List Of Images

Summary of Findings

Static Analysis:

- Used JADX to analyze XML files for permissions and activities such as analyzing the code
- Few apps had potential security issues like weaker encryption algorithms for hash function (MD5, SHA-1)
- Verifies APK signature versions (v1,v2,v3) to confirm app authenticity and integrity
- Most apps did not show signs of insecurity
- Analyzed some apps and found strong encryption such as SHA-256 and RSA
- DSA (digital signature algorithm) was also found, which is a combination of SHA-256 and RSA
- Found zip bomb attacks on some apps
- Analyzed permissions from “android.permission.DUMP” for potential malicious gathering of data
- Handshake methods were found in some apps, though these were not cryptographic handshakes, but rather a metaphorical handshake between the app and an advertisement service

Dynamic Analysis:

- Used Burp Suite to analyze HTTP traffic for each app
- Examined network requests and data transmissions for any user-sensitive information that may be transmitted
- Few apps didn’t send any HTTP requests other than advertising-related requests indicating no risk to the user
- The majority of apps did not transmit sensitive user data such as location or account information
- Some apps send non-compromising device information (OS, time zone, app version)

Apps Analysis Outcome:

- Many apps had similar outcomes, such as similar encryption methods and similar HTTP requests

Summary of Methodologies and Tools Used

Confidentiality and integrity are the two most fundamental yet alarming aspects of cybersecurity. Without these concepts being deeply understood, adversaries can intuitively take advantage and attack the unsecured channel or even the application itself, resulting in catastrophes such as privacy information leakage, and modification of the authentic data. As such, security measurement and analysis need to be carried out carefully so the prevention of these kinds of attacks can be inaugurated with absolute precision
The goal of this project is to perform an analysis of the security and privacy aspects of 30 Android applications, mainly game applications starting with “Woodoku”. The upcoming apps will be downloaded based on the advertisements displayed within each previously downloaded app, which creates a chain of interconnected apps. The project aims to do a static and dynamic analysis of these applications. Each app will examine its privacy, assess security measures that were implemented in each application, and look at the behavioral changes such as its malicious activities. These approaches will help us get a better understanding of how security and privacy are featured in these applications.

Relating to the above-mentioned problem, this project will demonstrate in detail the process of security and privacy analysis for a set of 30 Android apps. The details of how we download and do the analysis will be provided with more details in the upcoming sections

Initially, the apps were selected through an advertisement chain. The first app, Woodoku, was downloaded on a physical Android device. This was due to the Android Studio emulator not supporting advertisements, only test advertisements would be displayed on an app running in the emulator. Once an advertisement appeared, the app was downloaded as long as it contained advertisements, which was indicated in the Google Play store page. If the app was found to not contain advertisements, then the next advertisement that appeared would be used instead. Once the 30 apps were downloaded, a list was compiled and the corresponding APK’s were downloaded from the APKPure website.

For Static Analysis, JADX was the primary tool used. The APK was loaded into JADX and the code was able to be analyzed. This was used to analyze every app’s XML files to check what method of encryption was used, what permissions the app requires and the APK signatures. The encryption methods used can provide information on how secure data transfers are over the internet for a given application. APK signatures can provide useful information on the legitimacy of an app and ensure that it has not been tampered with and contains no malicious code. The permission dump was also analyzed to view the permissions required for a given app. Apps that require storage or location permissions can be concerning since these permissions can access user sensitive information. Furthermore, since most of the apps we are analyzing are games they should not require this information.

For Dynamic Analysis, Android Studio was used to emulate an API 33 Android device. Android Studio was used due to its familiarity for the group. ADB (Android Debug Bridge) was used to sideload the apps being analyzed. The apps were then opened and used normally. Burp Suite was used to analyze the HTTP traffic sent and received while the apps were running. To accomplish this a proxy had to be set up within

the Android Studio emulator to capture the traffic in Burp Suite. Any requests sent from the application were analyzed for user sensitive information, such as device location or personal information.

Results and Discussion

Static Analysis

Zip Bomb Attack

- Found in the following apps: Wooduku, 2248, WSOP Poker: Texas Holdem Game, Get Color-Water Sort Puzzle, Solitaire (see Figure 1).

The error from Figure 1 happened right after we opened the static code of the application. This error indicates a zip bomb file has been detected. The detail for this zip bomb attack analysis will be provided as follow:

- First, we need to know what a zip bomb attack is. In short, it is a way of attacking by attaching a malicious compressed file. This file, when decompressed, will contain a limitless amount of data that will overwhelm the OS and file system that is reading it
- One way to analyze this kind of attack is to run static analysis without actually decompressing the file.
- "Zip bomb attack detected, invalid sizes: compressed 1856, uncompressed 279848": This shows that this zip file when compressed only has 1856 bytes but 279848 bytes when decompressed. As we can see, the ratio between when compressed and decompressed is very high. This indicates that if the static analysis had not been performed, the device's OS and file system would have been completely rendered, and flooded with excessive and repetitive data
- The consequences of this kind of attack might lay one or more of the following repercussions
    - CPU and memory overload: This happens when the system tries to decompress the file. To do so, the OS will have to distribute a lot of hardware resources. As a result, the performance of the system will degrade (lagging, freezing, slowdowns) even leading to crashing
    - Denial of Service (DoS): if a zip file is enormous when decompressing, it will consume all network bandwidth and flood the process with massive amounts of binary data. As a result, a bottleneck will be inaugurated and network service might not be available for legitimate users
    - Downtime and Loss of Productivity: If the device is infected with this kind of attack, and to clean up, will be very time and resources consuming. And there is no guarantee that the system or OS will be fully recovered from this. Hence, there will be degradation for the system, in the long run,
    - Data Loss and Corruption: When decompressing the zip bomb file, since the size is very large, it can overload the storage capacity. Hence, it might be able to corrupt the information of the existing file, leading to potential data loss or irreparable damage to critical information.

Signature type

SHA256 with RSA (see Figure 2):

- Found in the following apps: Woodoku, Galaxiaga, Color Blast, WOW, Sudoku.com, Wood Block Puzzle, Block Blast, Bricks n Balls, Get Color - Water Sort Puzzle, Block Ocean Puzzle 1010, Tower War - Tactical Conquest, Stuff Sort - Sorting Master, Playdoku: Puzzle Block Game, Block Match - Wood Puzzle.

The reliability of SHA256withRSA for digital signature (certificate) signing is widely acknowledged. This algorithm, as implied by its name, incorporates the SHA256 hash function and RSA encryption algorithm to execute the signing process using a pre-existing private key (not generated by the algorithm). The ensuing elucidation provides a comprehensive insight into the robustness of this algorithm.

- SHA-256: This belongs to the SHA-2 (Secure Hash Algorithm 2) suite of cryptographic hash functions, renowned for their robust security characteristics. Generating a 256-bit hash value, SHA-256 ensures data uniqueness and stands resilient against collisions, significantly minimizing the probability of two distinct inputs yielding an identical hash output.
- RSA: this stands as a pivotal asymmetric cryptosystem (works on public and private keys) employed to fortify data transmission. Leveraging a private key for digital signature authentication (certificate), this system allows anyone possessing the corresponding public key to verify the data, thereby ensuring its integrity. Consequently, RSA serves as a cornerstone in verifying data integrity and confidentiality, attesting to the authenticity of the signer's data.
- Long-term security: Utilizing this algorithm with a sufficiently robust key entropy, specifically 2048 bits or higher, is recognized as an effective measure to establish formidable security against classical cryptanalysis methods. Notably, it provides resilience against known-plaintext, chosen-plaintext, and ciphertext-only attacks, bolstering the encryption's resistance to unauthorized decryption attempts.

SHA1 with RSA (see Figure 3):

- Found in the following apps: 1945 Airforce, Drop the Number, Wordscape, Search, 2248, Solitaire, Pro des mots.

The use of SHA1 with RSA is no longer considered secure due to its susceptibility to collision attacks, where distinct inputs yield the same hash value. This cryptographic algorithm combines SHA1 as the hash function and RSA as the encryption algorithm for digital signing.

- SHA1: SHA1's fixed-length 160-bit hash output, irrespective of input size, poses significant vulnerabilities to collision attacks. Given sufficient time, attackers could feasibly find different inputs generating an identical hash value, compromising the hash function's integrity. Consequently, this compromise undermines application trustworthiness, enabling the potential substitution of inputs undetected, ultimately leading to security vulnerabilities.
- RSA: Already explained in the above section (section includes Figure 2)

SHA1 with DSA (see Figure 4):

- Found in the following apps: Tetris, WSOP Poker: Texas Holdem Game.

The use of SHA1withDSA is no longer considered secure due to its susceptibility to collision attacks, where distinct inputs yield the same hash value. This cryptographic algorithm combines SHA1 as the hash function and DSA as the encryption algorithm for digital signing.

- SHA1: Already explained in the above section (section includes Figure 3)
- DSA: Digital Signature Algorithm (DSA) involves the generation of a private-public key pair, where the private key is employed for signing digital certificates (signatures). Derived from the private key, the public key serves the purpose of verifying these signatures. Similar to RSA, DSA allows anyone possessing the corresponding public key to validate data, thereby ensuring its integrity. Overall, DSA serves to guarantee message integrity, sender authenticity, and confidentiality by facilitating the creation and verification of digital signatures without divulging the private key.

*Note: Compared with RSA, DSA is faster at decrypting and signing; yet, slower at encrypting and verifying

MD5 with RSA (see Figure 5):

- Found in the following app: The Battle Cats

The security of MD5withRSA has been compromised due to its vulnerability to collision attacks, resulting in identical hash values for different inputs. This cryptographic scheme employs MD5 as the hash function and DSA as the encryption algorithm for digital signatures.

- The MD5 (Message Digest Algorithm 5) generates a consistent 128-bit hash output, regardless of input size, which presents notable susceptibility to collision attacks. Given ample time, adversaries could potentially discover diverse inputs producing an identical hash value, thereby compromising the integrity of the hash function. Consequently, this compromise erodes the reliability of applications, allowing potential undetected substitution of inputs and consequently leading to critical security vulnerabilities.
- RSA: Already explained in the above section (section includes Figure 2)

.xml for Android permission

Dump methods (see Figure 6):

- Found in the following apps: Color blast, Drop the Number, WOW, Wordscape, Search, Solitaire, The Battle Cats, Pro des mots, Sudoku.com, Wood Block Puzzle, Block Blast, Bricks n Balls, Get Color- Water Sort Puzzle, Block Ocean Puzzle 1010, Tower War- Tactical Conquest, Block Match- Wood Puzzle.

The highlighted section within the depicted figure denotes the authorization granted to the application for acquiring state dump information from system services, facilitating the debugging procedures. The presence of "android.work.impl.diagnostic.DiagnosticsReceiver" as a system receiver necessitates this permission for its proper functionality. As this authorization enables access to sensitive system information through state dump storage, it warrants careful scrutiny and thorough assessment when employed. There exists a potential risk for misuse by malicious applications or botnets to gather sensitive system data, underscoring the imperative need for diligent examination and cautious utilization of this permission.

Handshake methods (see Figure 7):

- Found in the following app: Tetris.

The term "handshake," as highlighted in the figure, denotes the process of reporting ad placements between the application and the advertisement service. In this context, it signifies a data exchange mechanism between the two entities. For this exchange to occur, the app requires the "android.permission.BIND_JOB_SERVICE," granting access to the JobScheduler service, enabling background task scheduling within the system. Utilizing this permission, the advertisement service can efficiently schedule tasks for ad retrieval, caching, and content updates, effectively managing ad-related processes without impeding the application's performance.

However, if this permission falls into the hands of malicious entities, it poses potential vulnerabilities:

1. Task Manipulation: Adversaries could tamper with and disrupt the app's scheduled tasks, causing a degradation in application performance.

2. Resource Exhaustion: Deliberate scheduling of numerous resource-intensive tasks by attackers could result in resource depletion, causing performance slowdowns, system instability, or even system-wide crashes due to inadequate resource allocation.

These vulnerabilities highlight the importance of safeguarding permissions like "android.permission.BIND_JOB_SERVICE" to prevent unauthorized access or misuse, ensuring the integrity of background task scheduling and overall system performance.

APK Signatures (see Figure 8)

` `

v1 and v2:

- Found in the following apps: 1945 Airforce, Drop the Number, Search, 2248

v2 and v3:

- Found in the following appsWOW, Wordscape, Tower War - Tactical Conquest

Other apps have v1, v2, v3.
No warnings

- Found in the following apps: WOW, Wordscape, Tower War - Tactical Conquest, Stuff Sort - Sorting Master, Playdoku: Puzzle Block Game

Warnings (see Figure 9):

- Found in the following apps: woodoku, Galaxiaga, 1945 Airforce, Color Blast, Drop the Number, Wordscape, 2248, Solitaire, WSOP Poker: Texas Holdem Game, The Battle Cats, Pros des mots, Sudoku.com, Wood Block Puzzle, Block Blast, Bricks n Balls, Get Color - Water Sort Puzzle, Block Ocean Puzzle 1010, Block Match - Wood Puzzle

"Files that are not protected by APK signature v1. Unauthorized modifications to these entries can only be deleted by APK signature v2 and higher."

Having a signature in an APK is important in Android Studio similarly, on how we have a signature and it shows our identity. The signature verifies the identity of the developer and makes sure that if there are any updates on the app, it is provided by the correct developer. Signature prevents malicious actions. On the figure above, it shows that the signature in V1 (version 1) is not secure enough to protect the APK files, however in the signature v2 provides a better security. If an app with only a signature with v1 was installed, an attacker may take advantage of this and download an older version of the app containing more security issues therefore putting the phone at risk.

Dynamic Analysis

A portion of the apps sent HTTP requests containing device information. These requests contained information such as the device model, the OS version, the time zone and other device information. None of this information is particularly sensitive. Time zones can give a general idea of a user's location but are generally too broad to be considered sensitive. Other HTTP requests from these apps only contained information related to serving advertisements. Apps that transmitted this kind of information over HTTP were Woodoku, Galaxiga, 1945 Airforce, Tetris, Drop the Number, WOW, Wordscape, Search, WSOP, The Battle Cats, Sudoku.com, Block Ocean Puzzle 1010. See Figure 10 in the appendix for an example of one of these HTTP requests. All of these apps sent similar information as described above. The exact format of these may be slightly different but all these apps sent the same information. Most apps sent this information directly in the HTTP request in hex format, while some encoded it in JSON format, like Drop the Number. There are no inherent security advantages to using a JSON format so this does not raise any concerns.

There was one app that sent significantly more information than the others, 2248. Most of the information is device information, though it does contain fields for user information. These user information fields are populated solely as "Guest" since accounts are not required to play the game. If accounts were required, this information being transmitted would raise security concerns since this information is personal. The app also sends the device's country that it's located in. While the country alone is not an exact location of the user, the user is not informed that this information is being sent to a server. Other device information sent

includes OS version and app version which is fairly standard amongst all the other apps. See Figure 11 for the HTTP request containing this information.

There was also a smaller group of apps that did not send any HTTP requests containing device or user information. Any HTTP requests sent during runtime were advertisement-related only. These apps include Pro des Mots, Block Blast, Get Color, Farm Jam, Card Shuffle Sort, Tower War, Stuff Sort and Block Match. Playdoku also behaved similarly to this but required a login step. Logging in can be completed as a guest and will send a related HTTP request. Following that it will only send advertising-related requests.

From dynamic analysis, we can determine that the vast majority of these applications are safe to use. Most do not send any user-sensitive information and mostly send device and application version-related information. There were some concerning applications such as 2248, which if real user accounts could send sensitive information through HTTP.

References

1. , John V. “DSA vs. RSA Encryption - Which Works Best for File Transfers? | JSCAPE.” *jscape*, 9 November 2022, https://www.jscape.com/blog/which-works-best-for-encrypted-file-transfers-rsa-or-dsa. Accessed 6 December 2023.
2. “android - Apk Metainfo Warning.” *Stack Overflow*, 31 August 2018, https://stackoverflow.com/questions/52122546/apk-metainfo-warning. Accessed 6 December 2023.
3. “Android Permissions.” *Android Permissions*, http://androidpermissions.com/permission/android.permission.DUMP. Accessed 6 December 2023.
4. Cobb, Michael. “What is the RSA algorithm? Definition from SearchSecurity.” *TechTarget*, https://www.techtarget.com/searchsecurity/definition/RSA. Accessed 6 December 2023.
5. Constantin, Lucian. “The SHA1 hash function is now completely unsafe.” *Computerworld*, 23 February 2017, https://www.computerworld.com/article/3173616/the-sha1-hash-function-is-now-completely-unsafe.html. Accessed 6 December 2023.
6. Guide, Step. “MD5 Hash Algorithm in Cryptography: Here's Everything You Should Know.” *Simplilearn.com*, 20 October 2023, https://www.simplilearn.com/tutorials/cyber-security-tutorial/md5-algorithm. Accessed 6 December 2023.
7. Jain, Sandeep. “RSA Algorithm in Cryptography.” *GeeksforGeeks*, 9 November 2023, https://www.geeksforgeeks.org/rsa-algorithm-cryptography/. Accessed 6 December 2023.
8. Kumar, Baivab. “Digital Signature Algorithm (DSA) in Cryptography: A Complete Guide.” *Simplilearn.com*, 14 February 2023, https://www.simplilearn.com/tutorials/cryptography-tutorial/digital-signature-algorithm. Accessed 6 December 2023.

9. Kumar, Baivab. “What Is SHA-256 Algorithm: How it Works and Applications.” *Simplilearn.com*, 29 August 2023, https://www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256-algorithm. Accessed 6 December 2023.
10. “Manifest.Permission.Dump Field (Android).” *Microsoft Learn*, https://learn.microsoft.com/en-us/dotnet/api/android.manifest.permission.dump?view=xamarin-android-sdk-13. Accessed 6 December 2023.
11. Mehta, Janki. “SHA1 Vs. SHA256 - What's the Difference Between?” *Cheap SSL Certificates*, https://cheapsslweb.com/blog/sha1-vs-sha256. Accessed 6 December 2023.
12. “RSA Algorithm Vs. DSA: Explore the Difference Between RSA Algorithm and DSA.” *BYJU'S*, https://byjus.com/gate/difference-between-rsa-algorithm-and-dsa/. Accessed 6 December 2023.
13. Saha, Ayan. “Promiscuous Permissions: Catching Your Android Apps in the Act.” *Keysight*, 24 March 2023, https://www.keysight.com/blogs/tech/nwvs/2023/03/24/promiscuous-permissions-catching-your-android-apps-in-the-act. Accessed 6 December 2023.
14. “SHA-256 Cryptographic Hash Algorithm implemented in JavaScript | Movable Type Scripts.” *Movable-type.co.uk*, https://www.movable-type.co.uk/scripts/sha256.html. Accessed 6 December 2023.
15. “What is a Zip Bomb Attack?” *Mimecast*, https://www.mimecast.com/content/what-is-a-zip-bomb/. Accessed 6 December 2023.
16. “What Is MD5 and Why Is It Considered Insecure?” *Section.io*, 20 April 2020, https://www.section.io/engineering-education/what-is-md5/. Accessed 6 December 2023.
17. “What is MD5? Understanding Message-Digest Algorithms.” *Okta*, 26 July 2022, https://www.okta.com/identity-101/md5/. Accessed 6 December 2023.

Appendix

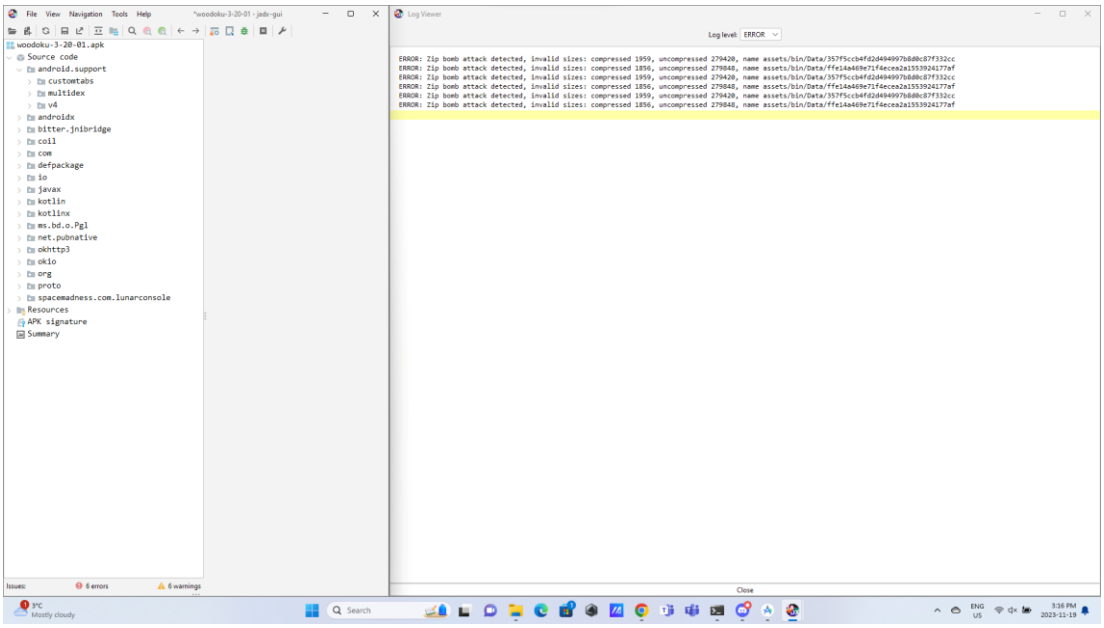

Figure 1. Zip Bomb Attack on JADX.

```
    Valid from: Thu Mar 12 08:59:06 EDT 2020
    Valid until: Sat Mar 12 07:59:06 EST 2050

    Public key type: RSA
    Exponent: 65537
    Modulus size (bits): 4096
    Modulus: 64524132045802561042920653347829822784665826946168542208657269544119410622925613329340255180122270351582604355 7

    Signature type: SHA256withRSA
    Signature OID: 1.2.840.113549.1.1.11

    MD5 Fingerprint: FF 17 58 8C 01 1D 57 2B BC C4 1E A5 E4 E5 99 C8
    SHA-1 Fingerprint: 3C EC 41 E4 34 44 26 59 3A 4B CE F9 1E 56 52 D5 3E 8D 1C BD
    SHA-256 Fingerprint: CE 57 F6 F3 30 28 FD B0 3C CA D4 CA D9 5B D7 89 10 8B AB 20 2D 64 E5 0A 11 3B 09 AF 31 F1 71 E1

id APK signature v2 found
```

Figure 2. Public Key SHA256withRSA

```
        Public key type: RSA
        Exponent: 65537
        Modulus size (bits): 2048
        Modulus: 19161384581596854589905030658962742380370728212307829282291340832792949129566553792160825149110071218854805890611651687314 1

        Signature type: SHA1withRSA
        Signature OID: 1.2.840.113549.1.1.5

        MD5 Fingerprint: 49 5B 28 71 61 6E DF 83 11 10 88 69 7D DF 92 4A
        SHA-1 Fingerprint: 1F AB B2 69 39 6D 18 E5 66 29 AE 6A 00 00 5C 88 5D 7B D7 33
        SHA-256 Fingerprint: 4B 5A 8E 19 4A 96 53 FB E0 5B 0C 6C 7B B8 C7 FA C1 02 43 C9 68 74 7E 5A 1E AA C1 23 4A 74 2D 3E

Valid APK signature v2 found
```

Figure 3. Public Key SHA1withRSA

```
Public key type: DSA
Y: 60254844055437424459095081615451522810995704835344702

Signature type: SHA1withDSA
Signature OID: 1.2.840.10040.4.3

MD5 Fingerprint: 42 02 49 12 3F 7B 40 F8 8E 30 98 BC B2
SHA-1 Fingerprint: 0D CF B0 2E C6 96 F8 C2 4E 96 76 D0 5
```

Figure 4. Public Key SHA1withDSA

```
Exponent: 65537
Modulus size (bits): 1024
Modulus: 116629700794149716122819947136756059051245541914555499

Signature type: MD5withRSA
Signature OID: 1.2.840.113549.1.1.4

MD5 Fingerprint: FC C6 E2 E8 77 BC 0C CA B0 1E 85 D1 DD 29 20 F
SHA-1 Fingerprint: 0D C1 AC CF C5 EE 10 C7 FB 37 5A 01 26 E0 1F
SHA-256 Fingerprint: BA F8 76 D5 54 21 33 31 C6 FE 5F 6B BF 9A
```

Figure 5. Public Key MD5withRSA

```
    <action android:name="android.intent.action.TIMEZONE_CHANGED"/>
</intent-filter>
</receiver>
<receiver android:name="androidx.work.impl.background.systemalarm.ConstraintProxyUpdateReceiver" android:enabled="@bool/enable_system_alarm_service_default" android:exported="false" android:directBootAware="false">
<intent-filter>
    <action android:name="androidx.work.impl.background.systemalarm.UpdateProxies"/>
</intent-filter>
</receiver>
<receiver android:name="androidx.work.impl.diagnostics.DiagnosticsReceiver" android:permission="android.permission.DUMP" android:enabled="true" android:exported="true" android:directBootAware="false">
<intent-filter>
    <action android:name="androidx.work.diagnostics.REQUEST_DIAGNOSTICS"/>
</intent-filter>
</receiver>
<activity android:theme="@android:style/Theme.NoTitleBar.Fullscreen" android:name="com.unity3d.services.ads.adunit.AdUnitActivity" android:exported="false" android:configChanges="fontScale|smallestScreenSize|scre
<activity android:theme="@android:style/Theme.Translucent.NoTitleBar.Fullscreen" android:name="com.unity3d.services.ads.adunit.AdUnitTransparentActivity" android:exported="false" android:configChanges="fontScale|
<activity android:theme="@android:style/Theme.Translucent.NoTitleBar.Fullscreen" android:name="com.unity3d.services.ads.adunit.AdUnitTransparentSoftwareActivity" android:exported="false" android:configChanges="fo
<activity android:theme="@android:style/Theme.NoTitleBar.Fullscreen" android:name="com.unity3d.services.ads.adunit.AdUnitSoftwareActivity" android:exported="false" android:configChanges="fontScale|smallestScreenS
<service android:name="com.google.firebase.messaging.cpp.ListenerService" android:exported="false">
<intent-filter>
    <action android:name="com.google.firebase.MESSAGING_EVENT"/>
</intent-filter>
```

Figure 6. Dump Method

```
    <activity android:name="com.facebook.ads.AudienceNetworkActivity" android:configChanges="screenSize|orientation|keyboardHidden"/>
    <activity android:theme="@android:style/Theme.Translucent.NoTitleBar" android:name="com.google.android.gms.common.api.GoogleApiActivity" android:exported="false"/>
    <activity android:name="com.facebook.CustomTabMainActivity"/>
    <provider android:name="com.facebook.internal.FacebookInitProvider" android:exported="false" android:authorities="com.ea.game.tetris2011_row.FacebookInitProvider"/>
    <activity android:theme="@android:style/Theme.Black.NoTitleBar.Fullscreen" android:name="com.ea.nimble.WebView"/>
    <receiver android:name="com.ea.nimble.NimbleLocalNotificationReceiver" android:exported="true"/>
    <activity android:theme="@style/Theme.MMTransparent" android:label="@string/mmadsdk_app_name" android:name="com.millennialmedia.internal.MMActivity" android:configChanges="layoutDirection|locale"/>
    <activity android:label="sdk" android:name="com.millennialmedia.internal.MMIntentWrapperActivity" android:configChanges="layoutDirection|locale"/>
    <service android:name="com.millennialmedia.internal.task.handshake.HandshakeRequestService" android:permission="android.permission.BIND_JOB_SERVICE" android:exported="true"/>
    <service android:name="com.millennialmedia.internal.task.reporting.PlacementReportingService" android:permission="android.permission.BIND_JOB_SERVICE" android:exported="true"/>
    <meta-data android:name="android.support.VERSION" android:value="26.1.0"/>
</application>
</manifest>
```

Figure 7. Handshaking Method

```
APK signature verification result:

Signature verification succeeded
Valid APK signature v1 found

    Signer BNDLTOOL.RSA (META-INF/BNDLTOOL.SF)

        Type: X.509
        Version: 1
        Serial number: 0x4cad1e2d
        Subject: CN=Unknown, OU=Unknown, O=PONOS, L=Kyoto, ST=Kyoto, C=jp
        Valid from: Wed Oct 06 21:11:09 EDT 2010
        Valid until: Sun Feb 21 20:11:09 EST 2038

        Public key type: RSA
        Exponent: 65537
        Modulus size (bits): 1024
        Modulus: 116629700794149716122819947136756059051245541914555499730824597152770847448900927787599674140584677918103175148531361711

        Signature type: MD5withRSA
        Signature OID: 1.2.840.113549.1.1.4

        MD5 Fingerprint: FC C6 E2 E8 77 BC 0C CA B0 1E 85 D1 DD 29 20 F9
        SHA-1 Fingerprint: 0D C1 AC CF C5 EE 10 C7 FB 37 5A 01 26 E0 1F FC 3A 8C C4 6A
        SHA-256 Fingerprint: BA F8 76 D5 54 21 33 31 C6 FE 5F 6B BF 9A E9 AF 2F 95 C2 0E 82 B1 4B C2 32 B0 AC 3A 77 68 0C B1

Valid APK signature v2 found

    Signer 1

        Type: X.509
        Version: 1
        Serial number: 0x4cad1e2d
        Subject: CN=Unknown, OU=Unknown, O=PONOS, L=Kyoto, ST=Kyoto, C=jp
        Valid from: Wed Oct 06 21:11:09 EDT 2010
        Valid until: Sun Feb 21 20:11:09 EST 2038

        Public key type: RSA
        Exponent: 65537
        Modulus size (bits): 1024
        Modulus: 116629700794149716122819947136756059051245541914555499730824597152770847448900927787599674140584677918103175148531361711

        Signature type: MD5withRSA
        Signature OID: 1.2.840.113549.1.1.4

        MD5 Fingerprint: FC C6 E2 E8 77 BC 0C CA B0 1E 85 D1 DD 29 20 F9
        SHA-1 Fingerprint: 0D C1 AC CF C5 EE 10 C7 FB 37 5A 01 26 E0 1F FC 3A 8C C4 6A
        SHA-256 Fingerprint: BA F8 76 D5 54 21 33 31 C6 FE 5F 6B BF 9A E9 AF 2F 95 C2 0E 82 B1 4B C2 32 B0 AC 3A 77 68 0C B1

Valid APK signature v3 found
```

Figure 8. APK Signatures

```
        Y: 166720175018539134553915612513340009806931655774086085802450703497949231353936247323604511345597115681186736765560729214437920631

        Signature type: SHA1withDSA
        Signature OID: 1.2.840.10040.4.3

        MD5 Fingerprint: 0B 64 56 DC 87 9E 54 30 3E 76 D6 99 90 85 F1 CB
        SHA-1 Fingerprint: 12 F1 98 C1 38 45 05 B5 B2 66 01 2E 3D F0 DC C2 25 E9 CB 43
        SHA-256 Fingerprint: 5B FF 7D 61 4E 1B A1 1A 56 6A BB 58 9C 86 3B 01 3F 79 AA 74 7B D2 14 67 33 36 6C 56 25 A5 F0 D2

Warnings

    Files that are not protected by APK signature v1. Unauthorized modifications to these entries can only be detected by APK signature v2 and higher.

        META-INF/com.android.support_animated-vector-drawable.version
        META-INF/com.android.support_appcompat-v7.version
```

Figure 9. Files Warning with APK Signature

Request

Pretty Raw Hex

```
1  POST /v3/p HTTP/1.1
2  Connection: close
3  Content-Type: application/json
4  Accept-Encoding: gzip, deflate, br
5  Content-Encoding: gzip
6  Device-OS: android
7  User-Agent: 4.4.0/OGY-FD8A4BFB696D/13
8  Package-Name: com.tripledot.woodoku
9  User: 00000000-0000-0000-0000-000000000000
10 Instance-Token: 3de6227b-d4aa-4821-9cdc-b6aff21572d9
11 Api-Key: [OGY-FD8A4BFB696D]
12 Sdk-Version: [4.4.0]
13 Timezone: -05:00
14 Connectivity: WIFI
15 Sdk-Version-Type: ads
16 Sdk-Type: 0
17 Content-Length: 22
18 Host: pad-v3.presage.io
19
20 □«®C¿¦£
```

Figure 10.HTTP request from Woodoku

Request

Pretty Raw Hex \n

```
6  User-Agent: BestHTTP
7  Content-Type: application/x-www-form-urlencoded
8  Content-Length: 3025
9
10 data=
%7B%0A%20%20%22loginType%22%3A%20%22NA%22%2C%0A%20%20%22userLevel%22%3A%201%2C%0A%20%20%22isAddFree%22%3A%20-10
0%2C%0A%20%20%22hideMe%22%3A%20-100%2C%0A%20%20%22adConsent%22%3A%20100%2C%0A%20%20%22stars%22%3A%20300%2C%0A%2
0%20%22bnr%22%3A%200%2C%0A%20%20%22ints%22%3A%200%2C%0A%20%20%22rv%22%3A%200%2C%0A%20%20%22network%22%3A%20%22H
uawei%22%2C%0A%20%20%22campaign%22%3A%20%22%22%2C%0A%20%20%22siteId%22%3A%20%22%22%2C%0A%20%20%22abTest%22%3A%2
0%22default%22%2C%0A%20%20%22abTestId%22%3A%20%22default%22%2C%0A%20%20%22adRev%22%3A%20%220%22%2C%0A%20%20%22i
apRev%22%3A%20%220%22%2C%0A%20%20%22exp%22%3A%200%2C%0A%20%20%22maxExp%22%3A%200%2C%0A%20%20%22hammer%22%3A%200
%2C%0A%20%20%22swap%22%3A%200%2C%0A%20%20%22undo%22%3A%200%2C%0A%20%20%22du%22%3A%200%2C%0A%20%20%22bomb%22%3A%
200%2C%0A%20%20%22x2%22%3A%200%2C%0A%20%20%22shuf%22%3A%200%2C%0A%20%20%22installVersion%22%3A%20507%2C%0A%20%2
0%22installDt%22%3A%20%222023-11-19%2013%3A59%3A29%22%2C%0A%20%20%22afId%22%3A%20%221700421312333-2791487663438
125661%22%2C%0A%20%20%22avatar%22%3A%20%220%22%2C%0A%20%20%22userId%22%3A%20%22afa980c3-1114-4115-91cc-6cc7ff6a
f7dc%22%2C%0A%20%20%22emailId%22%3A%20%22NA%22%2C%0A%20%20%22firstName%22%3A%20%22Guest%201700420369%22%2C%0A%2
0%20%22country%22%3A%20%22Canada%22%2C%0A%20%20%22userImgUrl%22%3A%20null%2C%0A%20%20%22countryImgUrl%22%3A%20n
ull%2C%0A%20%20%22facebookId%22%3A%20null%2C%0A%20%20%22currentOS%22%3A%20%22android%22%2C%0A%20%20%22deviceUID
%22%3A%20%22dea3ae354b6434a88857935cf4eeb4a2%22%2C%0A%20%20%22score%22%3A%20%22000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000%22%2C%0A%20%20
%22maxScore%22%3A%20%22000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000%22%2C%0A%20%20%22totScore%22%3A%20%22000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000%22%2C%0A%20%20%
22savedGame%22%3A%20%22%22%2C%0A%20%20%22blocks%22%3A%20%22%22%2C%0A%20%20%22appVersion%22%3A%20507%2C%0A%20%20
%22pushDeviceToken%22%3A%20%22ft7wODi-R2y8V-NOvEzvp6%3AAPA91bHEBD64nZkIIwcHDKljOBdV-u3nHKeOMXnbCfM6kiylgKC_cBRO
1TSjDxUDz_waNLefsRP3RFI_ufG9NSfQz9kcCGJOAfB22plLpc4Ga9k62yWbVoOXVZCfebCj-aGlkhis34GS%22%2C%0A%20%20%22fbAccessT
oken%22%3A%20null%2C%0A%20%20%22isIAPUser%22%3A%20-100%2C%0A%20%20%22drCount%22%3A%200%2C%0A%20%20%22lastDRAvai
ledID%22%3A%2020201101%2C%0A%20%20%22lastDRAvailedDT%22%3A%20%222001-01-01%2000%3A00%3A00%22%2C%0A%20%20%22adFr
eeSubType%22%3A%200%2C%0A%20%20%22adFreeExpiryDt%22%3A%20%222001-01-01%22%0A%7D&apiSubType=SaveUserProfile&
userId=afa980c3-1114-4115-91cc-6cc7ff6af7dc&firstName=Guest%201700420369&country=Canada&pushDeviceToken=
ft7wODi-R2y8V-NOvEzvp6%3AAPA91bHEBD64nZkIIwcHDKljOBdV-u3nHKeOMXnbCfM6kiylgKC_cBRO1TSjDxUDz_waNLefsRP3RFI_ufG9NS
fQz9kcCGJOAfB22plLpc4Ga9k62yWbVoOXVZCfebCj-aGlkhis34GS&appVersion=507&os=android&loginType=NA&emailId=NA&
fbAccessToken=&facebookId=&appId=jupiter
```

Figure 11. HTTP request from 2248