KEYSIGHT

Android-SigMorph

Covert Communication Exploiting Android Signing Schemes

I don't get nervous during presentation
Also me :

whois

Ayan Saha

Security Researcher

@ Ex-Keysight

Loves to play with Android, CTFs

Cat lover!

Achute Sharma

Technical Lead Security

@ Keysight

Loves everything security !

Android-SigMorph

Without breaking the signature How can we **misuse** the Signing Schemes of Android, to allow for additional data in the APK file format.

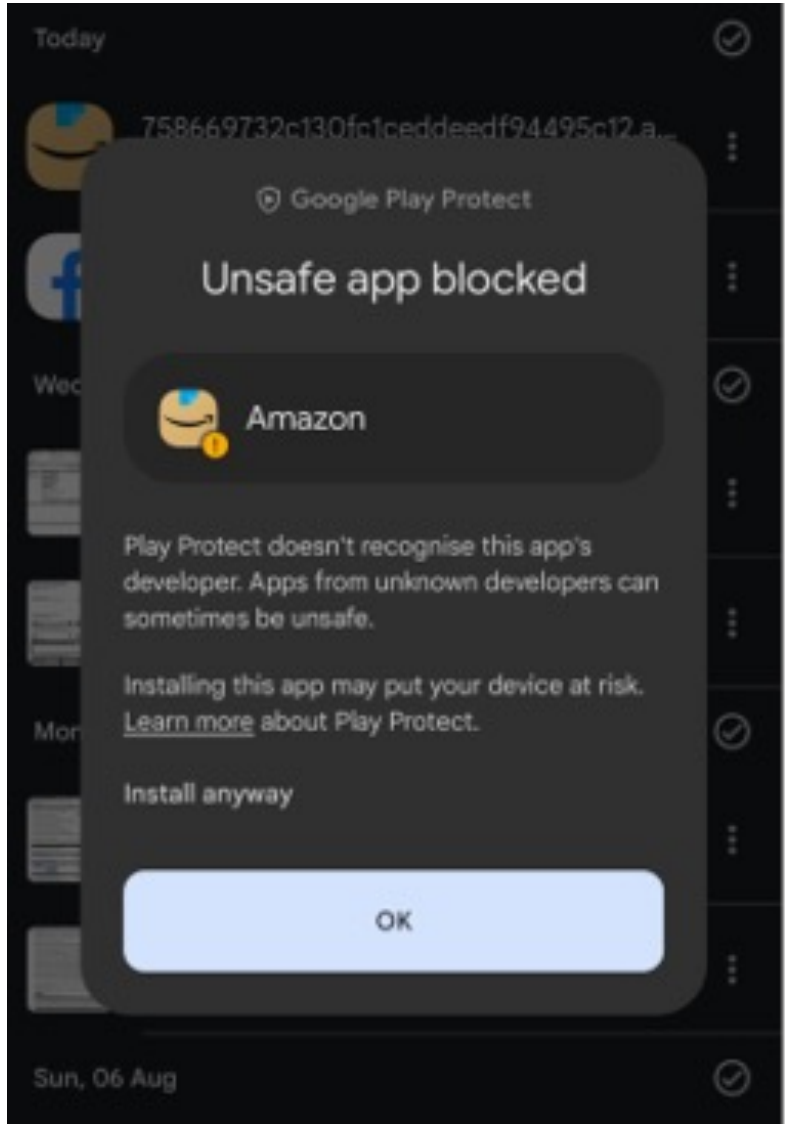
KEYSIGHT

Android App Repackaging

KEYSIGHT

Android App Repackaging

Repackaging an APK



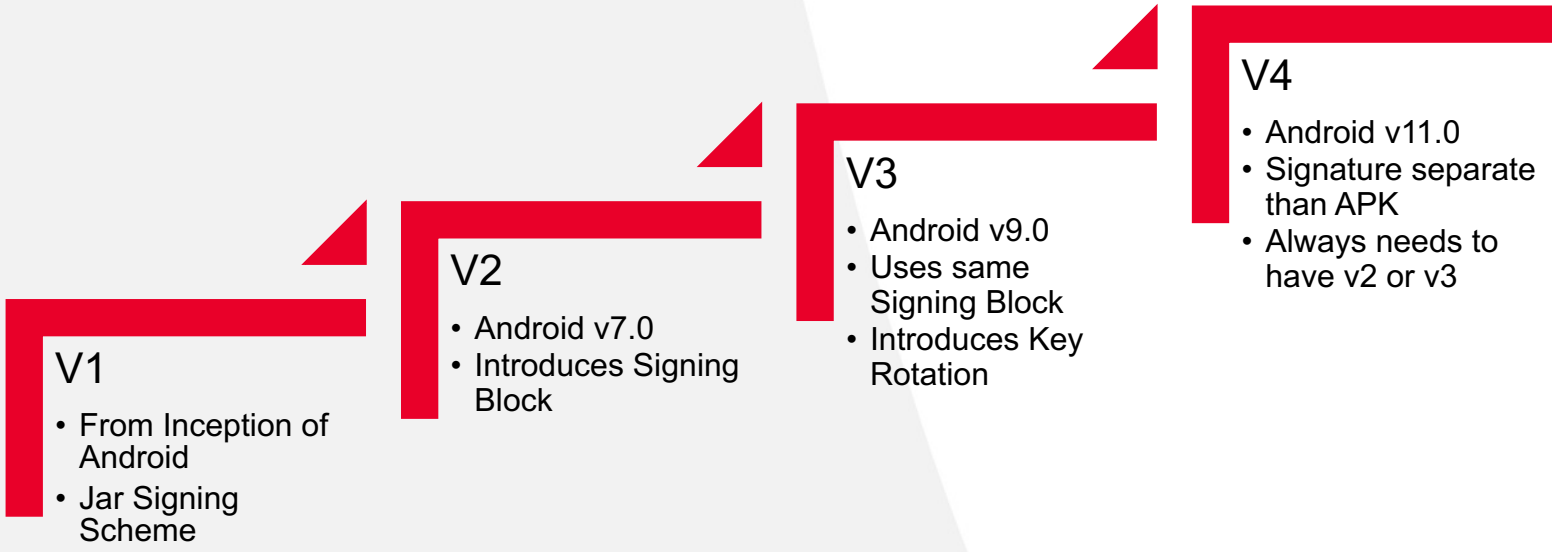
APK Repackaged	An indication of whether an APK file has been repackaged (True) or not (False). AutoFocus marks a repackaged APK file as suspicious because an attacker can repackage a benign file to contain malicious functionality.

Basically, re-packaging can cause lots of **detection signals** to go off.

Without repackaging or breaking the signed information of an APK

What and How much can we change….

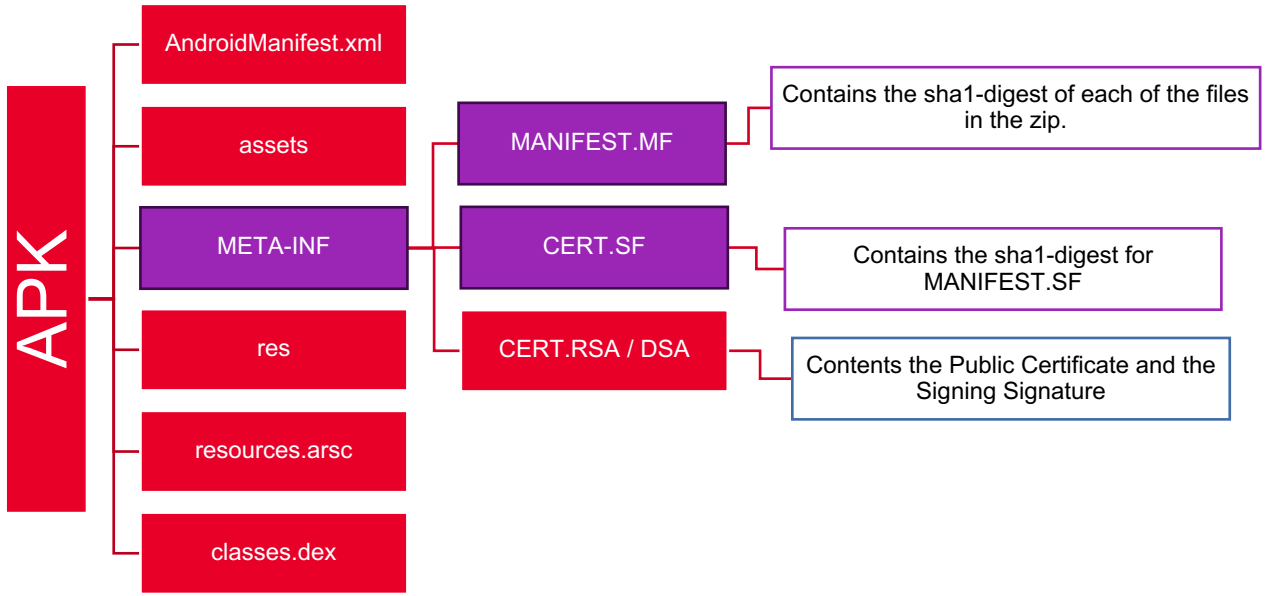
KEYSIGHT



Android Signing Schemes

V1 Signing Scheme

Also known as Jar Signing Scheme




```
research@research-VMware-Virtual-Platform:/tmp/test/META-INF$ cat MANIFEST.MF | head
Manifest-Version: 1.0
Built-By: 1.3.0
Created-By: Android Gradle 1.3.0

Name: res/layout/activity_web.xml
SHA1-Digest: /OGAWy/2sO1BpMqKsoj3ErPtwI8=

Name: AndroidManifest.xml
SHA1-Digest: OSw0Ys2DytXcNPyaGyCCwcIpvUg=

research@research-VMware-Virtual-Platform:/tmp/test/META-INF$ cat /tmp/test/res/layout/activity_
> | openssl dgst -sha1 -binary | base64
/OGAWy/2sO1BpMqKsoj3ErPtwI8=
```

```
research@research-VMware-Virtual-Platform:/tmp/test/META-INF$ cat CERT.SF | head
Signature-Version: 1.0
SHA1-Digest-Manifest: krKpzZo6Cl3Enmb05LmO9ZanNq4=
Created-By: 1.0 (Android)

Name: res/layout/activity_web.xml
SHA1-Digest: FdpcuPUTVPIy1ps6h+LSmbLP0Dc=

Name: AndroidManifest.xml
SHA1-Digest: QW6VQCibYYEZmch3OvIF29PwaFo=

research@research-VMware-Virtual-Platform:/tmp/test/META-INF$ sed -n '5,7p' MANIFEST.MF\
> | openssl dgst -sha1 -binary | base64
FdpcuPUTVPIy1ps6h+LSmbLP0Dc=
```

```
research@research-VMware-Virtual-Platform:/tmp/test/META-INF$ keytool -printcert -file CERT.RSA
Owner: CN=Glaze, OU=Glaze trading india pvt ltd, O=Glaze, L=Delhi, ST=Delhi, C=91
Issuer: CN=Glaze, OU=Glaze trading india pvt ltd, O=Glaze, L=Delhi, ST=Delhi, C=91
Serial number: 14124d43
Valid from: Thu Jul 06 11:25:06 IST 2017 until: Mon Jun 30 11:25:06 IST 2042
Certificate fingerprints:
         SHA1: F1:57:2C:E2:B6:CD:33:1C:EA:99:3C:68:01:5D:4D:96:20:75:AA:D3
         SHA256: 6E:6A:C8:8D:4C:92:BF:27:5F:20:5D:63:FF:7B:3C:6B:3A:98:4A:FB:8C:D0:82:72:4E:C4:1E:
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3
```

V1 Problems – Skipped Files

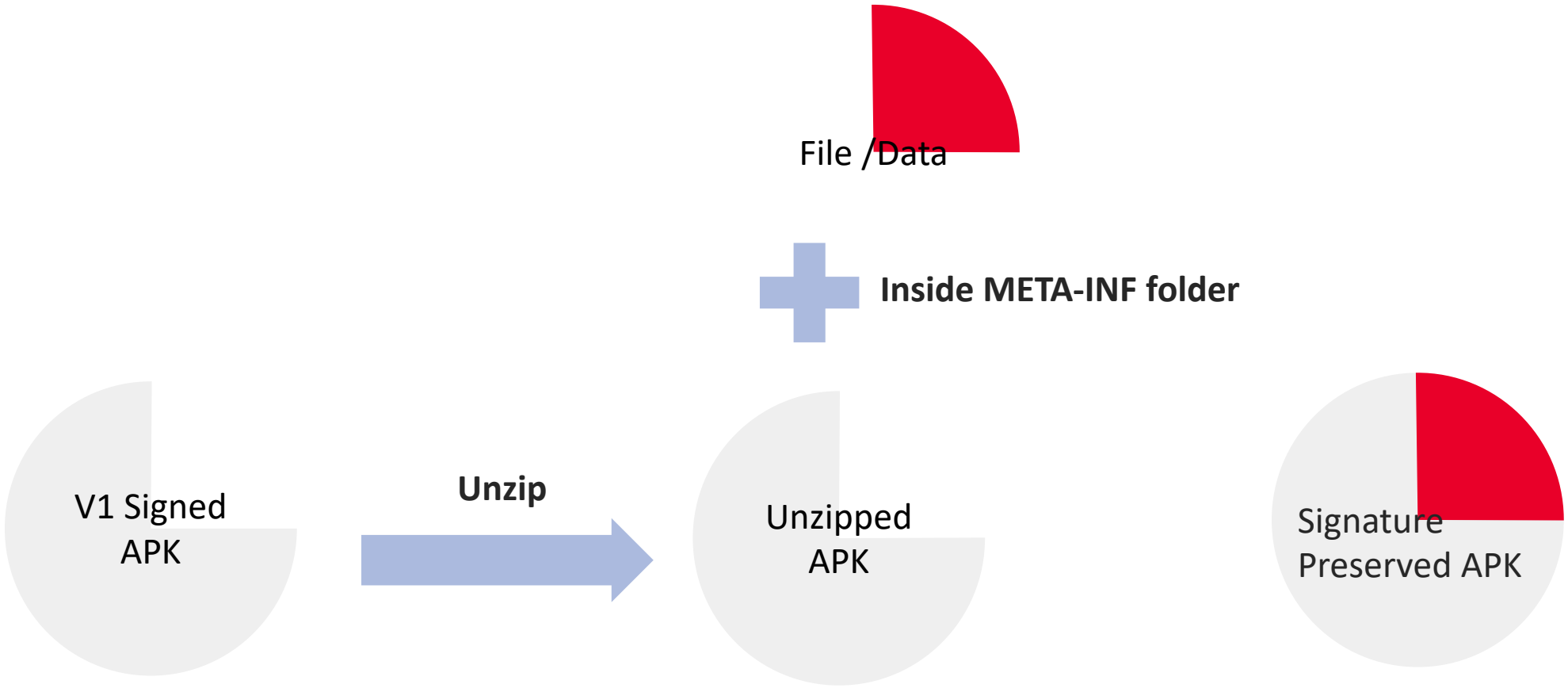
- The JAR signing scheme skips files in META-INF

```
WARNING: META-INF/com/android/build/gradle/app-metadata.properties not protected by signature. Unauthorized modifications to this JAR entry w
 the entry outside of META-INF/.
WARNING: META-INF/services/com.twitter.database.hydrator.HydrationRegistry$Registrar not protected by signature. Unauthorized modifications t
ed. Delete or move the entry outside of META-INF/.
WARNING: META-INF/services/com.twitter.model.json.common.JsonModelRegistry$Registrar not protected by signature. Unauthorized modifications t
ed. Delete or move the entry outside of META-INF/.
WARNING: META-INF/services/kq6 not protected by signature. Unauthorized modifications to this JAR entry will not be detected. Delete or move
WARNING: META-INF/services/kyd not protected by signature. Unauthorized modifications to this JAR entry will not be detected. Delete or move
WARNING: META-INF/services/ur2 not protected by signature. Unauthorized modifications to this JAR entry will not be detected. Delete or move
WARNING: META-INF/services/x5a not protected by signature. Unauthorized modifications to this JAR entry will not be detected. Delete or move
```

- APKs loading files from META-INF might get **replaced**.

- Most APK has v2 or newer schemes, so not so dangerous but some does still have v1 only and has files in META-INF.

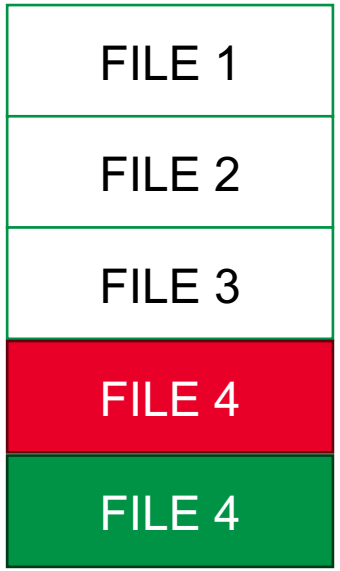
V1 Problems – Skipped Files – Add Extra Files



V1 Problems – CVE-2013-4787 – “Master Key Vuln”

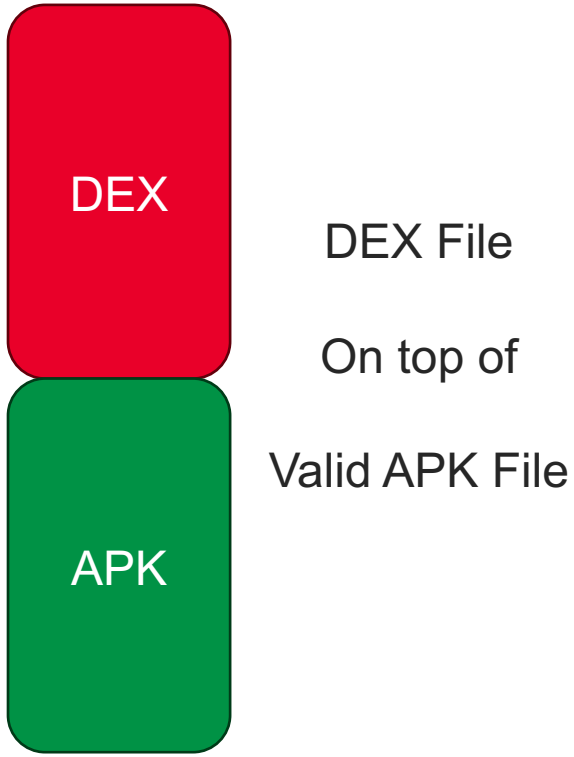
If there are 2 files with the same name.

The Android Runtime runs the first and ignore the second.



V1 Problems – JANUS Vulnerability

- CVE-2017–13156. Affects Android 5.0 < 8.1
- DEX prepended to a valid, signed APK file bypasses verification since those bytes are ignored in v1.
- ART which can load both DEX and APK, loads the malicious prepended DEX.
- Makes malicious DEX acceptable as an update to existing privileged apps like system apps.



V2 Signing Scheme

APK Signature Scheme v2

APK Signature Scheme v2 is a whole-file signature scheme that increases verification speed and strengthens integrity guarantees by detecting any changes to the protected parts of the APK.

Signing using APK Signature Scheme v2 inserts an APK Signing Block into the APK file immediately before the ZIP Central Directory section. Inside the APK Signing Block, v2 signatures and signer identity information are stored in an APK Signature Scheme v2 Block.

source.android.com/docs/security/features/apksigning/v2

V2 / V3 Signing Scheme

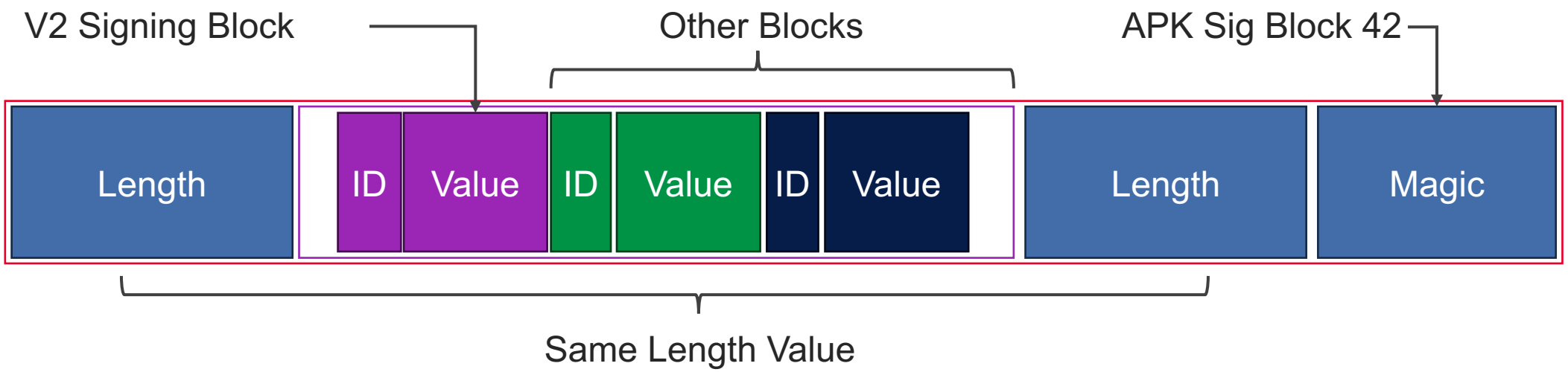
Before Signing

V2 / V3 Signing Scheme

Before Signing

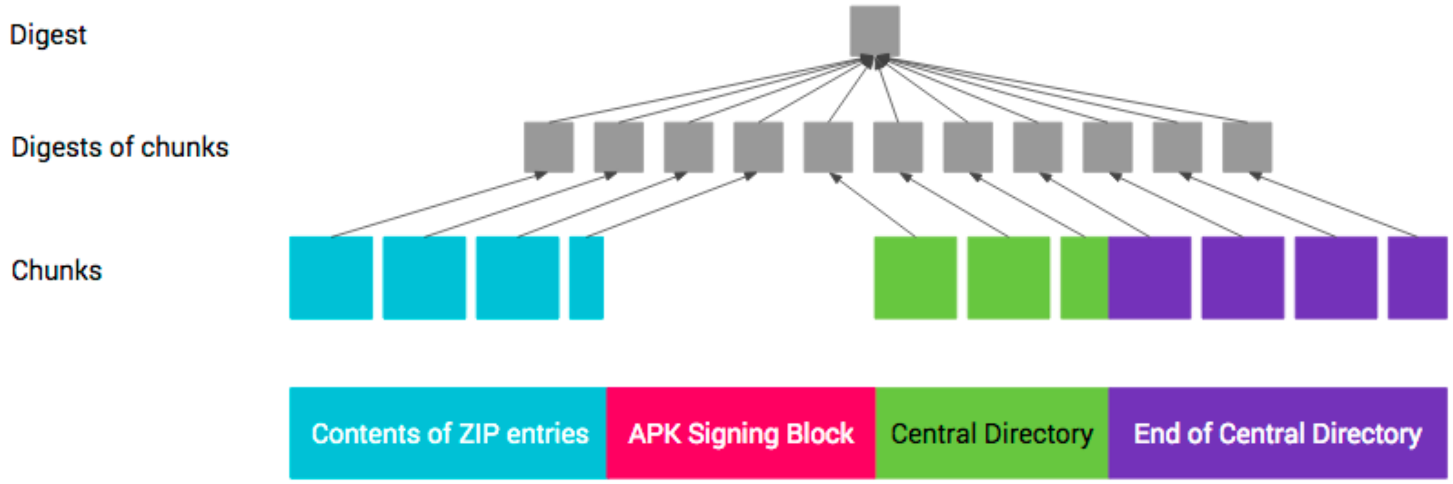
After Signing

APK Signing Block



V2 Block ID:0x7109871a


```
    '1a870971': [
        {
            'digest_cert_attr': {
                'digests': [
                    {
                        'algoID': 259,
                        'DigestBytes':
b'\xa2u\xc8\x1a\xe7\xc8\xd0Oq\xdf\xb7F\x02)9\xb0,\xa8\xc7\x1c"\xa9^\xac\xa7H\x
ee\xe5z\x16\x97z'
                    }
                ],
                'certs': [
                    b'MIIDNTCCAh2gAwIBAgIIDD+OTo4rNP4wDQYJKoZIhvcNAQELBQAwSDEQ
MA4GA1UEBhMHQW5kcm9pZDEQMA4GA1UEChMHQW5kcm9pZDEQMA4GA1UECxMHQW5kcm9pZDEQMA4GA1
UEAxMHQW5kcm9pZDAgFw0yMzAxMTQwNzMzNDRaGA8yMDc3MTAxNzA3MzM0NFowSDEQMA4GA1UEBhMH
QW5kcm9pZDEQMA4GA1UEChMHQW5kcm9pZDEQMA4GA1UECxMHQW5kcm9pZDEQMA4GA1UEAxMHQW5kcm
9pZDCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAMZ2iqVU7Ch7R1ZR6P/9X8TGNjckhgz1
clOjvOuFdl5AKwvCRTZ4JAHoteVV/ZA3cqM072iZdFuj1wJ2D0PEpgwkSr4bTvPL4fiCdp4RF30nmf
K9yyUZCvqB8bmpa34U7S+DxUbnBnDOSzxux6OjC131VdruRuFEKOGJTQ6n7anDAtEerrYhk44cmy//
zRz5HiVtH42DSeYdDNRcCqgeedKRWnbBgUHuxeSDFfVLlUjsSgZ6+OwF1aTn1scUWJEvTlfUOIAj+O
nwMzZx5jlErIQ2RtPB9s27hHiClG03Q7QvASUof5httKK/Xlsd7NTMI+JKstuaq2GqQWpWGrO75HMC
AwEAAaMhMB8wHQYDVR0OBBYEFDoOE2iLclgAFG4EjF+SsEeWcw3MMA0GCSqGSIb3DQEBCwUAA4IBAQ
CRhRLmmOgUrlgHJHIQVBPBhocq0+t5dAQTQLO/RbWz3QEtNYW4a2H5ctt/D74z1u93POf7uNtGZFdj
CDihOH74ABeqT2E2pYPM2nNzjESXP4+OqstjSxATR4iQxMFhUB7W7E8ct3DTDE/naOpSh9A1kDZiLV
jgS24Xyx+RS9y6JY+Dl0ifVYCdN3VKRyKVOqylqVSohtabT6TNxCfWR6KGbVdn540qpuhYWMhjGxPV
rf6U/ecm/p4T2WDmy89I1EZeZH0ch7J7qHbZQ7oopjMyR/X/qSzaTg4BFr00F5qU5MF+R2yZM0VkVO
iB8VJ5fOI7gx4PDfM9ikVwR6BCxFeP'
                ],
                'attrs': []
            },
            'signatures': [
                {
                    'algoID': 259,
                    'signature':
b'\xc6\x19\xb6\x80t\x98f\xde\xc1\xd4\x88.X\xb0\xa0\xd2[6\xf5=:$\x01\xc3>\xb4\x
cd\xcb*\xc7\xac"\xb9\xd2\xf8/%\xbf?\xdb\xac\x1c\xdb\xf1[\xa2\xb9{21\xa1>\xea\x
b3\x13\x1f\xf6\x8c\x812GN\xab\xe4+\xbe4\xbaHy
\xcc!\xbahF\x88(\x00O\xcb\x9eO\x14\xc6\x893X\xfcC\xa3B\xb0D+N6\r,@m\x1f\x0b\x9
7S,\x83H\xb2u\xf7\x87Pq\tW<\xf1\xfb\xa1\xe8\xc6\x0e2\x01\xc7\x1e\xe2\x03\x02\x
ce(\x9b\x15W\x17&\xb0d\xbcn6\xd1r2#c\xae\x03\r\xb3P1\xe3Ey\x05\xab\xa3o\x9a\x1
e\xcd\xc3\x147D\xd6\xc9\x836\xf7\x83\xca\xc5\xae#\n\x06\x80(\x84A@
\xb4{sdv\x19MX\xd9I\x7fE`L>\x14*\xc7h\x17\x1d\x0f\x85\x1dy\x8a\xdf*T\x13\xb5\x
9e\xbfz\xd4\xb5\xf5\xfb\xa1\xc0\xbb|\x8e\xbc\x8d\xe4F\xbd\xd5_\xaaC\xe7^0c[\x8
d\x1cgY!\xba\xbc\xae\x1f\xcfC\xf9.$'
                }
            ],
            'public_key':
b'MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAxnaKpVTsKHtHVlHo//1fxMY2NySGDPVy
U6O864V2XkArC8JFNngkAei15VX9kDdyozTvaJl0W6PXAnYPQ8SmDCRKvhtO88vh+IJ2nhEXfSeZ8r
3LJRkK+oHxualrfhTtL4PFRucGcM5LPG7Ho6MLXfVV2u5G4UQo4YlNDqftqcMC0R6utiGTjhybL//N
HPkeJW0fjYNJ5h0M1FwKqB550pFadsGBQe7F5IMV9UuVSOxKBnr47AXVpOfWxxRYkS9OV9Q4gCP46f
AzNnHmOUSshDZG08H2zbuEeIKUbTdDtC8BJSh/mG20or9eWx3s1Mwj4kqy25qrYapBalYas7vkcwID
AQAB'
        }
    ],
    '77657242':
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
```

Digests - SHA256 of the APK chunks

Public key certs - Key used to sign the digest

Signature - Signature after signing the digest

V3 Block ID: 0xf05368c0

APK Signature Scheme v3 Block

The v3 scheme is designed to be very similar to the v2 scheme. It has the same general format and supports the same signature algorithm IDs, key sizes, and EC curves.

However, the v3 scheme adds information about the supported SDK versions and the proof-of-rotation struct.

Format

APK Signature Scheme v3 Block is stored inside the APK Signing Block under ID `0xf05368c0`.

Proof-of-rotation and self-trusted-old-certs structs

The proof-of rotation struct allows apps to rotate their signing cert without being blocked on other apps with which they communicate. To accomplish this, app signatures contain two new pieces of data:

- assertion for third parties that the app's signing cert can be trusted wherever its predecessors are trusted
- app's older signing certs which the app itself still trusts

source.android.com/docs/security/features/apksigning/v3

Verity Padding Block ID:0x42726577

Used to increase the size of the Signing block (including the length and magic) to a multiple 4096.

tools/apksig/src/main/java/com/android/apksig/internal/apk/ApkSigningBlockUtils.java

ApkSigningBlockUtils.java

```
855
856         int blocksSize = 0;
857         for (Pair<byte[], Integer> schemeBlockPair : apkSignatureSchemeBlockPairs) {
858             blocksSize += 8 + 4 + schemeBlockPair.getFirst().length; // size + id + value
859         }
860
861         int resultSize =
862                 8 // size
863                 + blocksSize
864                 + 8 // size
865                 + 16 // magic
866                 ;
867         ByteBuffer paddingPair = null;
868         if (resultSize % ANDROID_COMMON_PAGE_ALIGNMENT_BYTES != 0) {
869             int padding = ANDROID_COMMON_PAGE_ALIGNMENT_BYTES -
870                     (resultSize % ANDROID_COMMON_PAGE_ALIGNMENT_BYTES);
871             if (padding < 12) {  // minimum size of an ID-value pair
872                 padding += ANDROID_COMMON_PAGE_ALIGNMENT_BYTES;
873             }
874             paddingPair = ByteBuffer.allocate(padding).order(ByteOrder.LITTLE_ENDIAN);
875             paddingPair.putLong(padding - 8);
876             paddingPair.putInt(VERITY_PADDING_BLOCK_ID);
877             paddingPair.rewind();
878             resultSize += padding;
879         }
880
```

Verity Padding Block ID:0x42726577

Used to increase the size of the Signing block (including the length and magic) to a multiple 4096.

tools/apksig/src/main/java/com/android/apksig/internal/apk/ApkSigningBlockUtils.java

ApkSigningBlockUtils.java

```
855
856         int blocksSize = 0;
857         for (Pair<byte[], Integer> schemeBlockPair : apkSignatureSchemeBlockPairs) {
858             blocksSize += 8 + 4 + schemeBlockPair.getFirst().length; // size + id + value
859         }
860
861         int resultSize =
862                 8 // size
863                 + blocksSize
864                 + 8 // size
865                 + 16 // magic
866                 ;
867         ByteBuffer paddingPair = nu
868         if (resultSize % ANDROID_CO
869             int padding = ANDROID_C
870                     (resultSize % A
871             if (padding < 12) {  //
872                 padding += ANDROID_
873             }
874             paddingPair = ByteBuffe
875             paddingPair.putLong(pad
876             paddingPair.putInt(VERITY_PADDING_BLOCK_ID);
877             paddingPair.rewind();
878             resultSize += padding;
879         }
880
```

```
long centralDirOffset = ZipUtils.getZipEocdCentralDirectoryOffset(eocd);
long signingBlockSize = centralDirOffset - beforeApkSigningBlock.size();
if (signingBlockSize % ANDROID_COMMON_PAGE_ALIGNMENT_BYTES != 0) {
    throw new RuntimeException(
            "APK Signing Block size is not multiple of page size: " +
            signingBlockSize);
}
```

Source Stamp Block ID:0x6dff800d

Includes metadata such as timestamp of the build, the version of the build tools, source code's git commit hash etc.

Basically: Version Control information.

Source Stamp Block ID:0x6dff800d

Includes metadata such as timestamp of the build, the version of the build tools, source code's git commit hash etc.

Basically: Version Control information.

The source stamp is stored in a file called stamp-cert-sha256, present in the APK.

This is matched with the digest from the SOURCE_STAMP_BLOCK

```
research@research-VMware-Virtual-Platform:/tmp/research$ ls
AndroidManifest.xml  assets  classes.dex  META-INF  res  resources.arsc  stamp-cert-sha256
research@research-VMware-Virtual-Platform:/tmp/research$
research@research-VMware-Virtual-Platform:/tmp/research$ cat stamp-cert-sha256 | xxd
00000000: 0823 2f40 2cd8 ad9d 49fc b1f7 5d6b 6a74  .#/@,...I...]kjt
00000010: 78bf 2788 acfb b540 6e8d 7be7 c534 c4a8  x.'....@n.{..4..
research@research-VMware-Virtual-Platform:/tmp/research$
```

Google Play Frosting ID:0x2146444e

Introduced in 2018 – to prove that an APK originated from the Play Store.

Protobuff encoded and signed with Google’s private key to prove authenticity.

Security metadata in early 2018

Next year we'll begin adding a small amount of security metadata on top of each APK to verify that it was officially distributed by Google Play. Often when you buy a physical product, you'll find an official label or a badge which signifies the product's authenticity. The metadata we're adding to APKs is like a Play badge of authenticity for your Android app.

No action is needed by developers or users. We'll adjust Play's maximum APK size to take into account the small metadata addition, which is inserted into the APK Signing Block and does not alter the functionality of your app. In addition to enhancing the integrity of Play's mobile app ecosystem, this metadata will enable new distribution opportunities for developers in the future and help more people keep their apps up to date.

Google Play Frosting ID:0x2146444e

```
research@research-VMware-Virtual-Platform:~/notes/conferences/nullcon
/avast_tool/test$ go run main.go ~/nullcon/com.facebook.katana.apk

Verification scheme used: v2
Frosting verified
research@research-VMware-Virtual-Platform:~/notes/conferences/nullcon
/avast_tool/test$ go run main.go ~/nullcon/com.facebook.katana_frosti
ng_single_byte_changed.apk

Verification scheme used: v2
Frosting not verified : invalid frosting signature
```


github.com/avast/apkverifier

Google Play Frosting ID:0x2146444e

```
research@research-VMware-Virtual-Platform:~/notes/conferences/nullcon/avast_tool/test$
go run main.go ~/nullcon/facebook_frost_added_in_pegasus.apk
Verification failed: This apk has 'x-android-apk-signed: 2, 3', cannot be verified usin
g v1 scheme, downgrade attack?

Verification scheme used: v1
Frosting not verified : frosting apk file digest mismatch
research@research-VMware-Virtual-Platform:~/notes/conferences/nullcon/avast_tool/test$
```

Google Play Frosting ID:0x2146444e

There is a SHA256 Signature so we cant just put this on random APKs.

```
├─ metadata array, pick first non-disabled
    ├─ entry size (varint)
    ├─ disabled flag (varint), checked ≠ 0 in Play Store
    ├─ public key index (varint), from finsky.peer_app_sharing_api.frosting_public_keys comma separated array
    ├─ fileSha256 (digest over the apk before signing block, schemev2 signing block and eocd, see verifyApk()
    ──────────────────────────── SIGNED DATA END ────────────────────────────
│
├─ size of the signatures array (varint)
├─ signatures array, indexing matches the metadata array
    ├─ entry size (varint)
    ├─ signature (byte array)
──────────────────────────────────────────────────────────────────────────


All the size fields are in bytes and counted excluding the size varint itself.

The SIGNED DATA end after the metadata entry you're checking againts, even if it is not the last entry.
They are verified against signature created by frostingPublicKeys key. It uses ECDSAWithSHA256 algorithm.
The fileSha256 hash must match the APK for the frosting to be valid.
```

```go
// For the purposes of integrity verification, ZIP End of Central Directory's field Start of
// Central Directory must be considered to point to the offset of the APK Signing Block.
eocd := make([]byte, len(eocdOrig))
copy(eocd, eocdOrig)
binary.LittleEndian.PutUint32(eocd[eocdCentralDirOffsetOffset:], uint32(signingBlockOffset))
hasher.Write(eocd)

if !bytes.Equal(hasher.Sum(nil), f.fileSha256) {
  return ErrFrostingDigestMismatch
}
```

Dependency Info Block ID:0x504b4453

Block that contains dependency metadata, which is saved by the Android Gradle plugin to identify any issues related to dependencies.

We Can Change this:

- but not much malicious use case here ?

Stripping it:
- will not allow Play Console to Analyze it.

Dependencies metadata

When building your app using Android Gradle plugin 4.0.0 and higher, the plugin includes metadata that describes the dependencies that are compiled into your app. When uploading your app, the Play Console inspects this metadata to provide you with the following benefits:

- Get alerts for known issues with SDKs and dependencies your app uses
- Receive actionable feedback to resolve those issues

The data is compressed, encrypted by a Google Play signing key, and stored in the signing block of your release app. However, you can inspect the metadata yourself in the local intermediate build files in the following directory: `<project>/<module>/build/outputs/sdk-dependencies/release/sdkDependency.txt`.

If you'd rather not share this information, you can opt-out by including the following in your module's `build.gradle` file:

```
android {
  dependenciesInfo {
      // Disables dependency metadata when building APKs.
      includeInApk = false
      // Disables dependency metadata when building Android App Bundles.
      includeInBundle = false
  }
}
```

developer.android.com/build/releases/past-releases/agp-4-0-0-release-notes#dependency-metadata

Zero Block ID:0xff3b5998

Value is always \x00 * 4084

53022728	F80F0000	00000000	98593BFF	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
53022796	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
53022864	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
53022932	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
53023000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
53023068	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
53023136	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
53023204	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
53023272	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
53023340	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
53023408	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
53023476	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
53023544	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
53023612	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
53023680	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
53023748	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
53023816	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
53023884	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
53023952	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
53024020	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
53024088	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
53024156	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
53024224	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
53024292	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
53024360	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
53024428	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

We only found this on Facebook / Instagram APKs.

Zero Block ID:0xff3b5998

Value is always \x00 * 4084

```
53022728 F80F0000 00000000 98593BFF 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
53022796 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
53022864 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
53022932 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
53023000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
53023068 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
53023136 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
53023204 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
53023272 00000000 00000000 00000000 (
53023340 00000000 00000000 00000000 (
53023408 00000000 00000000 00000000 (
53023476 00000000 00000000 00000000 (
53023544 00000000 00000000 00000000 (
53023612 00000000 00000000 00000000 (
53023680 00000000 00000000 00000000 (
53023748 00000000 00000000 00000000 (
53023816 00000000 00000000 00000000 (
53023884 00000000 00000000 00000000 (
53023952 00000000 00000000 00000000 (
53024020 00000000 00000000 00000000 (
53024088 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
53024156 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
53024224 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
53024292 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
53024360 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
53024428 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

Facebook / #3b5998 Hex Color Code

The hexadecimal color code #3b5998 is a shade of blue. In the RGB color model #3b5998 is comprised of 23.14% red, 34.9% green and 59.61% blue. In the HSL color space #3b5998 has a hue of 221° (degrees), 44% saturation and 41% lightness. This color has an approximate wavelength of 472.93 nm.

We only found this on Facebook / Instagram APKs.

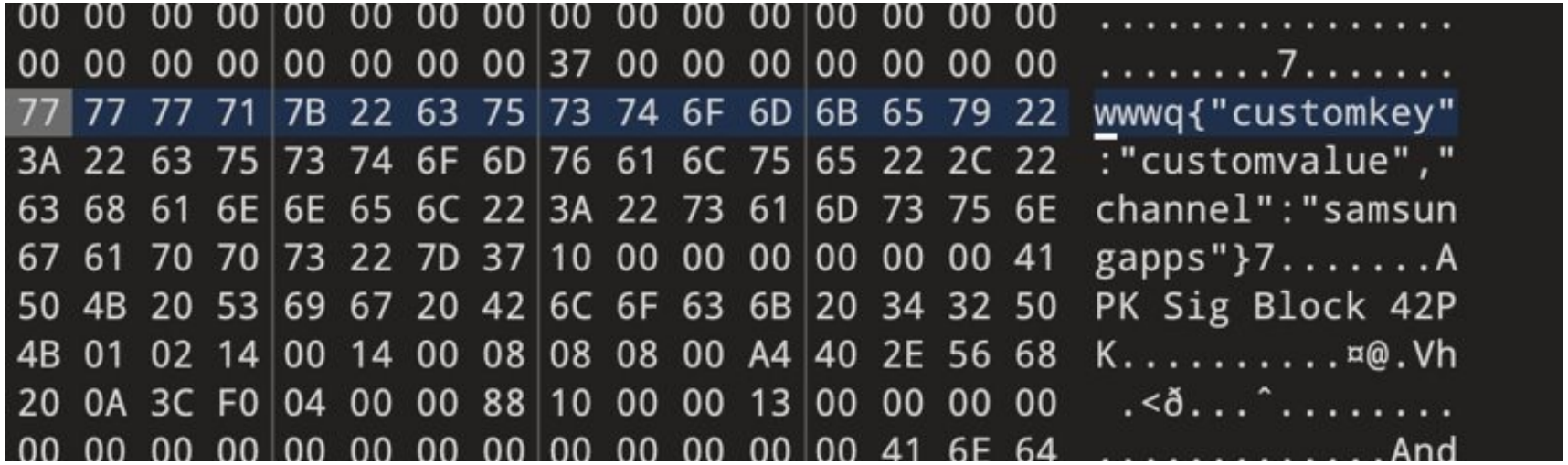
APK Channel Block ID:0x71777777

Used to track channels of distribution for an APK, mostly Chinese APKs have this.

Can be added to any app through this framework called walle
https://github.com/Meituan-Dianping/walle

Channel specific information in key-value pairs, which is not encrypted.

You are most welcomed to change it ☺




```
1   meituan # 美团
2   samsungapps #三星
3   hiapk
4   anzhi
5   xiaomi # 小米
6   91com
7   gfan
8   appchina
9   nduoa
10  3gcn
11  mumayi
12  10086com
13  wostore
14  189store
15  lenovomm
16  hicloud
17  meizu
18  wandou
19  # Google Play
20  # googleplay
21  # 百度
22  baidu
23  #
24  # 360
25  360cn
26  #
27  # 应用宝
28  myapp
```

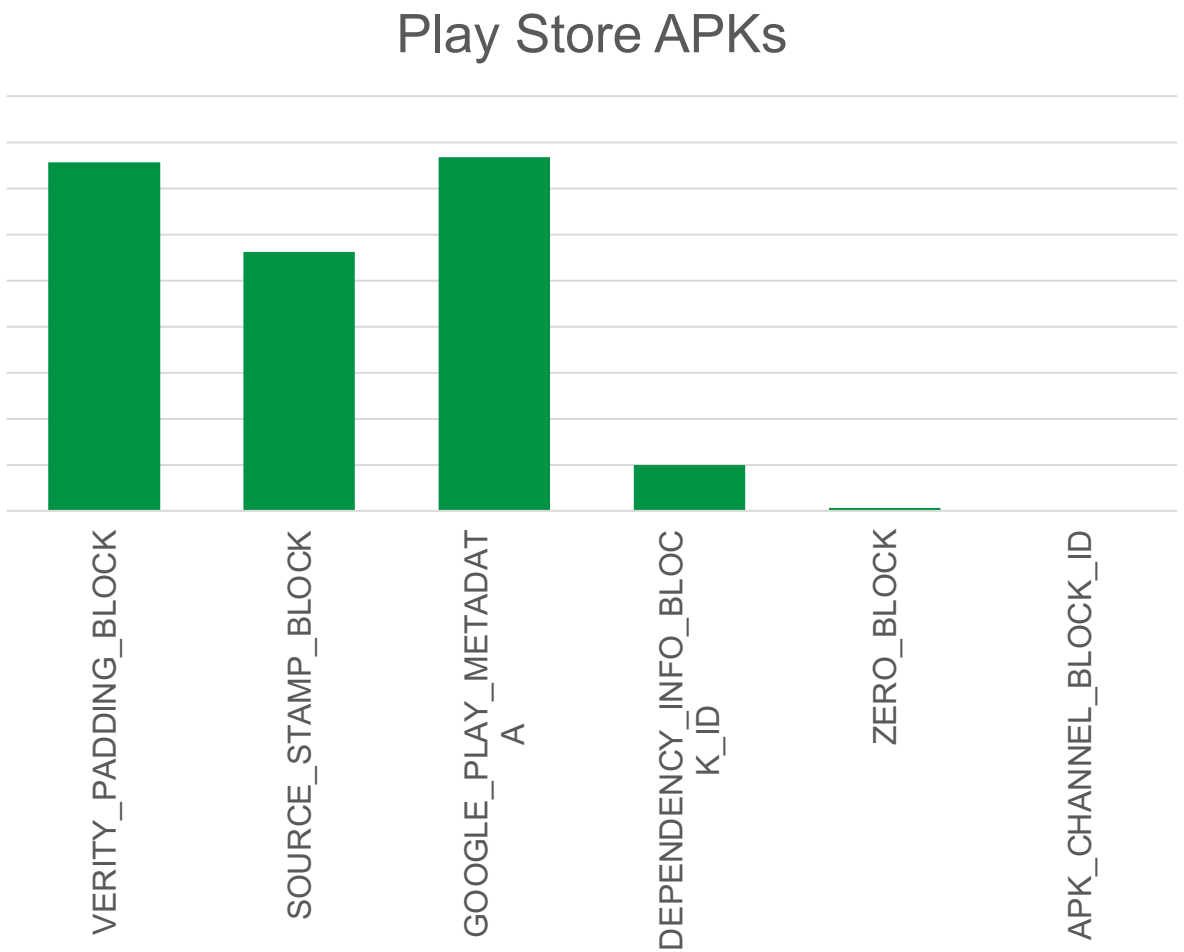
Demo 1

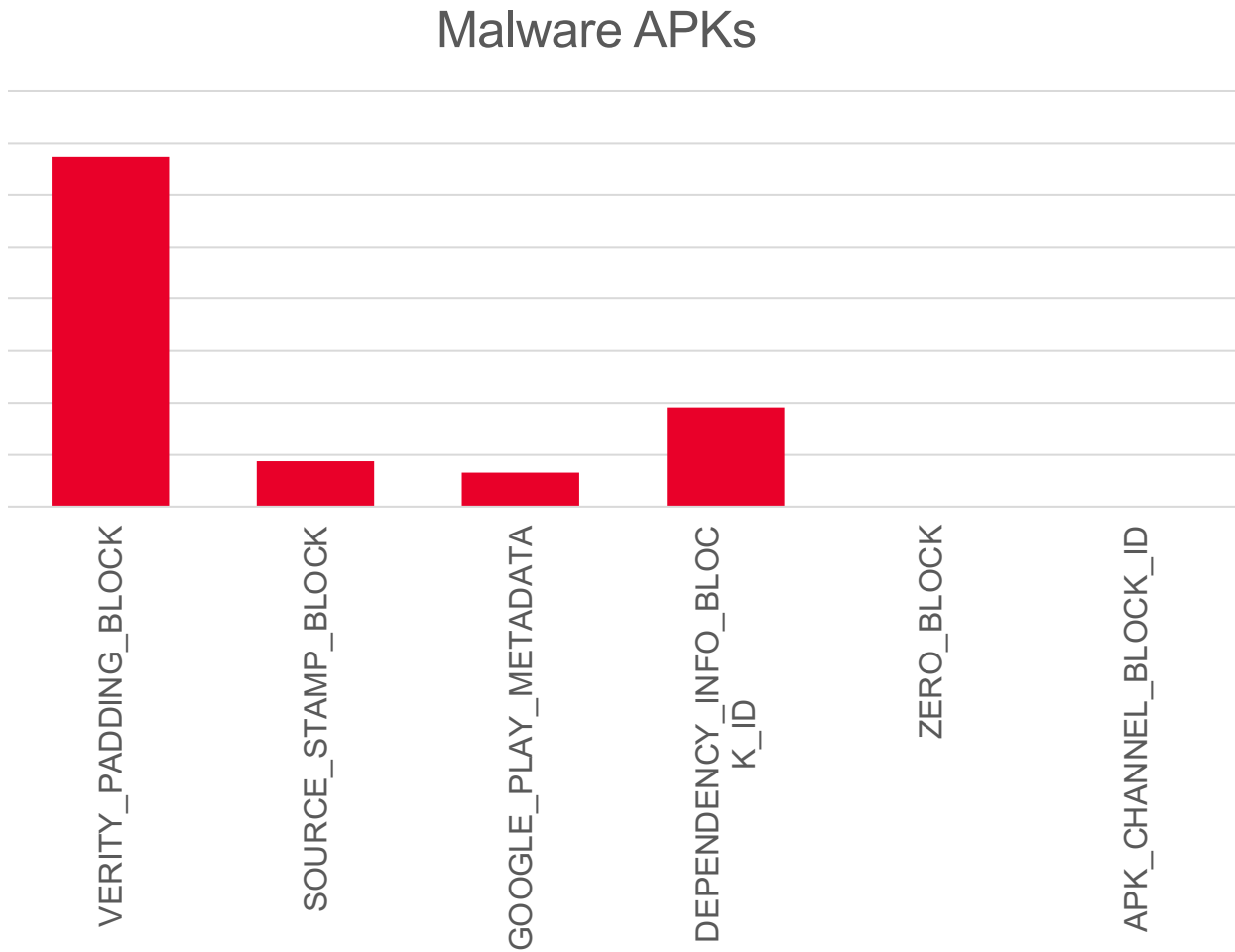
Showcase how modifying the APK Signing Block still retain the verification status.

KEYSIGHT

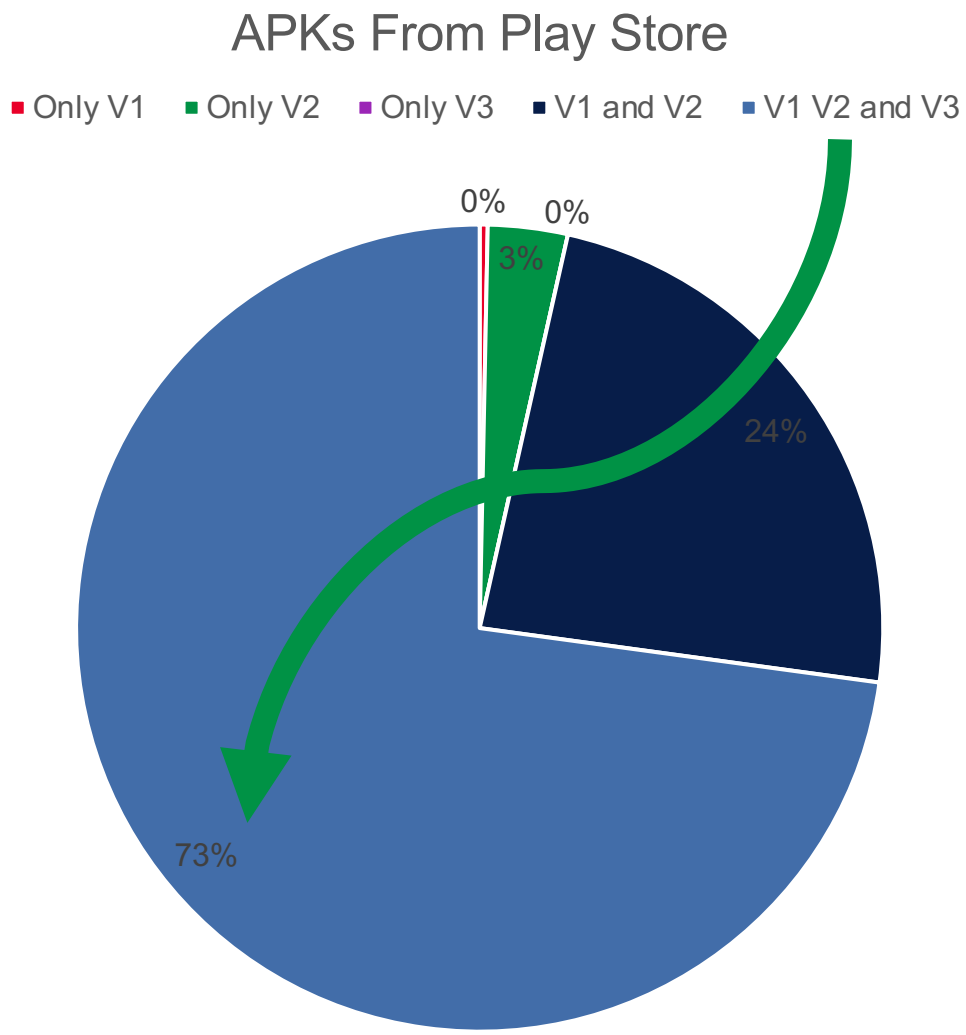
Cool, do APKs have these ‘other’ blocks ?

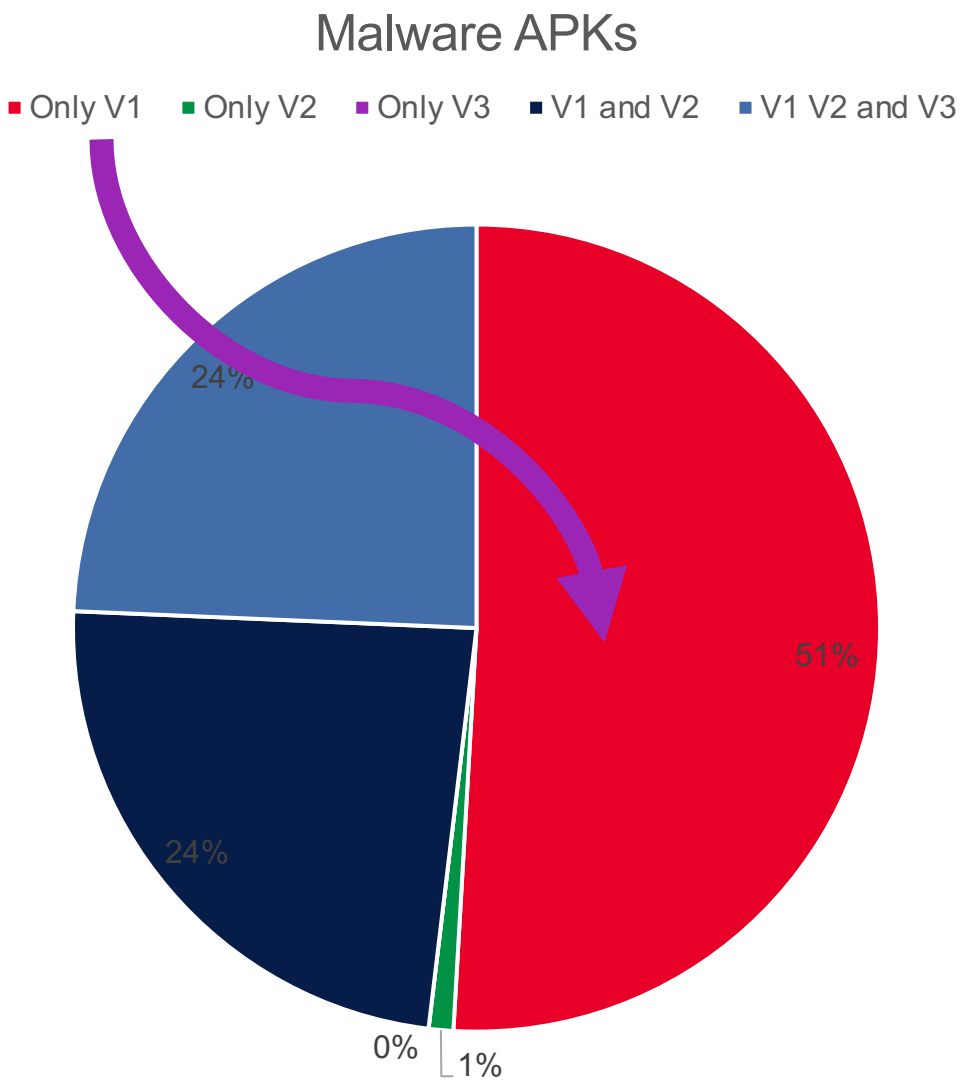
Block Usage Survey





Exclusive Signing Version Distribution





How much can we change / modify ?

Initially we thought that it depends upon the existing size of the APK Signing Blocks, and the type and number of other blocks present ?

How much can we change / modify ?

Initially we thought that it depends upon the existing size of the APK Signing Blocks, and the type and number of other blocks present ?

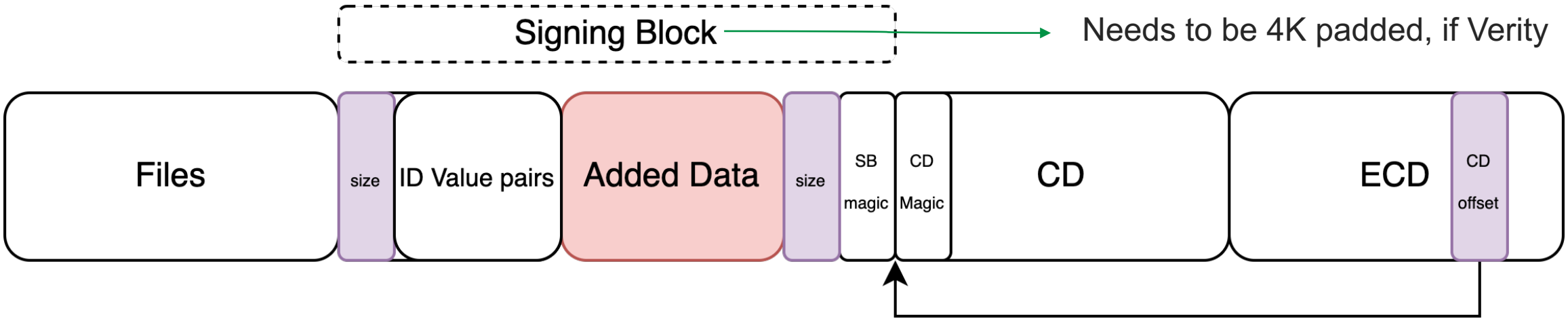
Protection of section 4 (ZIP End of Central Directory) is complicated by the section containing the offset of ZIP Central Directory. The offset changes when the size of the APK Signing Block changes, for instance, when a new signature is added. Thus, when computing digest over the ZIP End of Central Directory, the field containing the offset of ZIP Central Directory must be treated as containing the offset of the APK Signing Block.

```
// For the purposes of verifying integrity, ZIP End of Central Directory (EoCD) must be
// treated as though its Central Directory offset points to the start of APK Signing Block.
// We thus modify the EoCD accordingly.
ByteBuffer modifiedEocd = ByteBuffer.allocate(eocd.remaining());
int eocdSavedPos = eocd.position();
modifiedEocd.order(ByteOrder.LITTLE_ENDIAN);
modifiedEocd.put(eocd);
modifiedEocd.flip();

// restore eocd to position prior to modification in case it is to be used elsewhere
eocd.position(eocdSavedPos);
ZipUtils.setZipEocdCentralDirectoryOffset(modifiedEocd, beforeApkSigningBlock.size());
```

How much can we change / modify ?

Initially we thought that it depends upon the existing size of the APK Signing Blocks, and the type and number of other blocks present ?



KEYSIGHT

Use Cases

1. Good Use Case – Version Tracking ; Metadata in APK.
2. Malware Evasions – Embedding good in the Bad.
3. Covert Communications – Embedding Bad in the Good.
4. Code BOMBS - Embedding and reading from other apps

Use Case 0: Good

1. Version Tracking
2. Frosting
3. Custom Use Cases ?

Reproducible signatures

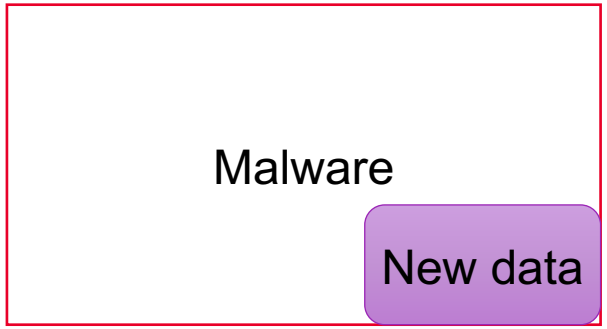
F-Droid verifies reproducible builds using the APK signature (a form of embedded signature), which requires copying the signature from a signed APK to an unsigned one and then checking if the latter verifies. The old v1 (JAR) signatures only cover the *contents* of the APK (e.g. ZIP metadata and ordering are irrelevant), but v2/v3 signatures cover *all other bytes in the APK*. Thus, the APKs must be completely identical *before* and *after* signing (apart from the signature) in order to verify correctly.

Copying the signature uses the same algorithm that `apksigner` uses when signing an APK. It is therefore important that (upstream) developers do the same when signing APKs, ideally by using `apksigner`.

Use Case 1: Malware Evasions without Repackaging

Steps:

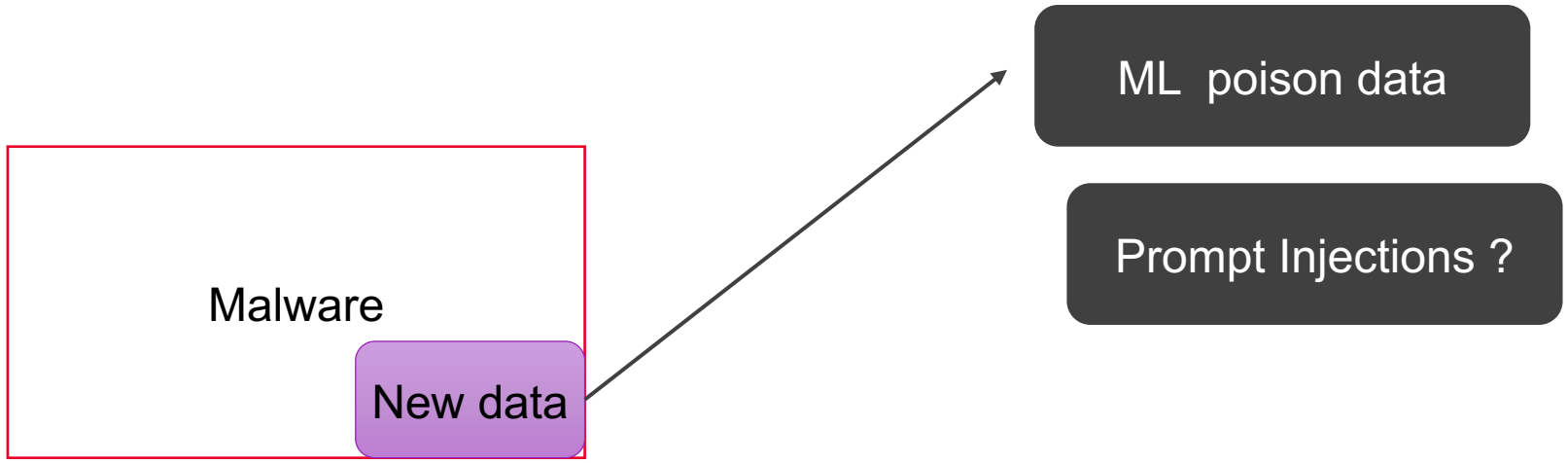
1. Take A Malicious APK.
2. Morph it by adding / changing the APK Signing blocks.
3. The detection rates goes down.



Use Case 1: Malware Evasions without Repackaging

Steps:

1. Take A Malicious APK.
2. Morph it by adding / changing the APK Signing blocks.
3. The detection rates goes down.



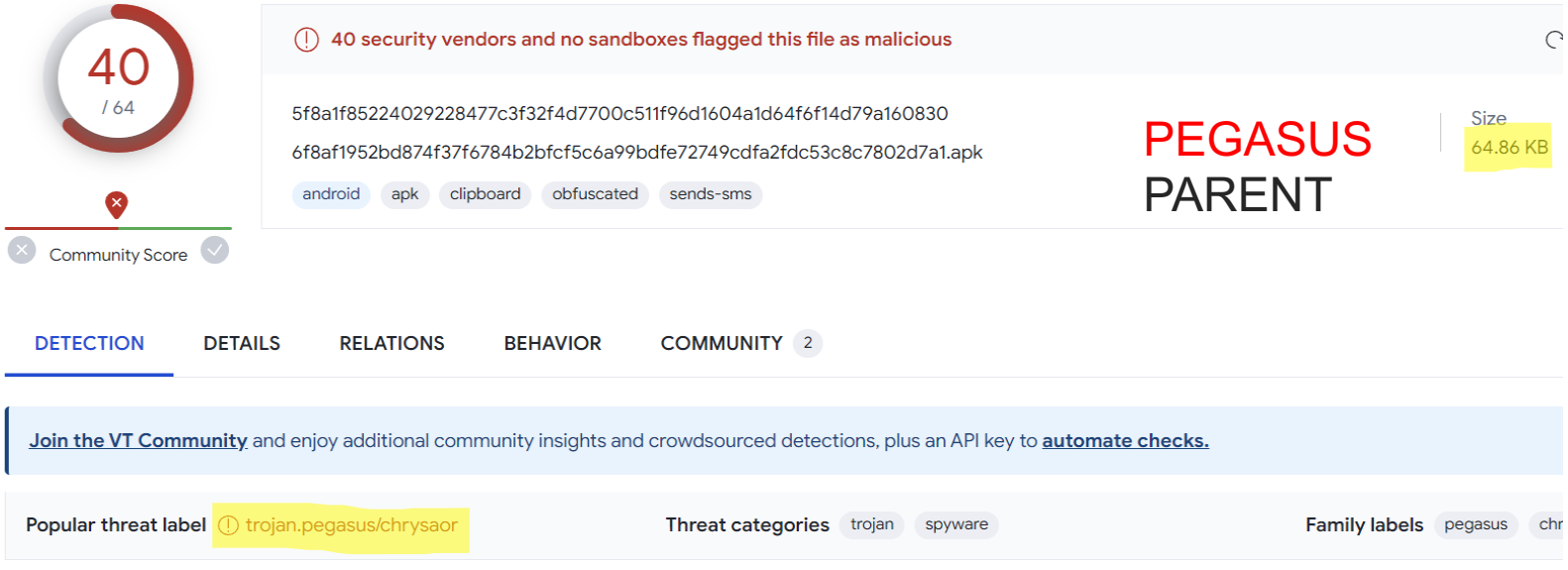
Use Case 1: Malware Evasions without Repackaging

Steps:

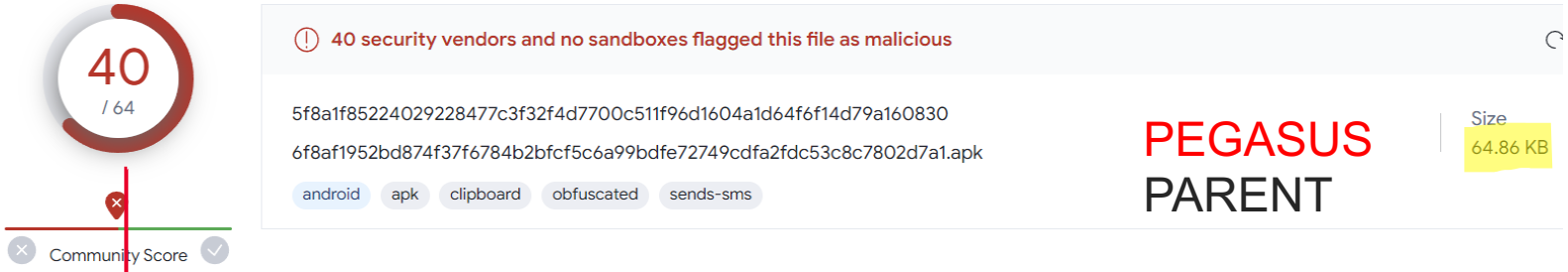
1. Take A Malicious APK.
2. Morph it by adding / changing the APK Signing blocks.
3. The detection rates goes down.

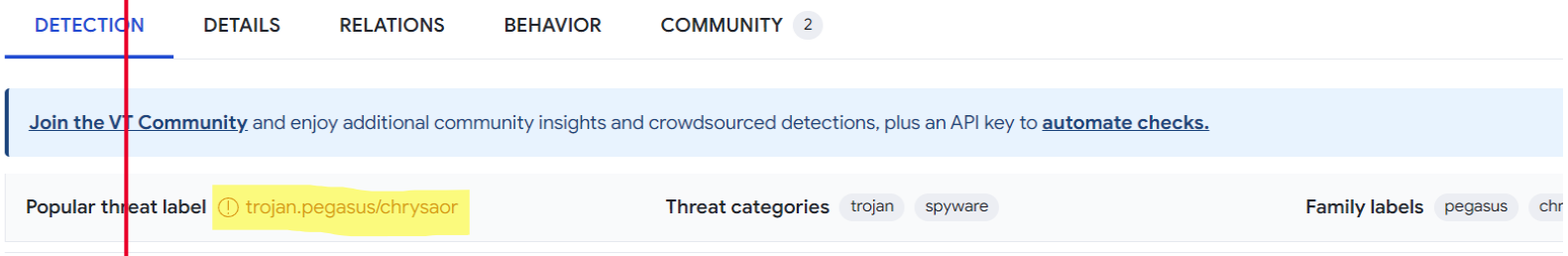
Mostly the hash-based lookup fails, since the hash of the file has changed!

Use Case 1: Malware Evasions without Repackaging

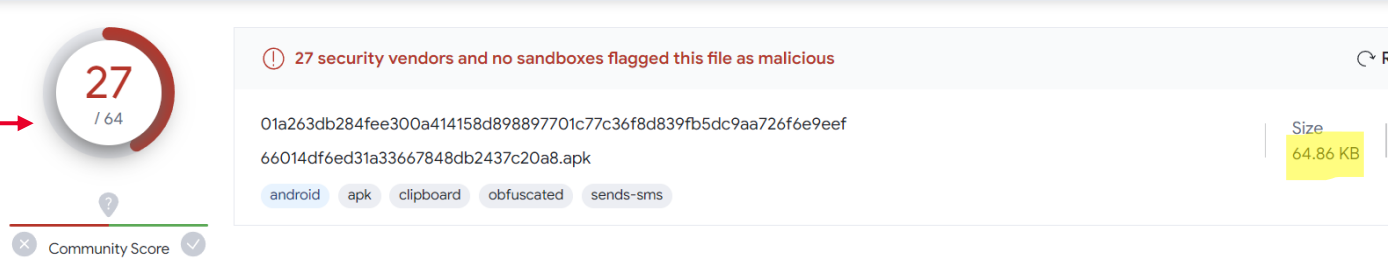


Use Case 1: Malware Evasions without Repackaging

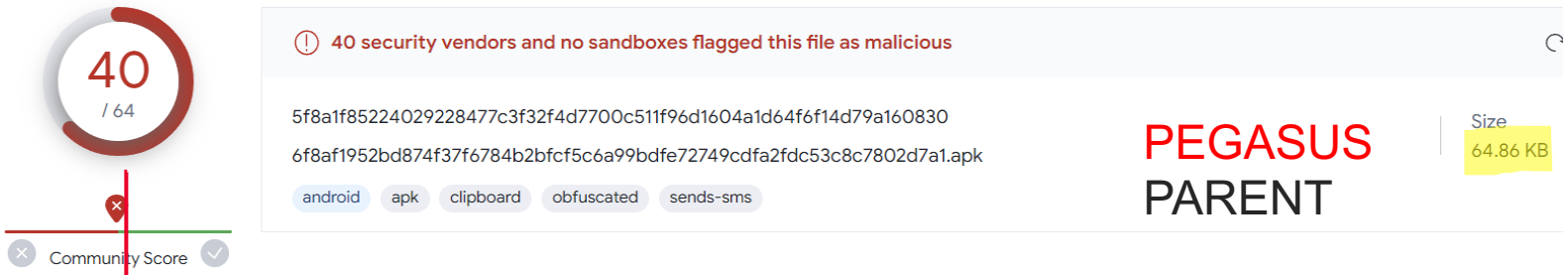


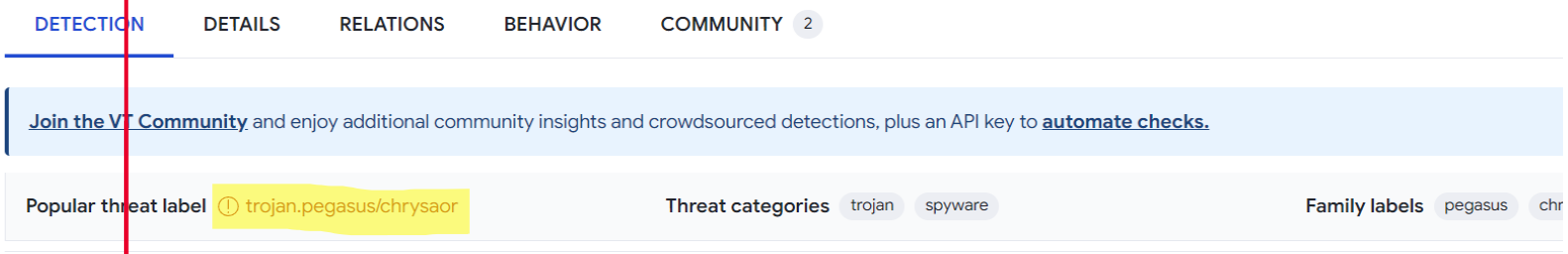


VERITY
MODIFIED

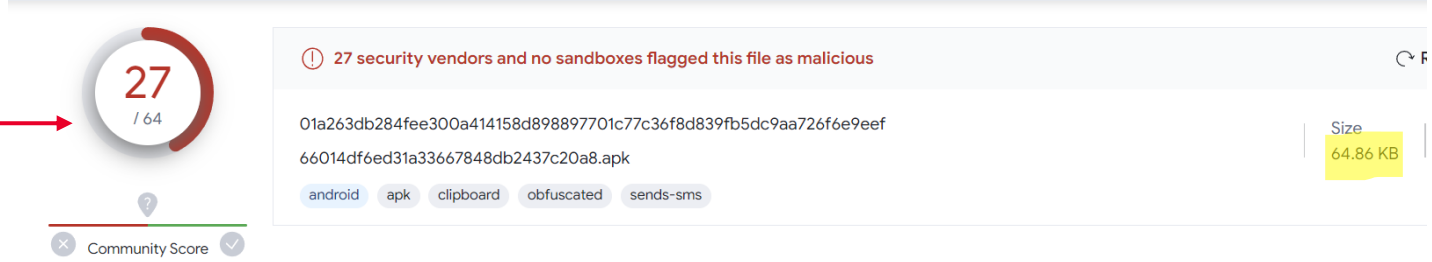


Use Case 1: Malware Evasions without Repackaging

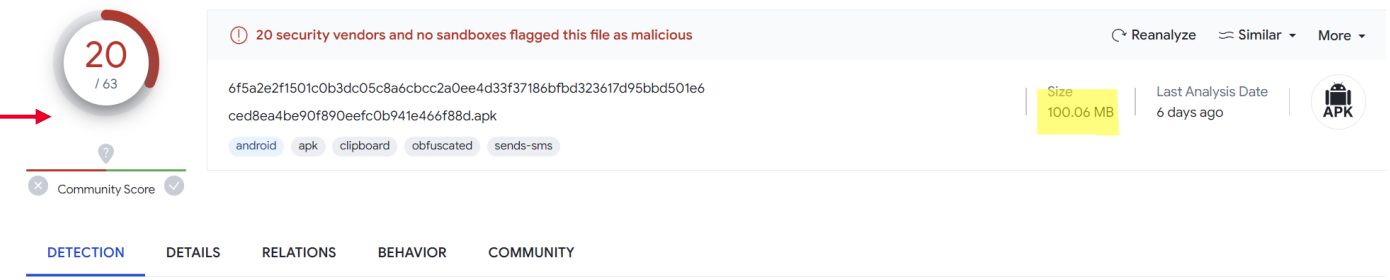




VERITY
MODIFIED



BLOATED
APK



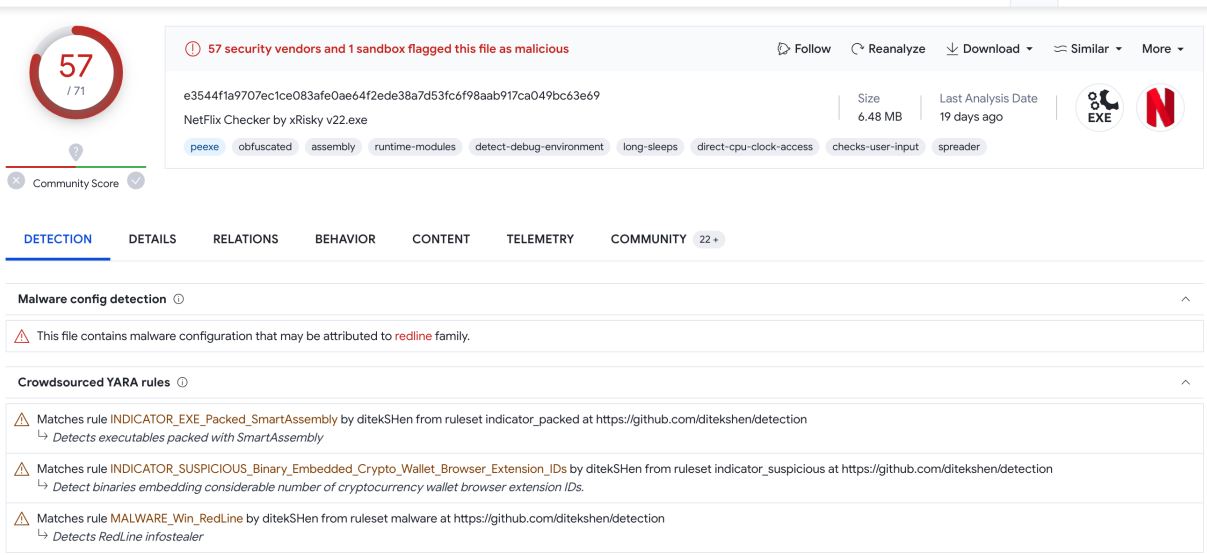
KEYSIGHT

Use Case 2: Embedding Malicious content inside APK

Steps:

1. Take A Good Signed and Verified APK.
2. Morph it by adding / changing the APK Signing blocks.
3. Added malicious content is not scanned ☺

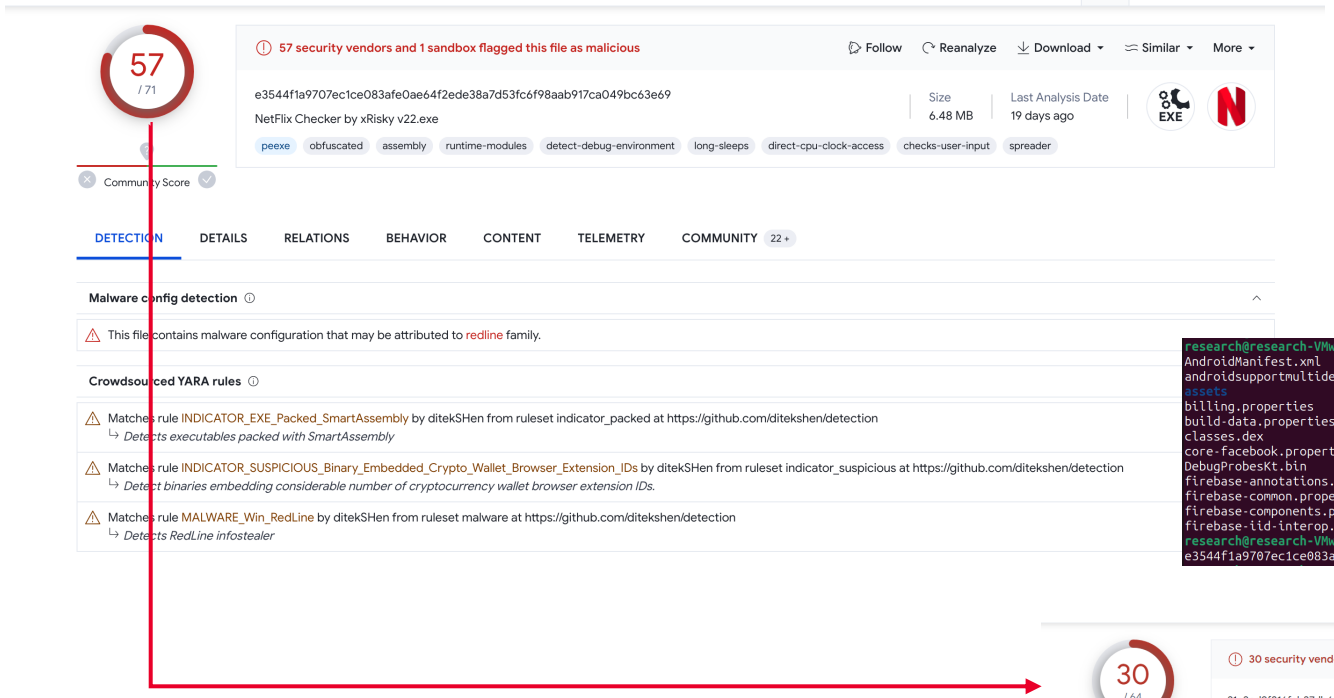
Use Case 2: Embedding Malicious content inside APK



Original Malicious Content

RedLine Stealer

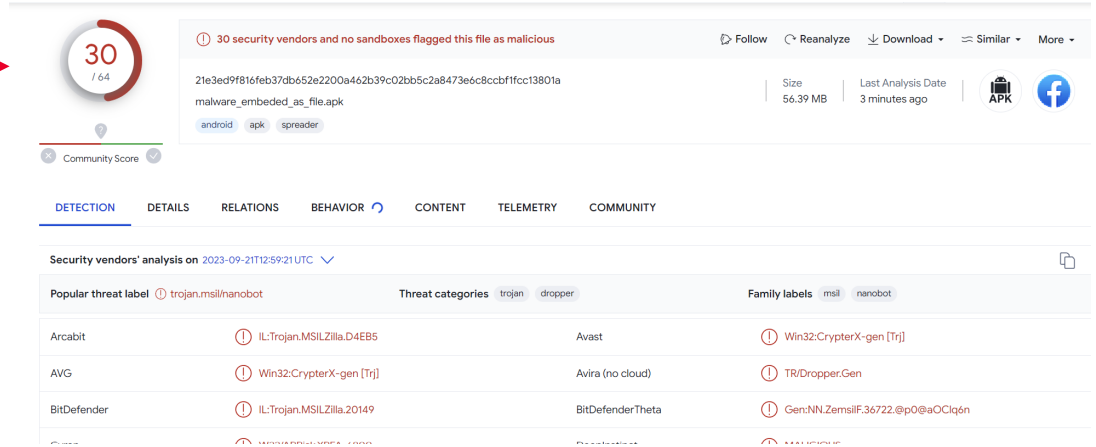
Use Case 2: Embedding Malicious content inside APK



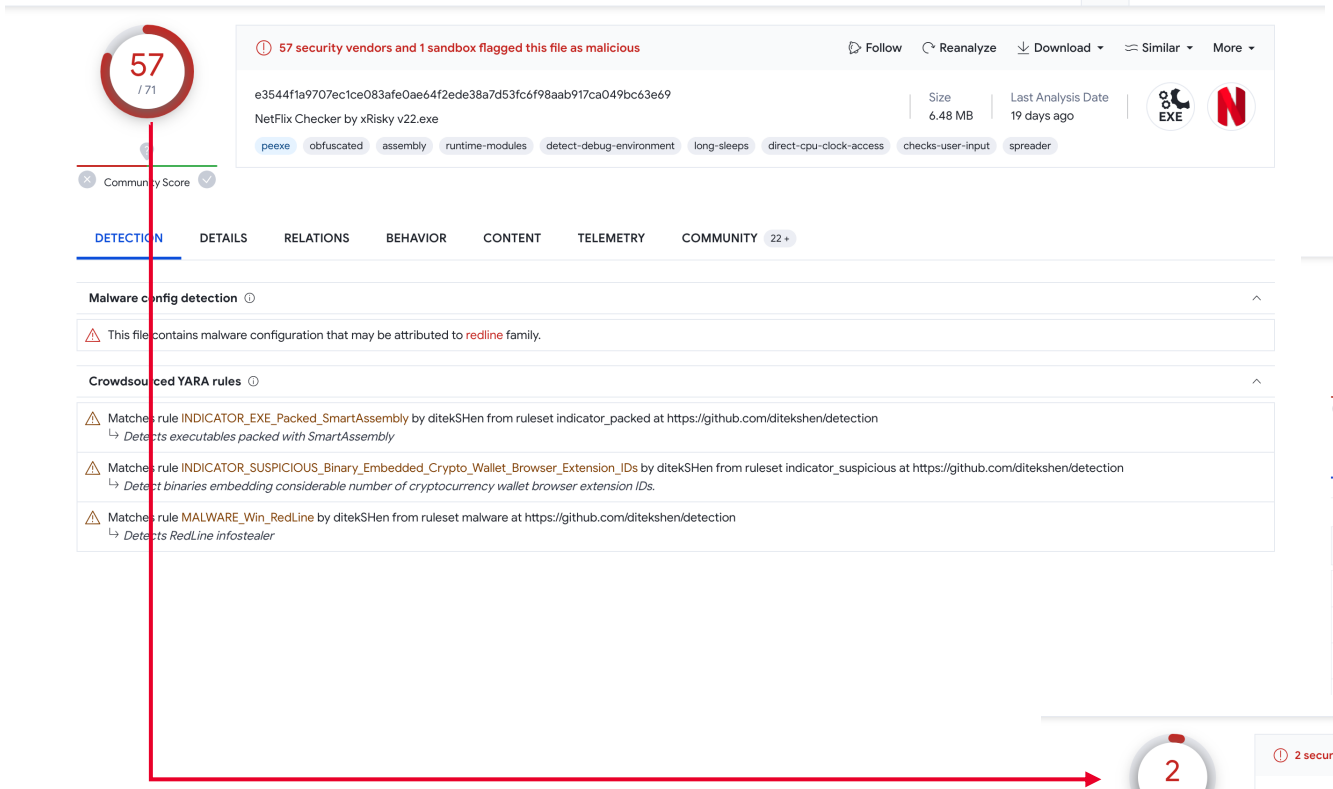
Original Malicious Content

```
research@research-VMware-Virtual-Platform:/tmp/test$ ls
AndroidManifest.xml                 firebase-iid.properties                 play-services-auth-api-phone.properties   play-services-places-placereport.properties
androidsupportmultidexversion.txt   firebase-measurement-connector.properties play-services-auth-base.properties       play-services-places.properties
assets                              firebase-messaging.properties           play-services-auth-blockstore.properties  play-services-safetynet.properties
billing.properties                  integrity.properties                    play-services-auth.properties             play-services-stats.properties
build-data.properties               kotlin                                  play-services-basement.properties         play-services-tagmanager-v4-impl.properties
classes.dex                         lib                                     play-services-base.properties             play-services-tasks.properties
core-facebook.properties            META-INF                                play-services-cast.properties             play-services-wearable.properties
DebugProbesKt.bin                   NOTICE                                  play-services-fido.properties             r
firebase-annotations.properties     org                                     play-services-flags.properties            resources.arsc
firebase-common.properties          play-services-ads-identifier.properties play-services-instantapps.properties
firebase-components.properties      play-services-analytics-impl.properties play-services-location.properties
firebase-iid-interop.properties     play-services-analytics.properties      play-services-maps.properties
research@research-VMware-Virtual-Platform:/tmp/test$ sha256sum r/normal_file
e3544f1a9707ec1ce083afe0ae64f2ede38a7d53fc6f98aab917ca049bc63e69  r/normal_file
```

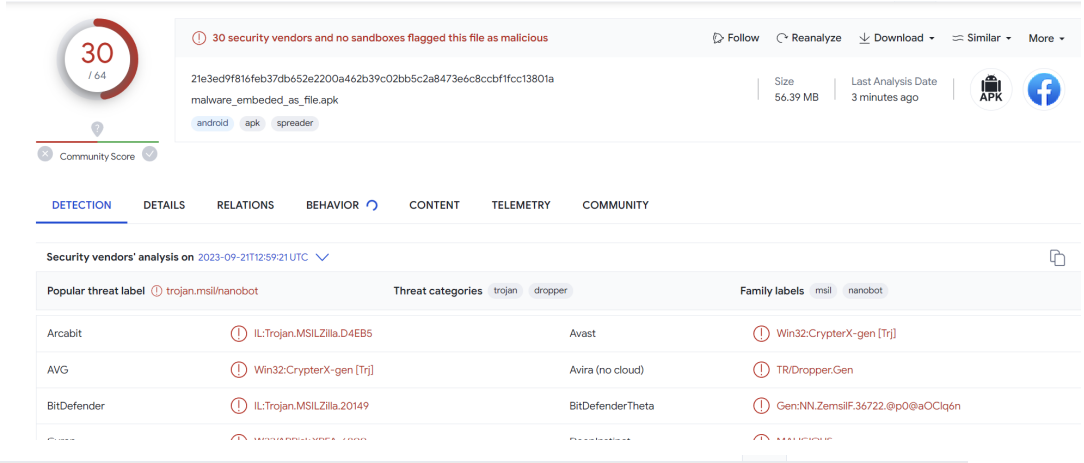
Same Content embedded inside APK Resources



Use Case 2: Embedding Malicious content inside APK



Original Malicious Content



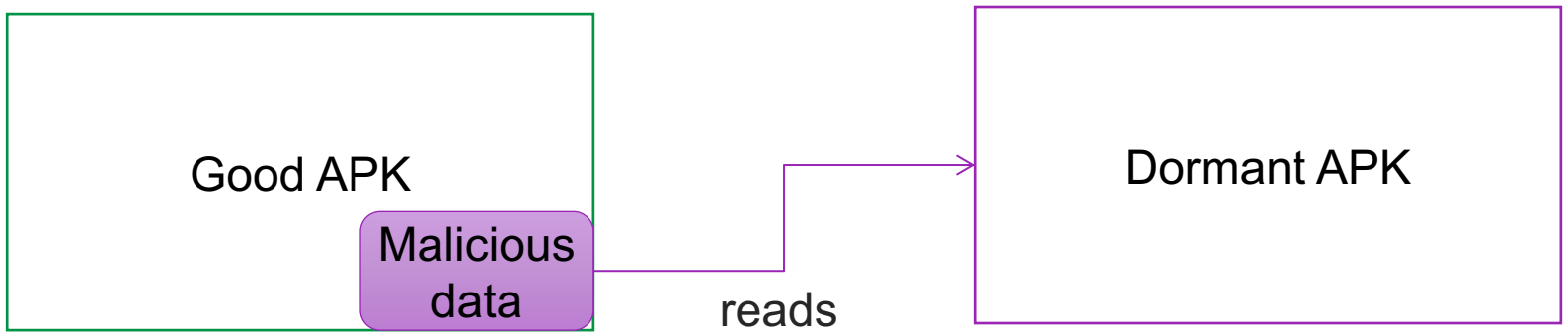
Same Binary Blob embedded inside APK Signing Block

Use Case 3 : Embedding Code Bombs inside APK

Steps:

1. Take A Good Signed and Verified APK.
2. Add messages / content as part of the Signing blocks.
3. Added content is not scanned ☺

4. It stays on the device forever, waiting to be ignited by any other apps.



Use Case 3 : Embedding Code Bombs inside APK

- In all android versions, one App can read another APK file
- But getting the path to the APK file is a bit tricky
- It can be fetched by running pm path package_name but only can be done till Android 10

```
generic_x86_64:/ # ls -la /data/app
total 32
drwxrwx--x  4 system system 4096 2023-09-22 16:44 .
drwxrwx--x 45 system system 4096 2023-09-21 16:58 ..
drwxrwxr-x  3 system system 4096 2023-09-22 16:44 com.example.nullcondemo-yFPtt-rwQZMhiHgBbcwoqQ==
drwxrwxr-x  4 system system 4096 2023-09-22 16:44 com.facebook.lite-BnQgmPKW25n-idpsGllqFQ==
```

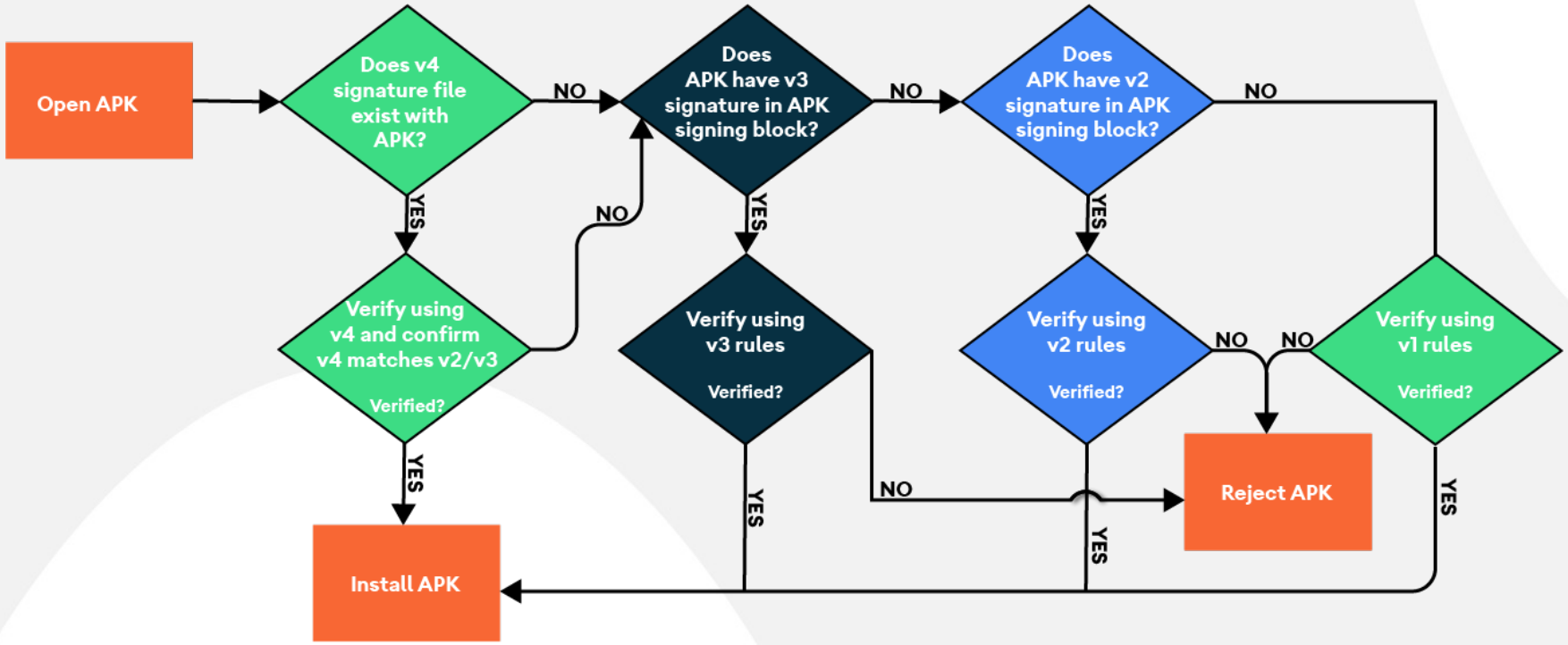
Demo 2

Code BOMB detonations

KEYSIGHT

APK Signature Scheme v4

Android 11 supports a streaming-compatible signing scheme with the APK Signature Scheme v4. The v4 signature is based on the Merkle hash tree calculated over all bytes of the APK. It follows the structure of the fs-verity hash tree exactly (for example, zero-padding the salt and zero-padding the last block). Android 11 stores the signature in a separate file, `<apk name>.apk.idsig` A v4 signature requires a complementary v2 or v3 signature.



KEYSIGHT

Can we Fuzz the APK Signing Block ?


```
## Generate a keystore for signing
echo y | keytool -genkeypair -dname "cn=Android, ou=Android, o=Android, c=Android" -alias Android -keypass Android -keystore /tmp/Android.keyst

## Sign the APK which has an existing signature and use --append-signature to preserve it
apksigner sign -ks /tmp/Android.keystore --append-signature apk_v2_block_size_2147400000.apk
```

```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at com.android.apksig.internal.apk.ApkSigningBlockUtils.getApkSignatureBlocks(ApkSigningBlockUtil
        at com.android.apksig.DefaultApkSignerEngine.inputApkSigningBlock(DefaultApkSignerEngine.java:654
        at com.android.apksig.ApkSigner.sign(ApkSigner.java:337)
        at com.android.apksig.ApkSigner.sign(ApkSigner.java:228)
        at com.android.apksigner.ApkSignerTool.sign(ApkSignerTool.java:400)
        at com.android.apksigner.ApkSignerTool.main(ApkSignerTool.java:92)
```

Java Heap Memory exhaustion error in the apksigner - DOS

Caused as apksigner assigns array memory without any checks, ex : new byte[(int) cdRecord.getUncompressedSize()]

```
apksigner verify -v apk_sample_manifestsize_2147400000.apk
```

```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at com.android.apksig.internal.zip.LocalFileRecord.getUncompressedData(LocalFileRecord.java:449)
        at com.android.apksig.ApkSigner.getAndroidManifestFromApk(ApkSigner.java:965)
        at com.android.apksig.ApkVerifier.getAndroidManifestFromApk(ApkVerifier.java:1046)
        at com.android.apksig.ApkVerifier.verifyAndGetMinSdkVersion(ApkVerifier.java:636)
        at com.android.apksig.ApkVerifier.verify(ApkVerifier.java:184)
        at com.android.apksig.ApkVerifier.verify(ApkVerifier.java:151)
        at com.android.apksigner.ApkSignerTool.verify(ApkSignerTool.java:570)
        at com.android.apksigner.ApkSignerTool.main(ApkSignerTool.java:95)
```

Take-A-Ways Attackers

APK signing blocks can be used to embedded stuffs to change the binary without impacting the signed or the signing schemes.

Use this to either embed malicious stuffs or c2 communications.

Take-A-Ways Attackers

APK signing blocks can be used to embedded stuffs to change the binary without impacting the signed or the signing schemes.

Use this to either embed malicious stuffs or c2 communications.

Take-A-Ways Defenders

Start scanning for these APK Signing blocks irrespective of

- the APK Verification status

- or Repackaging Detections.

KEYSIGHT

Thank you

Reach out:

Ayan – saha.ayan1996 [at] gmail.com

Achute – achute.sharma [at] keysight.com