Monospace — Writing and Notes Broken Encryption Feature

Security advisory

2020-04-09

Lena David

Vulnerability description

Presentation of *Monospace — Writing and Notes*

“Minimalistic [Android] *writing and notes app that lets you focus on your content”.*

https://play.google.com/store/apps/details?id=com.underwood.monospace

The issue

Monospace – Writing and Notes provides a feature aimed to let the user password-protect some of their notes. Synacktiv discovered that this feature in such way that the notes are not encrypted, but rather base64-encoded. This makes it possible to retrieve a password-protected note’s content without knowing the password chosen by the user.

It should be noted that the application appears to be unmaintained, as it received its latest update in June 2016.

Affected versions

Version 2.6.3, currently available from Google Play, is known to be affected.

Fix availability

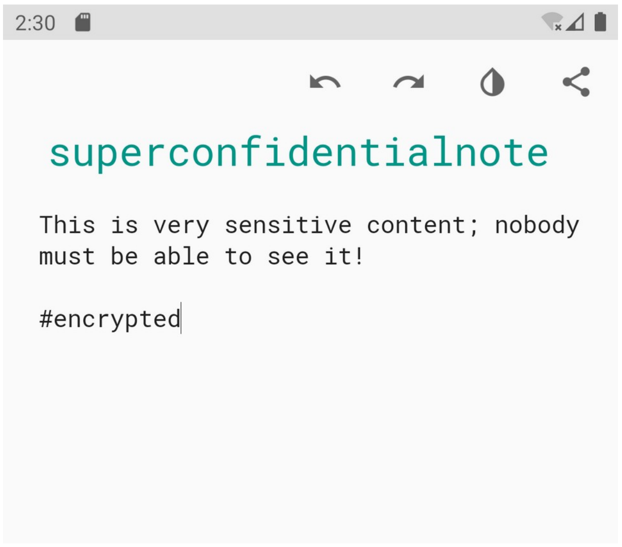
No fixed version of the application has been released at the time of publication of this advisory, nor has any response from the developer been received by Synacktiv.

Timeline

Date	Action
2020-01-09	Advisory sent to *Monospace — Writing and Notes* support (support@underwoodapps.com), suggesting a 90-day deadline.
2020-03-02	In the absence of a response to the email sent in January, direct message sent to the developer on Twitter.
2020-04-09	Publication of this advisory.

Technical description and proof-of-concept

When writing a note, the user can add a "*#encrypted*" string on its last line to indicate it must be stored encrypted on the filesystem. They are then asked to choose a password when exiting the note modification activity:

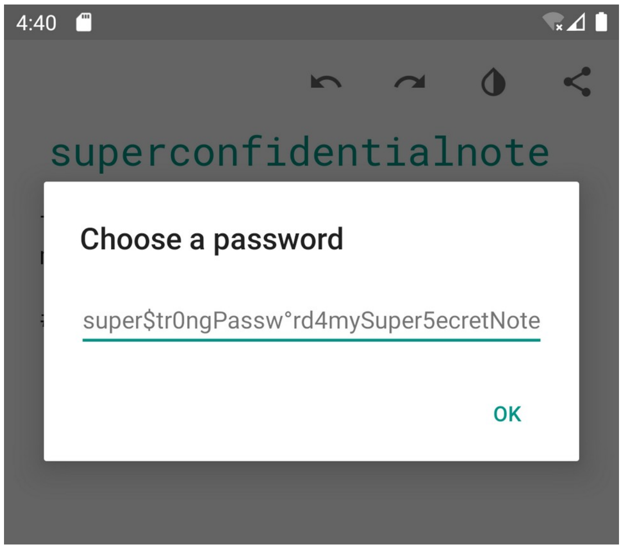

2:30

superconfidentialnote

This is very sensitive content; nobody must be able to see it!

#encrypted

Illustration 1: Writing a note, using *#encrypted* to indicate it must be encrypted.


4:40

superconfidentialnote

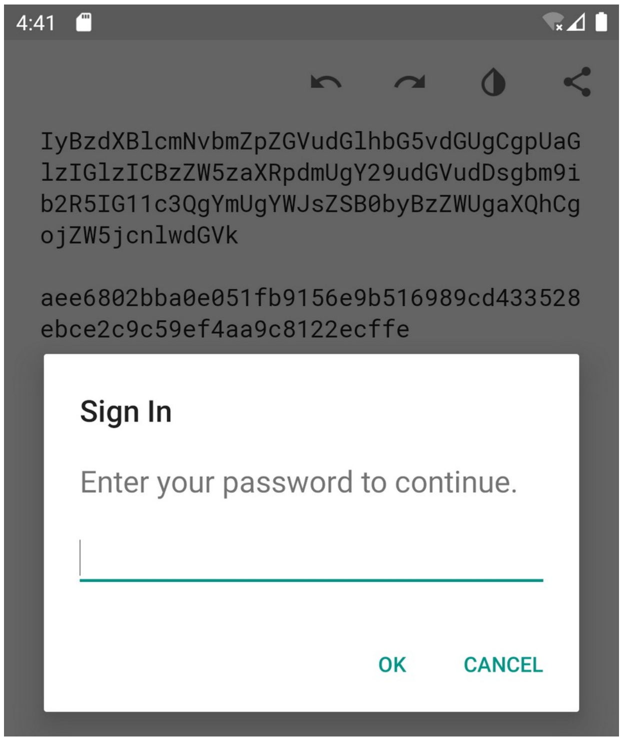
Choose a password

super$tr0ngPassw°rd4mySuper5ecretNote

OK

Illustration 2: Choosing a password to protect the note.

When opening the note again, the user is prompted for a password:


4:41

```
IyBzdXBlcmNvbmZpZGVudGlhbG5vdGUgCgpUaG
lzIGlzICBzZW5zaXRpdmUgY29udGVudDsgbm9i
b2R5IG11c3QgYmUgYWJsZSB0byBzZWUgaXQhCg
ojZW5jcnlwdGVk

aee6802bba0e051fb9156e9b516989cd433528
ebce2c9c59ef4aa9c8122ecffe
```

Sign In

Enter your password to continue.

OK CANCEL

Illustration 3: Prompt for password when opening the note again.

The upper part of the note's content, in the background, looks base64-encoded. Let us look at the contents of the corresponding file:

```
/mnt/runtime/full/emulated/0/Monospace # cat superconfidentialnote.txt
IyBzdXBlcmNvbmZpZGVudGlhbG5vdGUgCgpUaGlzIGlzIHZlcnkgc2Vuc2l0aXZlIGNvbnRlbnQ7
IG5vYm9keSBtdXN0IGJlIGFibGUgdG8gc2VlIGl0IQoKI2VuY3J5cHRlZA==

aee6802bba0e051fb9156e9b516989cd433528ebce2c9c59ef4aa9c8122ecffe
```

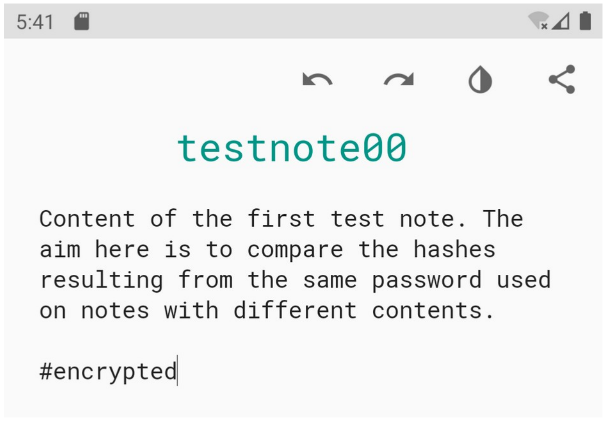
Decoding the upper part of the file's content results in the following:

```
$ echo -ne  "IyBzdXBlcmNvbmZpZGVudGlhbG5vdGUgCgpUaGlzIGlzIHZlcnkgc2Vuc2l0aXZlIGNvbnRlbnQ7IG
5vYm9keSBtdXN0IGJlIGFibGUgdG8gc2VlIGl0IQoKI2VuY3J5cHRlZA==" | base64 -d
# superconfidentialnote

This is very sensitive content; nobody must be able to see it!

#encrypted
```

The note is thus not actually encrypted, but rather encoded in base64. In particular, retrieving the decoded content does not require knowing the password set by the user.

Nevertheless, in the standard workflow of the application, the decoded content of the note is only displayed upon submission of the right password, which suggests it is used to compute the second part of an "encrypted" note's content. As can be seen above, this second part takes the form of a 64-nibble value, and could thus be a SHA256 digest.

Additionally, creating two notes with different contents but an identical password results in the same value:

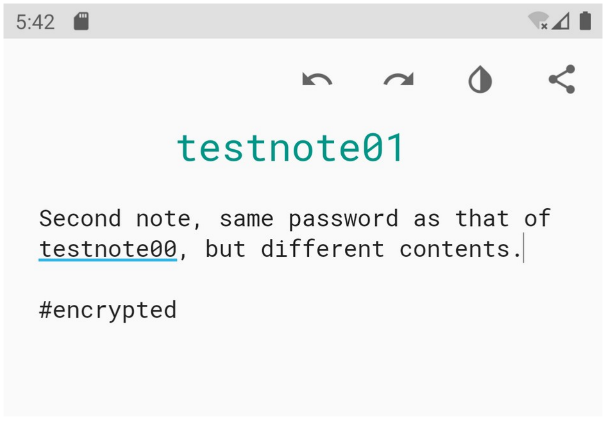

5:41

testnote00

Content of the first test note. The aim here is to compare the hashes resulting from the same password used on notes with different contents.

#encrypted

Illustration 4: First test note, protected with password "testpassword".


5:42

testnote01

Second note, same password as that of testnote00, but different contents.

#encrypted

Illustration 5: Second test note, with a different content but protected by the same password: "testpassword".

The resulting files are as follows:

```
# cat testnote00.txt
IyB0ZXN0bm90ZTAwIAoKQ29udGVudCBvZiB0aGUgZmlyc3QgdGVzdCBub3RlLiBUaGUgYWltIGhl
cmUgaXMgdG8gY29tcGFyZSB0aGUgaGFzaGVzIHJlc3VsdGluZyBmcm9tIHRoZSBzYW1lIHBhc3N3
b3JkIHVzZWQgb24gbm90ZXMgd2l0aCBkaWZmZXJlbnQgY29udGVudHMuCgojZW5jcnlwdGVk

5f3295b696a57bceeb67b02a48f4193822ee3a0477c5a3626a85c4e74f7ccd4e
```

```
# cat testnote01.txt
IyB0ZXN0bm90ZTAxIAoKU2Vjb25kIG5vdGUsIHNhbWUgcGFzc3dvcmQgYXMgdGhhdCBvZiB0ZXN0
bm90ZTAwLCBidXQgZGlmZmVyZW50IGNvbnRlbnRzLgoKI2VuY3J5cHRlZA==

5f3295b696a57bceeb67b02a48f4193822ee3a0477c5a3626a85c4e74f7ccd4e
```

The final lines of both files are identical, which shows they do not depend on the note's content but only on the chosen password.

Code-wise, the involved snippets can be traced from the *save()* method of the *com.underwood.monospace.main.MainActivityFragment* class:

```
com.underwood.monospace.main.MainActivityFragment
public void save(boolean encrypt, boolean sync) {
    if (getActivity() != null && !getActivity().isFinishing()) {
        final Context applicationContext = getActivity().getApplicationContext();
        if (this.mEditText.isEnabled()) {
            final SpannableStringBuilder builder = new
SpannableStringBuilder(this.mEditText.getText());
            final boolean z = encrypt;
            final boolean z2 = sync;
            new Thread(new Runnable() {
                public void run() {
                    if (applicationContext != null) {
                        new WeakReference(applicationContext);
                        SpannableToMarkdown generator = new SpannableToMarkdown(new
WeakReference(applicationContext));
                        if (z) {
                            MainActivityFragment.this.encryptText();
                        }
                        try {
                            MainActivityFragment.this.writeToFile(
(MainActivityFragment.this.mEncryptedText.equals("") || !z) ? generator.toMarkdown(builder) :
MainActivityFragment.this.mEncryptedText,
                            true);
                        } catch (Exception e) {
                            Crashlytics.logException(e);
                            MainActivityFragment.this.writeToFile(builder.toString(), false);
                        }
                        [...]
                    }
                }
            }).start();
        }
    }
}
```

The *encryptText()* method is implemented as shown below:

```
com.underwood.monospace.main.MainActivityFragment
public void encryptText() {
    try {
        this.mEncryptedText = Utils.base64Encode(new SpannableToMarkdown(new
WeakReference(getActivity())).toMarkdown(this.mEditText.getText())) + "\n" + this.mHashedPassword;
    } catch (Exception e) {
        Crashlytics.logException(e);
        Log.e("LOG", "", e);
    }
}
```

This confirms base64 is used on the content of the note, instead of an actual encryption primitive.

As for the final line present in the files and the way it is related to the chosen password, an explanation can be found in the *onInterceptedPause()* method:

```
com.underwood.monospace.main.MainActivityFragment
private void interceptedPause() {
    this.mHasInterceptedPaused = true;
    createHashtags();
    if (!this.mHashtags.contains("encrypted")) {
        save(false, true);
        if (this.mBackPressed) {
            this.mBackPressed = false;
            getActivity().finish();
        }
    } else if (this.mHashedPassword.equals("")) {
        new Builder(getActivity()).title((int)
R.string.choose_a_password).inputType(144).input((CharSequence)
getResources().getString(R.string.password), (CharSequence) "", (InputCallback) new InputCallback() {
            public void onInput(@NonNull MaterialDialog materialDialog, CharSequence charSequence) {
                MainActivityFragment.this.mHashedPassword = Utils.sha256(charSequence.toString());
                MainActivityFragment.this.save(true, true);
                MainActivityFragment.this.getActivity().finish();
            }
        }).build().show();
    } else {
        save(true, true);
        getActivity().finish();
    }
}
```

Looking at the *sha256()* method in *com.underwood.monospace.Utils*, one can see:

```
com.underwood.monospace.Utils
public static String sha256(String base) {
    try {
        byte[] hash = MessageDigest.getInstance("SHA-256").digest((base +
"monospace").getBytes(HttpRequest.CHARSET_UTF8));
        StringBuffer hexString = new StringBuffer();
        for (byte b : hash) {
            String hex = Integer.toHexString(b & 255);
            if (hex.length() == 1) {
                hexString.append('0');
            }
            hexString.append(hex);
        }
        return hexString.toString();
    } catch (Exception ex) {
        throw new RuntimeException(ex);
    }
}
```

Therefore, the final line in a file containing a password-protected note, is the SHA-256 digest of the value obtained by appending the string “monospace” to the password the user chose to protect their note.