to prevent disclosure of communication content.
- Plausible deniability features in mobile applications, such as static size of encrypted storage and messages, the absence of CoverDrop specific logs, and covert messages.
- Full disk encryption applied to on-premise servers.

Despite the multiple countermeasures in place, the following possible attack scenarios require additional remediation measures:

**Attack Scenarios:**

- Unauthorized access to Signal groups due to the absence of strict invitation and moderation rules.
- N-1 attacks[108] against dead drop messages under certain edge conditions.
- Extraction of sensitive data from memory on mobile devices.

[108] https://www.freehaven.net/anonbib/cache/danezis:wpes2003.pdf

- Phishing attacks resulting from insufficient implementation of countermeasures on mobile devices and the absence of integration guidelines for other adopters of the solution.
- Leakage of sensitive data via git history due to the absence of pre-commit hooks.
- Exfiltration of data from on-premise infrastructure due to the lack of appropriate egress filtering rules.
- Stealthy takeover of Signal accounts (*signal-bridge*/journalist) due to the absence of anomaly detection and integration.

**Recommendation:**
It is advised to consider the following technical solutions to improve defenses against the aforementioned attacks:

- The environment may tighten rules for Signal groups and implement anomaly detection while using the third-party protocol in the system. Ideally, a custom and more controllable communication solution should be implemented.
- Both the applications and the backend could improve control over sensitive data and address common attacks against mobile device users by employing memory protection mechanisms.
- When the software is publicly released, it is crucial to provide appropriate guidelines derived from multiple security reviews for other adopters of the system. This ensures that applications embedding *CoverDrop* do not break the security properties of the system due to trivial development bugs.
- More tests focused on anonymity and plausible deniability should be conducted to ensure the highest possible security.
- Backend services should implement anomaly detection for indicators of N-1 attack exploitation. Additionally, to remove fixed parameters and predictable outcomes, it is recommended to investigate the dynamic size of published dead drop batches, random order of published messages, and enforced covert messages included in each batch, instead of filling the batch till the fixed size of 500 messages.

**Threat 05: Temporary or Permanent Denial of Service**

**Overview:**
Denial of service occurs when the system is unable to perform its intended actions, either temporarily or permanently. Ensuring availability is crucial for software responsible for exchanging sensitive messages. Users, who may be targeted by sophisticated threat actors, as well as journalists awaiting valuable information, may experience disruptions in communication in extreme cases, potentially leading to data loss. Both parties must be assured that the system is as resilient as possible against both common and large-scale attacks.

**Countermeasures:**

- Utilizing a CDN in conjunction with an auto-scalable cloud infrastructure to manage messaging through Fastly and the Kinesis queue.
- Block commands in Signal groups preventing potential abusive or spam messages

**Attack Scenarios:**

- Flood attacks due to the absence of smart rate-limiting functionality for mobile clients sending messages to the Kinesis queue. This may particularly affect the on-premise infrastructure.
- Temporary Signal account outages bound to *signal-bridge* may occur due to the lack of spam protection functionality, while Signal servers implement rate limiting in response to users sending multiple small messages within a short timeframe, which may result in throttling from Signal infrastructure.
- Disk space outages in on-premise components may arise due to the lack of OS-level hardening and resource limitations, as well as the configuration of the containers and libraries in use (attachment and stores in *signal-cli*).
- A single Signal account, used by *signal-bridge,* represents a single point of failure in the event of a successful SIM swapping attack, for example.
- Denial of service due to the usage of personal GitHub tokens to access the container registry.
- System overload due to the lack of a high availability configuration for on-premise components.
- Denial of service conditions due to insufficient protection against DNS hijacking on mobile devices.

**Recommendation:**

The following technical solutions ought to be evaluated to improve the security posture against the listed attacks:

- Backend services should implement smart rate-limiting features to prevent various flood attacks, particularly in the initial stages of payload processing.
- Employing a high-availability cluster configuration to enhance the resilience of on-premise infrastructure.
- Whenever possible, the backend should use multiple Signal accounts.
- Personal tokens should be avoided in all parts of the system, configuring dedicated accounts instead.
- DNS hijacking attacks should be included in the threat model of the mobile applications and secured using DNS over HTTPS or DNS over TLS.

**Threat 06: Privilege Escalation**

**Overview:**

Privilege escalation describes a situation in which an attacker gains sufficient access to compromise the most critical assets in the environment. Threats falling under this

category are typically the most devastating, indicating that the attacker has breached most of the defensive mechanisms. Among privilege escalation attacks, the most typical are remote code execution, local privilege escalation, or supply chain attacks.

**Countermeasures:**
The CoverDrop team regards the cloud infrastructure as a breachable asset, while the on-premise infrastructure handles the most sensitive data. Therefore, the following countermeasures were confirmed to be implemented:

- Zero-trust architecture for the cloud infrastructure.
- Custom containers running in a Kubernetes cluster using low-privileged users.
- Dependency pinning.
- Container signatures.

**Attack Scenario:**

- Access to the on-premise infrastructure via supply chain attacks, i.e. targeting the built-in Kubernetes containers.
- Access to the on-premise cluster via poisoning GHCR containers.
- Remote code execution and/or container escape due to insufficient hardening of on-premise servers and containers.
- Attacks originating from CI/CD pipelines (RiffRaff) or other AWS accounts within the organization.
- Spear phishing attacks against journalists leveraging the desire to exchange sensitive information as the primary purpose of the *CoverDrop* application.

**Recommendation:**
Despite the difficulty to perform privilege escalation attacks, it is advised to consider the following technical solutions to improve defenses against the aforementioned scenarios:

- Message inspection for common attack vectors, using well-known or multiple security solutions, including inspection of decrypted payloads before passing data to external components (such as *signal-cli*).
- Enhancement of container deployment settings to ensure all containers are thoroughly vetted.
- Periodic auditing of AWS organization accounts with an emphasis on cross-account privilege escalation.
- Isolation of AWS account and CI/CD process from internal organization toolsets and shared environments to limit privilege escalation attack vectors.

# WP7: CoverDrop Supply Chain Implementation

## Introduction and General Analysis

The *8th Annual State of the Software Supply Chain Report*, released in October 2022[109], revealed a 742% average yearly increase in software supply chain attacks since 2019. Some notable compromise examples include *Okta*[110], *Github*[111], *Magento*[112], *SolarWinds*[113], and *Codecov*[114], among many others. To mitigate this concerning trend, Google released an End-to-End Framework for *Supply Chain Integrity* in June 2021[115], named *Supply-Chain Levels for Software Artifacts* (*SLSA*)[116].

This area of the report elaborates on the current state of the supply chain integrity implementation of the CoverDrop 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.

The following sections elaborate on the results against versions 0.1 and 1.0 of the SLSA standard. At the time of this assignment, CoverDrop components are hosted on private GitHub repositories, however, the intention is to release these as open source projects.

The CoverDrop project will be distributed using GitHub, it uses automated tools to synchronize the source code of the packages to their own GitHub repositories. Similarly, distribution is also performed through Maven artifacts, attached to GitHub Actions. These processes align closely with the principles of SLSA, notably enhancing *Provenance*. Essentially, this signifies that not only is the build process documented, but also, the resulting artifacts are intricately linked to a known and controlled build environment.

While auditing the supply chain implementation, the CoverDrop project provided a number of positive impressions that must be acknowledged here:

- The size and complexity of custom Docker images is minimal.
- The build process for Docker images is well implemented.
- Lock files are present in cargo files.
- Github workflows are in place for mobile and Rust backend components.
- *CDK* and *RiffRaff* are leveraged for automated *Infrastructure-as-Code.*

[109] https://www.sonatype.com/press-releases/2022-software-supply-chain-report
[110] https://www.okta.com/blog/2022/03/updated-okta-statement-on-lapsus/
[111] https://github.blog/2022-04-15-security-alert-stolen-oauth-user-tokens/
[112] https://sansec.io/research/rekoobe-fishpig-magento
[113] https://www.techtarget.com/searchsecurity/ehandbook/SolarWinds-supply-chain-attack...
[114] https://blog.gitguardian.com/codecov-supply-chain-breach/
[115] https://security.googleblog.com/2021/06/introducing-slsa-end-to-end-framework.html
[116] https://slsa.dev/spec/

- A single AWS account is used, however, privileges are fine grained via multiple separated security groups.
- There is a dedicated on-premise Kubernetes cluster.
- Modern centralized logging mechanisms are implemented in some infrastructure areas.
- The workflows have automated dependency checks in place, such as *dependabot* and *cargo audit*.
- Code review processes are clearly in use.

## 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** represent the lack of SLSA[117].
- **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[118].
- **Build L2: Hosted build platform**. Builds run on a hosted platform that generates and signs the provenance[119].
- **Build L3: Hardened builds**. Builds run on a hardened build platform that offers strong tamper protection[120].

To produce artifacts with a specific SLSA level, the responsibility is split between the *Build* platform and the *Producer*. Broadly speaking, the *Build* platform must strengthen the security controls 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.

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.

In the context of the SLSA framework v1.0, CoverDrop adherence to Build Level 3 (L3) is demonstrated through its management of software artifact provenance. The

[117] https://slsa.dev/spec/v1.0/levels#build-l0
[118] https://slsa.dev/spec/v1.0/levels#build-l1
[119] https://slsa.dev/spec/v1.0/levels#build-l2
[120] https://slsa.dev/spec/v1.0/levels#build-l3

CoverDrop project is hosted on GitHub, a platform capable of producing Build Level 3 provenance. This ensures that CoverDrop fulfills the requirement to "*Choose an appropriate build platform*." Moreover, the CoverDrop artifact generation process is clearly defined. Each step is meticulously scripted, the use of GitHub Actions is leveraged to produce Maven artifacts, and for iOS, the Swift Package Manager generates Swift packages, in this case, CoverDrop meets the Producer requirements, specifically "*Follow a consistent build process*."

Furthermore, CoverDrop ensures that provenance information is not only present but also effectively distributed through the respective package manager ecosystems. By leveraging these established ecosystems, CoverDrop effectively satisfies the Producer requirements essential for achieving Build Level 3 (L3) within the SLSA framework.

| 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.

The CoverDrop project generates artifact provenance in each step; This not only identifies the output artifacts but also provides information regarding how each of them was created, this guarantees that the criterion of "*Provenance generation Exists*" is entirely met. Additionally, the format of this provenance is designed to be simply understood by customers and fits frictionlessly within the ecosystem of package management.

Nevertheless, despite diligent efforts to ensure the existence of provenance, a crucial aspect falls short when it comes to the *Authenticity* requirement. The absence of signed artifacts prevents consumers from effectively validating the legitimacy of the provenance. Additionally, the *Unforgeable* degree cannot be met either, because the provenance is not resistant to forgery. This is due to the inexistence of key material to sign artifacts.

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

In conclusion, while the CoverDrop project is SLSA v1.0 compliant, it is possible to reach level 3 (L3) easily as follows:

- Signed Artifacts: The CoverDrop project should take proactive steps to leverage available mechanisms to sign artifacts. For example, at least some signed artifacts could be distributed through Maven. Such signatures would greatly enhance the ability of consumers to validate the legitimacy of the provenance.
- After the above, automated tools like *slsa-github-generator*[121] and *slsa-verifier*[122] 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[123] 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.

[121] https://github.com/slsa-framework/slsa-github-generator
[122] https://github.com/slsa-framework/slsa-verifier
[123] https://slsa.dev/spec/v0.1/levels

**Source code control requirements:**

| Requirement | L1 | L2 | L3 | L4 |
|---|---|---|---|---|
| Version controlled | ✅ | ✅ | ✅ | ✅ |
| Verified history | | | ✅ | ✅ |
| Retained indefinitely | | | ⛔ (18 mo.) | ⛔ |
| Two-person reviewed | | | | ✅ |

**Build process requirements:**

| Requirement | L1 | L2 | L3 | L4 |
|---|---|---|---|---|
| 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 | L1 | L2 | L3 | L4 |
|---|---|---|---|---|
| Security | | | | ⛔ |
| Access | | | | ⛔ |
| Superusers | | | | ⛔ |

**Provenance requirements:**

| Requirement | L1 | L2 | L3 | L4 |
|---|---|---|---|---|
| Available | ✅ | ✅ | ✅ | ✅ |
| Authenticated | | ⛔ | ⛔ | ⛔ |
| Service generated | | ✅ | ✅ | ✅ |
| Non-falsifiable | | | ⛔ | ⛔ |
| Dependencies complete | | | | ✅ |

**Provenance content requirements:**

| Requirement | L1 | L2 | L3 | L4 |
|---|---|---|---|---|
| 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 CoverDrop is still not SLSA v0.1 L2 compliant, due to the available GitHub tools it is possible to reach level SLSA v0.1 L3 as follows:

- *GitHub Artifact retention policy*[124] ought to be implemented to comply with the retained indefinitely requirements.
- After the above, automated tools like *slsa-github-generator*[125] and *slsa-verifier*[126], could be integrated into the build process to further harden the supply chain implementation.

[124] https://docs.github.com/en/organizations/managing-organization-settings/configuring-the-retention-[...]
[125] https://github.com/slsa-framework/slsa-github-generator
[126] https://github.com/slsa-framework/slsa-verifier

# Conclusion

***Client note:*** *Many of the recommendations summarized below were addressed during the test period. For details please see the notes accompanying the issue descriptions in the "Identified Vulnerabilities" and "Hardening Recommendations" section of this report.*

The *CoverDrop* solution defended itself well against a broad range of attack vectors. In fact, not a single critical or high severity issue could be identified during this engagement. Continued cycles of security testing and hardening will further fortify the platform, making it even more resistant to potential attacks.

7ASecurity would like to highlight several positive aspects of CoverDrop, as observed by the audit team:

- Multiple checks were found to be in place in all CoverDrop components to enhance the resilience against potential exploits and unauthorized access.
- Most libraries and dependencies were found to be up-to-date, demonstrating a prioritization and maintenance of security hygiene. This approach mitigates vulnerabilities and guarantees a robust defense against potential threats..
- Overall, the CoverDrop backend components were found to be robust against many traditional web application security attack vectors. For example, no *Command Injection*, *SQL Injection (SQLi)*, *Cross-Site Request Forgery (CSRF)*, *Local File Inclusion (LFI)* or *Remote Code Execution (RCE)* issues could be identified during this exercise.
- The session and messaging implementation was resistant to manipulation, cracking attempts and replay attacks.
- The source code of the solution is well-written, easy to read, and generally adheres to a number of security best practices. In addition to this, the project is actively maintained and commits are thoroughly documented.
- Importantly, no sensitive data was found to be exposed within the code.
- The CoverDrop cloud infrastructure exhibits well-thought-out designs and configurations, prioritizing security best practices.
- The AWS infrastructure leverages modern DevOps tools, such as code automation, enabling straightforward maintenance for the team members.
- The Kubernetes cluster has a clean setup, benefiting from on-premise infrastructure. This environment enhances security, offering isolation, restricted access, and robust monitoring.
- CoverDrop demonstrates robust security measures across the iOS and Android platforms, including the avoidance of sensitive content leaks, proactive exclusion from backups, protection screens against screenshot leaks, and detection of

other possible attacks, showcasing a comprehensive approach to safeguarding user data.

- CoverDrop communications are strongly protected with TLS encryption and Certificate Pinning. Hence, traffic between the mobile applications and backend CoverDrop Nodes is protected even against high profile adversaries able to craft TLS certificates trusted by the Android and iOS operating systems (i.e. many governments, some companies). Furthermore, the Android app explicitly blocks clear-text HTTP communications, while the iOS app achieves the same result by not weakening its ATS configuration.
- The deployment of security mechanisms such as *IntegrityGuard* for Android and *SecuritySuite* for iOS classes, indicates that the development of CoverDrop components was undertaken with a strong emphasis on security.
- The Android and iOS apps correctly leverage the hardware-backed security enclaves in the operating system to protect secrets. For example, the iOS app utilizes *ios-sloth*, supported by the *Secure Enclave*, to securely store sensitive keys on the device, enhancing the overall security posture and safeguarding critical data against unauthorized access.
- The iOS and Android apps ensure no sensitive content leakage via NSLog/logcat, backups or state preservation.
- Additionally, the root/jailbreak detection in place correctly alerts and educates users about the security risks of running the apps in such an environment. Similarly, the Android application incorporates defenses against Tapjacking attacks, alerting users in case of such attempts.

The security posture of the CoverDrop mobile applications will improve with a focus on:

- **Protection of Messages:** It is advised to implement biometrics and strong passphrase prompts to better protect confidential messages from attackers with physical access (COV-01-015).
- **Denial-of-Service (DoS):** It is recommended to implement appropriate fallback mechanisms to better protect users against DoS attacks. The application should implement safe DNS resolution mechanisms with integrity and confidentiality protections (COV-01-013).
- **Hijacking Attacks:** The Android application should mitigate well-known Task Hijacking attacks (COV-01-001).
- **Memory Leakage:** The Android application should mitigate risks related to sensitive information in memory (COV-01-023).
- **General Hardening:** Other less significant findings include utilizing all available platform protections to safeguard sensitive information, such as implementing secure configuration options (COV-01-014) and the hardening of iOS (COV-01-002) and Android (COV-01-019) binaries.

The security of the CoverDrop backend server components and library implementation may be enhanced with a focus on the following areas:

- **Signal Configuration Hardening:** The server-side Signal configuration should be hardened to reduce the attack surface as much as possible, this will mitigate potential Denial of Service (DoS) attacks via attachments (COV-01-028).
- **Signal Chat Hardening:** A number of settings and design decisions can be incorporated to better protect users and mitigate attacks related to the Signal Chat in edge-case scenarios, such as possible Signal Chat eavesdropping and hidden messages (COV-01-029), as well as potential impersonation attacks (COV-01-030).
- **TLS Hardening:** A number of servers support insecure TLS protocols with publicly known security vulnerabilities (COV-01-006). Efforts should be made to address these issues and ensure the TLS configuration is hardened to protect users from Man-In-The-Middle (MitM) attacks.

Both the mobile applications and backend components would also benefit from:

- **Software Patching:** All CoverDrop components should adhere to appropriate software patching procedures, consistently applying security patches in a timely manner (COV-01-003, COV-01-005). In a day and age when a significant portion of code comes from underlying software dependencies, routine patching is crucial to prevent potential security vulnerabilities. Possible automation for this could include tools like *Snyk.io*[127] or *Renovate Bot*[128].

Hardening of CoverDrop backend servers should be prioritized in the following areas:

- **File Permissions**: Files and directories ought to have the minimum necessary permissions for the solution to function, reducing the potential for unauthorized access by unprivileged users (COV-01-021).
- **Encryption of Data at Rest:** All sensitive server data should be appropriately encrypted at rest, significantly reducing the possibility of leaks via server backups or unauthorized access (COV-01-022).
- **Network Configuration:** The network stack configuration should be improved to avoid a number of possible attacks (COV-01-024).
- **SSH Access Hardening:** The SSH configuration could be enhanced implementing MFA and other options (COV-01-025).
- **Separate Logging Server:** A separate logging server ought to be in place to preserve the integrity of captured security related events (COV-01-026).
- **Bootloader Password:** Backend hosts should set a bootloader password (COV-01-020).

Last but not least, implementing these measures will reinforce the security posture of the CoverDrop Cloud infrastructure:

- **Hardening of the Kubernetes Cluster:** The Kubernetes Infrastructure would

[127] https://snyk.io/
[128] https://github.com/renovatebot/renovate

benefit from a configuration better aligned to the *CIS Benchmark* for Kubernetes[129] (COV-01-031), Pod hardening to reduce the potential for privilege escalation attacks (COV-01-032) and implementing outbound traffic restrictions (COV-01-033).

- **Workflow Hardening:** It is important to harden all CI/CD workflows as much as possible. For example, enforcing commit signing for every commit in the Git repositories (COV-01-027) will reduce the potential for supply chain and compromised developer attacks, while the secure storage of tokens will provide protection against unauthorized access and pivoting throughout the infrastructure (COV-01-011).
- **Logging and Monitoring:** It is important to enable and configure correctly all security-relevant AWS tools and features, such as logging and monitoring mechanisms, on load balancers (COV-01-007), as well as throughout all other AWS cloud components (COV-01-010)
- **Attack Surface Reduction:** The cloud infrastructure would benefit from eliminating anything that is not strictly required to ensure adversaries have as little opportunity as possible. For example, unused regions should be deleted (COV-01-009), and outbound traffic ought to be disallowed (COV-01-033).
- **Least Privilege:** Strong consideration should be given to removing broad access roles, utilizing restricted KMS policies, and following the least privilege principle throughout the infrastructure (COV-01-017), as well as ensuring accounts are better isolated from each other to prevent privilege escalation attacks (COV-01-018).

It is advised to address all issues identified in this report, including informational and low severity tickets where possible. This approach will not only significantly enhance the security posture of the platform but also contribute to a reduction in the number of tickets in future audits.

Once all recommendations 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. Future audits should ideally allocate a greater budget, enabling test teams to delve into more complex attack scenarios.

It is suggested to test the application regularly, at least once a year or when substantial changes are deployed, to make sure new features do not introduce undesired security vulnerabilities. Consistently following this approach will lead to a reduction in the number of security issues and fortify the application against online attacks over time.

7ASecurity would like to take this opportunity to sincerely thank Daniel Hugenroth, Dominic Kendrick, Mario Savarese, Sabina Bejasa-Dimmock, Sam Cutler, and the rest

[129] https://www.cisecurity.org/benchmark/kubernetes

of the CoverDrop team, for their exemplary assistance and support throughout this audit. Last but not least, appreciation must be extended to the Open Technology Fund (OTF) for sponsoring this project.