Spoof-of-Work
Evaluating Device Authorisation in Mobile Mining Processes

Dominik Ziegler[1], Bernd Prünster[2], Marsalek Alexander[2] and Christian Kollmann[3]
[1]*Know-Center GmbH, Graz, Austria*
[2]*Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology, Graz, Austria*
[3]*A-Sit Plus GmbH, Vienna, Austria*

Keywords: Device Authorisation, Android, Cryptocurrency, Mining, REST, App Integrity, Smartphone, Electroneum, Remote Attestation, Key Attestation.

Abstract: Mobile mining of cryptocurrencies, without relying on CPU-heavy computations, is a novel attempt to foster adoption of a token. However, this approach leaves room for attacks. In this paper, we perform a thorough analysis of Electroneum, one of the first cryptocurrencies to introduce a mobile mining process. We show that mobile mining, without relying on a consensus algorithm (e.g. Proof-Of-Work), is not feasible on current generation Android smartphones. We further demonstrate that the security mechanisms employed by Electroneum can be circumvented and that mobile mining can be exploited successfully. Based on this analysis, we discuss several practical countermeasures, which can be applied on smartphones to enforce device authorisation and prevent abuse.

1 INTRODUCTION

The popularity of cryptocurrencies, such as Bitcoin (Nakamoto, 2008), has increased vastly in the past decade. Despite this fact, mass adoption cannot be observed, due to poor user experience (Devries, 2016).

In addition, cryptocurrencies typically require large amounts of computing power, to validate transactions and to reach a common consensus. This is attributed to the fact that cryptocurrencies usually store their transactions in a distributed ledger (usually referred to as *blockchain*). To participate in this consensus mechanism, users need to solve a cryptographic puzzle. This task is commonly called *mining*. As a reward, users receive new tokens.

In theory, this process could also be replicated on smartphones. However, mining poses a significant strain on hardware, especially in a mobile environment with limited battery capacity. Nevertheless, creators of cryptocurrencies seek to foster adoption, and try to make currencies available on mobile devices.

One approach for a mobile mining process is a so-called *Airdrop* of tokens, where a large number of tokens is generated by the creator of the cryptocurrency in advance. Tokens are then awarded to users, e.g., for using a dedicated smartphone application. Hence, phones do not mine in the conventional sense. Instead, mobile mining apps periodically benchmark the theoretical system performance and report performance figures to a back-end service. Based on this, the system awards the user with tokens. This approach significantly reduces the load on the device. However, one major problem of mobile mining is that the back-end can hardly validate whether it is interfacing with a benign application executed on a genuine device.

The scientific contribution of this paper is twofold. First, we analyse the mobile mining process of *Electroneum* (Electroneum Ltd, 2017), one of the first cryptocurrencies to introduce mobile mining. We show that if the reward system is not solely based on Proof-of-Work (PoW) or similar consensus algorithms, and no additional measures are in place to ensure execution on a real device, it is impossible to prevent tampering with the mobile mining process. Secondly, we discuss currently practical countermeasures and show how to authorise mobile devices. We further describe how to mitigate protocol exploitation and prevent device and app impersonation. To do so, we first introduce the topic of cryptocurrencies and Android application security. Subsequently, we describe the Electroneum architecture and discuss possible attack vectors. Finally, we propose a solution based on existing as well as novel approaches for device authorisation in the context of Android applications.

Ziegler, D., Prünster, B., Alexander, M. and Kollmann, C.
Spoof-of-Work - Evaluating Device Authorisation in Mobile Mining Processes.
DOI: 10.5220/0006859003800387
In *Proceedings of the 15th International Joint Conference on e-Business and Telecommunications (ICETE 2018) - Volume 2: SECRYPT*, pages 380-387
ISBN: 978-989-758-319-3

2 BACKGROUND

This work focuses on analysing the feasibility of resource-saving cryptocurrency mining on smartphones. We, therefore, discuss cryptocurrencies in brief. We also provide background information regarding the Android Operating System (OS) and mobile application development.

2.1 Cryptocurrencies

The first blockchain-based cryptocurrency, Bitcoin, was conceived in 2008. It uses a fully decentralised consensus algorithm and the blockchain as a distributed ledger, which records all transactions in blocks. Each block is linked to is its predecessor. To create a block, so-called miners, try to solve a cryptographic puzzle. Whoever manages to solve this puzzle, and thus manages to create a new block, receives a so-called block reward and the fees of all transactions included in the block. This consensus mechanism is called *Proof-of-Work (PoW)*. The difficulty of the puzzle is regularly adapted with the goal of an average block creation time of ten minutes. It is this competition among the miners which secures the blockchain. An attacker who wants to modify or delete a block in the blockchain must thus solve a puzzle at least as hard as the block to replace. If this block is not on top of the chain, the attacker must also recalculate all subsequent blocks. Thus, an attacker needs significant computing resources to outperform the remaining, honest miners.

However, the price of this security mechanism is high power consumption, as all miners race against each other and only one miner per block will finally win and earn the reward. In recent years, several developers thus tried to improve various cryptocurrencies in different aspects, like scalability or privacy.

One of these cryptocurrencies is Electroneum. It focuses on user privacy and uses the *CryptoNight* (Seigen et al., 2013) PoW algorithm. Still, it is not optimised for mining on mobile devices, as the high load on the CPU would lead to an increased ageing rate of specific hardware components (Leng et al., 2015). Therefore, Electroneums' mobile mining process does not rely on the PoW algorithm to secure the blockchain. Instead, Electroneum distributes pre-mined coins for the mobile mining process (Electroneum Ltd, 2017).

2.2 Android Application Development

Android provides an ecosystem for third-party developers to create and publish applications. Applications resemble, in principle, a zip archive file. This structure and the fact that Android applications are typically based on Java or Kotlin enables easy decompilation and analysis (Enck et al., 2011). Mobile applications for Android can also be developed using Web technology. As such hybrid applications typically are not converted to native instructions, code can easily be extracted.

2.3 Android Security Model

Android incorporates extensive security mechanisms (Enck et al., 2009). Due to the scope of this work, we do not provide a thorough analysis of the architecture. Instead, we discuss specific in-place security features necessary for the analysis of Electroneum and refer to the official documentation[1].

Android, executes each application within a dedicated virtual memory in an Application Sandbox. This process ensures that applications do not access or modify (private) files of other applications.

With *Trusty*[2], Android also offers a Trusted Execution Environment (TEE). TEEs offer a secure and tamper resistant runtime environment for applications, separated from the actual OS. As of this writing, Trusty does not support third-party applications. Instead, all applications are developed and packaged into an image by a single manufacturer. The digitally signed image is then verified by the bootloader on startup.

Android provides pre-installed root Certificate Authoritys (CAs) for network security. Device manufacturers are not allowed to modify the system provided CAs, which are located on the read-only partition. By default, applications only trust system provided CAs.

3 RELATED WORK

To the best of our knowledge, no extensive analysis of the security of mobile mining approaches exists. However, a significant amount of research has been conducted in the field of client-side detection of application vulnerabilities or mobile malware. Thus, we first discuss related work targeting Android security and provide an overview of selected device verification and authorisation mechanisms.

3.1 Android Security

Many tools have been developed to detect malware on Android (Enck et al., 2009; Vidas and Christin, 2014; Karbab et al., 2016). As a result, adversaries shift their focus to so-called app repacking attacks. In

[1] https://source.android.com/security/

[2] https://source.android.com/security/trusty/

these attacks, an attacker modifies well-known applications, to, for example, acquire account information. Typically, users can hardly detect whether a binary has been tampered with. Hence, several solutions target this problem by providing detection mechanisms or frameworks to automatically identify rogue Android applications (Desnos and Gueguen, 2011; Jung et al., 2013; Zhou et al., 2013; Huang et al., 2013; Ren et al., 2014). However, these approaches do not account for an adversary who is actively modifying an application to manipulate genuine functionality of the app.

Further, several studies have examined whether Android applications are vulnerable to specific TLS attacks or exhibit substantial flaws in their TLS implementations (Fahl et al., 2012; Georgiev et al., 2012; Sounthiraraj et al., 2014). Results show that a significant amount of applications are either vulnerable to Man-In-The-Middle (MITM) attacks or do not provide sufficient transport layer security. These studies focus on monitoring network traffic in a passive attack scenario. In contrast to that, our approach covers active, deliberately altered network traffic.

3.2 Device Integrity & Authorisation

Multiple approaches, which allow service providers to verify whether an Android device is in a trusted state, have been proposed in the last decade (Nauman et al., 2010; Bente et al., 2011; Jeong et al., 2014). These attestation mechanisms, typically also allow detecting if an application has been installed and whether it has been altered or not. Thus, service providers can not only rely on the fact that the device is in an uncompromised state but also that an adversary has not tampered with installed applications. However, while these approaches provide a solution, most of them are not integrated into the Android system. Additionally, some approaches are either purely academic, and their proof of concept is based on Trusted Computing (TC) and Trusted Platform Modules (TPMs), neither of which are currently available at large scale on mobile platforms, or they are implemented in software, which, in theory, allows an adversary with root access to tamper with this mechanism. Hence, current service providers cannot directly rely on these techniques. Similarly, related approaches targeting Android application integrity, such as static or dynamic code integrity checks at runtime are currently not integrated into the Android operating system.

4 ANALYSIS: ELECTRONEUM

We provide an overview of the Electroneum mobile miner and discuss its system security mechanisms and properties. We further show how the mobile mining process can be tampered with.

4.1 System Architecture

The Electroneum mobile miner is a mobile application relying on the *Apache Cordova*[3] framework. As such it is implemented using mainly JavaScript and HTML. To analyse as well as attack the application we could thus directly modify JavaScript files. In the following, we describe core elements of the mobile mining process we learnt from this analysis.

Authentication & Authorisation: Electroneum enforces a Two-Factor Authentication (2FA) mechanism consisting of a combination of username and password as well as a user-defined PIN. Furthermore, a valid phone number is necessary. To authorise devices, the server first authenticates the user. Subsequently, it evaluates the transmitted *User-Agent* header to identify the accessing device. If the device is not yet known, a verification email is sent. A *reCAPTCHA*[4] challenge is presented, to prevent automatic device authorisation. A user can only continue if they can successfully solve the reCAPTCHA. Once a user successfully authorises a device, it can be used for mobile mining.

Benchmarking: The mobile miner application does not rely on PoW. Instead, a custom benchmarking algorithm, depicted in Algorithm 1, is used. This mechanism evaluates the CPU performance without relying on expensive hash operations. Hence, battery drain of smartphones is reduced. In the first step of this process, the hash duration, i.e. the maximum time the benchmarking process is allowed to run, is retrieved from the server. Next, the algorithm retrieves the current system time. These two parameters serve as the termination condition for a *while* loop. In each iteration, several slow operations, i.e. regular expression matching, are performed. At the end of each loop, the program retrieves the current system time again and increases a counter (*amt*). As soon as the program reaches its termination condition, it sends the number of loops (*amt*) to the server. The server will then determine the hash rate awarded to the device.

Mining: One of the goals of the mobile miner application is to reduce battery drain. Hence, the application does not perform any network validation services. Instead, the application periodically performs a system benchmark. Subsequently, the mobile miner transmits

[3]https://cordova.apache.org

[4]https://www.google.com/recaptcha

Algorithm 1: Benchmarking algorithm used in Electroneum Mobile Miner.

```
hash_duration ← getDurationFromServer()
start ← currentMillis()
end ← 0
while end < (start + (hash_duration * 1000)) do
    regexTest('o','HelloWorld!')
    indexOf('o','HelloWorld!') > −1
    regexMatch('HelloWorld','o')
    end ← currentMillis()
    amt ← amt + 1
end while
```

the benchmark results to the server. Based on the calculated theoretical hash rate the server awards the user with tokens. As long as a device notifies the service periodically, the application is considered active, and the service continuously awards the user with tokens.

4.2 Security Mechanisms

Electroneum employs several mechanisms to prevent fraud. We discuss prominent features we discovered.

Account Verification: To create a mobile mining account, first, a valid email address is required. Once the user confirms control over its address, a valid phone number is required. The server verifies phone numbers via an SMS-verification challenge. In each step, the user further needs to solve a reCAPTCHA.

Maximum Hash Rate: Our analysis suggests that the maximum hash rate for one device is limited to 50 H/s[5]. A device claiming better performance is automatically banned. Once the device claims lower computing power again, mobile mining is resumed.

Maximum Number of Devices: Electroneum does not restrict logging into the same account with multiple devices. However, our tests implicated that using multiple devices linked to a single account does not increase the number of tokens received.

Device Authorisation: In the process of analysing the Electroneum API, we observed that devices seem to be authorised based on the supplied User-Agent header as well as the source IP address. Once the IP address changed, the device needed to be authorised again.

Automation Protection: Electroneum claims to provide systems to protect against large automation systems. Our tests have indicated that once more than five different accounts are used with a single IP address, users need to reauthorise their accounts periodically. This process too requires users to solve a reCAPTCHA challenge.

[5]This number does not represent the actual hashing performance. Instead, it only represents a performance index.

4.3 Attacking the Mobile Mining Process

Based on the analysis of the system architecture, conducted in Section 4.1, we describe possible approaches to bypass the security mechanisms. The procedures described in this section enables the emulation of an arbitrary number of devices. As a consequence, an adversary can, in theory, illegitimately obtain a virtually unlimited number of tokens.

4.3.1 Attack Classes

We present three different methods to attack the mobile mining process. These procedures include modification of network traffic, modification of the application and application impersonation.

[A1] - Modification of Network Traffic: An adversary may alter network traffic transmitted to the Electroneum servers. As a result, they can submit arbitrary values, including benchmarking results. Modification of network traffic, in most cases, does not require changing the application code. However, recent studies have shown that Android applications are relying on TLS connections on a large scale (Fahl et al., 2012). When implemented correctly, TLS can prevent tampering with network traffic—on uncompromised devices at least. Considering that applications, since Android 7.0 (Android Developers Blog, 2016), do not automatically trust user imported CAs certificates, this attack requires additional modifications of the mobile operating system.
In general, several tools exist to modify network traffic on Android devices. For example, traffic can either be altered on the device directly via Virtual Private Network (VPN) applications or via proxies running on desktop machines such as *Burp Suite*[6].

[A2] - App Repackaging: The Electronum mobile miner is using the Cordova framework. Such apps do not need to b compiled to machine code. As a result, Cordova apps are easy targets for repackaging attacks, as shown by (Kudo et al., 2017). When attacking the application, an adversary can directly modify the benchmarking algorithm described in Algorithm 1. Hence, it is possible to skip benchmarking altogether. By repackaging the application, an adversary can thus create modified binaries and distribute them on multiple devices. As a result, high hash rates can be spoofed even on outdated hardware.

[A3] - Application Impersonation: A significant problem of REpresentational State Transfer (REST) APIs is the verification of clients. In fact, by exposing a REST API, a server can hardly distinguish between

[6]https://portswigger.net/burp

trusted and rogue clients. An adversary, who can reconstruct the network protocol, could thus impersonate the Electroneum mobile miner. Indeed, the impersonation of the application also entails that an adversary can run multiple instances of an application on a single machine. This fact, however, contradicts the goal of mobile mining. An attacker who can operate a virtually unlimited amount of app instances would also gain an unjustified advantage over regular users.

4.3.2 Results

Our tests indicated that the security mechanisms can be successfully circumvented and an arbitrary amount of mobile miners could be emulated on a single machine.

[A1]: To test whether the Electroneum mobile mining process is vulnerable to modifications of network traffic, we tried to directly tamper with the outgoing network traffic of the application. The Electroneum application does not enforce certificate pinning (c.f. Section 5.3). Thus, we used *Magisk*[7], to import a proxy CA, into the system's trust store. As a result, we were able to tamper with the benchmarking results the application sent to the server. Relying on this method, we could achieve the maximum possible hash rate, even on outdated hardware.

[A2]: We performed this attack by repackaging the application and exchanging the benchmarking algorithm. Our test device, a Samsung Galaxy S5 achieved around 6 H/s without any modifications. After modifying the source code, the modified binary was able to claim a maximum hash rate of 50 H/s.

[A3]: We reconstructed the network protocol, used by the mobile miner. We then developed an application which emulates the behaviour of the Electroneum mobile miner. We used *Tor*[8] and fixed exit nodes to prevent blocking of IP addresses and keep virtual devices authorised. We set the benchmarking results to match values between 42 H/s and 47 H/s to reduce possible conspicuousness. With this setup, we successfully evaded the in-place automation protection and emulated more than five devices on a single machine.

To summarise, all employed techniques allowed us to either tamper with the mobile mining process or completely emulate a device. This setup allows us to mine approximately 9.36 ETN/day per account. A PoW miner with a comparable hash rate, could mine around 1.787 ETN/day (numbers based on values from March 2018). Figure 1 shows the successful payout of tokens using a single emulated miner. In general, an adversary can profit from these loopholes by creating a large number of accounts and running multiple instances on a single machine. This scheme can even be extended to a larger scale as multiple services exist to solve reCAPTCHA automatically. Phone number verification can also be circumvented by relying on virtual phone numbers to receive SMS, with prices ranging from 0.2 USD/month to 1.5 USD/month.

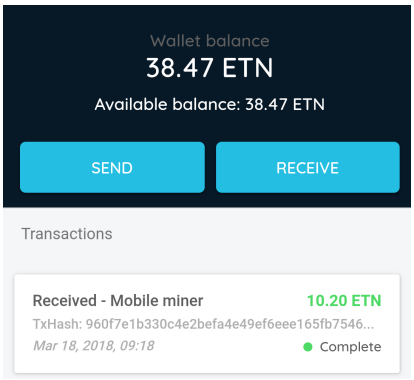

Figure 1: Successful payout with impersonated miner.

These results lead to one conclusion: Tamper proof mobile mining, without consensus algorithms or proper device authorisation and verification is infeasible.

5 DEVICE AUTHORISATION

We have shown that we can easily tamper with the Electroneum mobile mining process. We argue that this behaviour is due to two major design flaws. First, the mobile mining process does not ensure that the application is running on a smartphone. Second, the benchmarking algorithm is designed to replace complex computations. However, the server cannot validate reported numbers, which, by definition, renders the process practically ineffective. We propose several currently feasible procedures, to ensure proper device authorisation, with high certainty. Electroneum is used as the prime example, but the methods described are suitable for any case of device authorisation.

5.1 Remote Attestation (SafetyNet)

Remote attestation is one of the most important features of TC (Haldar et al., 2004). It allows a server to establish a trust relationship with a client, running in an unmanaged environment. Although many different approaches for remote attestation exist, in this work, we focus on the most prominent one for Android, namely *SafetyNet* (Android Open Source Project, 2018a).

[7]Magisk allows gaining root access on an Android operating system without modifying the system. It has been successfully evading virtually all relevant Frameworks aimed at detecting system modifications.

[8]https://www.torproject.org

SafetyNet allows applications on Android devices to request an attestation about the device they are running on. Locally, SafetyNet loads (signed) code from Google servers[9]. The process runs in the background and continuously gathers data from various sources on the device. All information gathered that way is sent to the back-end service operated by Google and analysed in private. The result of an attestation request reports on e.g. a locked bootloader, a custom system image, the certification of the manufacturer and essential integrity features like running on a non-emulated and not-rooted device. Additionally, the attestation result includes the package name of the requesting app, a digest of the application package as well as a digest of the certificate used to sign the package. However, several tools, including Magisk, successfully bypass this basic integrity check and provide a rooting method that is not detected.

5.2 Device Attestation (Key Attestation)

Device attestation mechanisms enable a device to prove its integrity. Typically, TEEs are used for this process. However, as discussed in Section 2.3, Android does not provide a TEE for third-party applications. We present an approach, how device attestation can be achieved on current-generation smartphones.

Since Android 7.0, applications can request attestations about key pairs generated in the Android KeyStore (Android Open Source Project, 2018b). The attestation contains information about the key pair itself, e.g. the algorithm, key size, and whether user authentication is required to use the private key. Optionally, the attestation also includes information about the application itself: the application identifier and a digest of the signature of the package. Additionally, information about the system may be included: The version of the operating system, the patch level, and the state of the verified boot chain. Most importantly, security levels for both the attestation and the KeyStore implementation are provided. The value indicates the type of the respective module: It may either run as a system-level software or inside a TEE. The data structure representing the attestation can be authenticated by verifying the chain of certificates signing the data. If the attestation was generated in secure hardware, the root certificate contains a known public key from Google.

Hence, to establish trust with a device, a server sends an attestation challenge to the application. Subsequently, the application creates a new key in the Android KeyStore and attaches this challenge. Next, the application sends the attestation, which was created during this process, to the server. Finally, the server verifies the transmitted attestation. If successful, access can be granted, and the device is authorised. However, only very few devices run Android 7.0 and support hardware-level key attestation, and nearly all information in the attestation structure is marked as optional. Therefore, on most devices, the attestation is generated by the Android system in software and can thus be tampered with by an attacker. Further, attestation via the Android KeyStore cannot guarantee the integrity of an Android application and may not include all desirable information. However, if the KeyStore is hardware-backed and the device is not rooted, this process can guarantee that an application is running on a genuine device.

5.3 Certificate Pinning

We have shown that a significant amount of applications already relies on TLS connections to secure communications between client and server. However, only a small amount of applications also implements certificate pinning. Certificate pinning, also known as Public Key Pinning, allows an application to "pin" a specific X.509 certificate. In general, this mechanism enables the application to verify a particular certificate of a server upon connection establishment. Depending on the desired security level, applications can choose to either trust root, intermediate or leaf certificates. However, leaf certificates typically have a shorter expiry time. Thus, applications employing certificate pinning need to be redistributed or updated on a regular basis. Certificate pinning also mitigates MITM attacks, as even system-wide certificates trusted by the OS are rejected. However, it cannot protect against repackaging attacks or network traffic modification on rooted devices.

5.4 Comparison

In the following, we will discuss how the presented mechanisms can ensure device authorisation to a high degree. However, it should be noted that it is currently not possible to guarantee that an application running on an arbitrary, unmanaged Android device has not been modified. As a result, all attacks discussed in Section 4 are, in theory, still feasible.
Table 1 provides an overview of the aforementioned countermeasures. It also indicates whether a technique can mitigate a specific attack. A filled circle implies that the countermeasure can prevent a certain attack on unrooted devices. A partially filled circle indicates that the effort for a successful attack can be increased with this mechanism. An empty circle implies that this

[9]https://koz.io/inside-safetynet/

Table 1: Overview of various attacks and possible countermeasures.

	Traffic Modification	App Repackaging	App Impersonation	Rooting
SafetyNet	○	●	●	◐
Key Attestation	○	○	●	◐
Certificate Pinning	●	○	○	○

technique does not help mitigating an attack.

SafetyNet: We argue that SafetyNet can prevent app repackaging attacks as it can attest the certificate, the Android Application Package (APK) was signed with. This can also prevent app impersonation attacks, as SafetyNet can attest whether the application runs on a genuine device. To establish a trust relationship with and authorise a certain client a SafetyNet attestation can be requested.

Android KeyStore: In principle, this procedure allows drawing conclusions about the state of a device. This includes attestation concerning the state of the bootloader (unlocked or not) or installed applications. However, KeyStore attestation does not attest whether a device has been rooted. Still, we can rely on this information to restrict access to a service. As a result, a service provider can exclude devices with, unlocked bootloaders. Indeed, KeyStore attestation suffers from two drawbacks. First, it is not available on all devices. Second, if the KeyStore is not hardware-backed, an adversary can still tamper with the attestation process.

Certificate Pinning: Certificate Pinning is an effective way to prevent monitoring or modification of network traffic. In combination with remote or device attestation, an attacker can be prevented from tampering with environment variables.

Summarising, applications employing all presented approaches can significantly increase the chances for a service provider to successfully detect the device state.

5.5 Improvements

One of the goals of the Electroneum mobile mining process is to enforce adaption of their token. However, we have shown in Section 4.3 that the current implementation can successfully be exploited and large-scale app impersonation is, in theory, possible. We, therefore, propose several procedures, which can harden the mobile mining process and increase the detection rate of manipulation attempts.

First, we propose to alter the benchmarking algorithm, to include server-side verification of benchmarks. For example, the server can send some random nonce to the device. Similar to conventional PoW implementations, the device needs to solve a cryptographic puzzle of a certain difficulty. The server then measures the time the device needs to solve this task. This way, repackaged applications cannot simply skip the benchmarking process. As benchmarks are not performed constantly, this approach should also not affect battery life. However, relying on server-side verification of benchmarks can increase the load on the server.

Naturally, this countermeasure does not prevent app impersonation attacks. An adversary can still perform these calculations on a much more powerful device. Similarly, it is possible to emulate multiple devices on a single machine due to benchmarking processes not being performed constantly. Hence, we advise to employ both SafetyNet as well as Android KeyStore device attestation mechanism, to enforce execution on a genuine device.

Admittedly, we cannot prevent app impersonation on rooted devices. An adversary can still emulate a genuine device, by forwarding attestation challenges to a device with a modified, rooted OS. In this setting, benchmarking computations are performed on dedicated hardware, whereas valid attestation claims are generated on a smartphone.

6 CONCLUSIONS

We have conducted an extensive analysis of the Electroneum mobile mining process. The analysis shows, that even though mechanisms to protect against automated systems exist, the in-place security mechanisms seem to be vulnerable to app repackaging attacks as well as app emulation. In our tests, we successfully emulated multiple mobile miners on a single machine leading to an accumulated hash rate equal to several PoW miners. We further showed that for these attacks no root access or modifications of the operating systems were necessary. Based on these vulnerabilities we concluded that a mobile mining process without verification of benchmark results could lead to exploitation of the reward system. We further have presented several approaches which can be used to increase the trust relationship between server and client applications. We adhered to responsible disclosure guidelines and informed the developers about our findings.

REFERENCES

Android Developers Blog (2016). Android Developers Blog: Changes to Trusted Certificate Authorities in Android Nougat.

Android Open Source Project (2018a). Protecting against Security Threats with SafetyNet.

Android Open Source Project (2018b). Verifying Hardware-backed Key Pairs with Key Attestation.

Bente, I., Dreo, G., Hellmann, B., Heuser, S., Vieweg, J., von Helden, J., and Westhuis, J. (2011). Towards Permission-Based Attestation for the Android Platform. In *Lecture Notes in Computer Science*, volume 6740 LNCS, pages 108–115.

Desnos, A. and Gueguen, G. (2011). Android: From Reversing to Decompilation. *Proc. of Black Hat Abu Dhabi*, pages 1–24.

Devries, P. D. (2016). An Analysis of Cryptocurrency, Bitcoin, and the Future. *International Journal of Business Management and Commerce*, 1(2):1–9.

Electroneum Ltd (2017). Electroneum overview & white paper.

Enck, W., Octeau, D., McDaniel, P., and Swarat Chaudhuri (2011). A Study of Android Application Security William. In *Proceedings of the 20th USENIX Security Symposium*, pages 315–330.

Enck, W., Ongtang, M., and McDaniel, P. (2009). On Lightweight Mobile Phone Application Certification. *Proceedings of the 16th ACM conference on Computer and communications security - CCS '09*, page 235.

Fahl, S., Harbach, M., Muders, T., Smith, M., Baumgärtner, L., and Freisleben, B. (2012). Why Eve and Mallory Love Android: An Analysis of Android SSL (In)Security. *Proceedings of the 2012 ACM conference on Computer and communications security - CCS '12*, page 50.

Georgiev, M., Iyengar, S., Jana, S., Anubhai, R., Boneh, D., and Shmatikov, V. (2012). The Most Dangerous Code in theWorld: Validating SSL Certificates in Non-Browser Software. In *Proceedings of the 2012 ACM conference on Computer and communications security - CCS '12*, page 38, New York, New York, USA. ACM Press.

Haldar, V., Chandra, D., and Franz, M. (2004). Semantic remote attestation: a virtual machine directed approach to trusted computing. In *USENIX Virtual Machine Research and Technology Symposium*.

Huang, H., Zhu, S., Liu, P., and Wu, D. (2013). A Framework for Evaluating Mobile App Repackaging Detection Algorithms. In Huth, M., Asokan, N., Čapkun, S., Flechais, I., and Coles-Kemp, L., editors, *Trust and Trustworthy Computing*, pages 169–186, Berlin, Heidelberg. Springer Berlin Heidelberg.

Jeong, J., Seo, D., Lee, C., Kwon, J., Lee, H., and Milburn, J. (2014). MysteryChecker: Unpredictable attestation to detect repackaged malicious applications in Android. In *2014 9th International Conference on Malicious and Unwanted Software: The Americas (MALWARE)*, pages 50–57. IEEE.

Jung, J.-H., Kim, J. Y., Lee, H.-C., and Yi, J. H. (2013). Repackaging Attack on Android Banking Applications and Its Countermeasures. *Wireless Personal Communications*, 73(4):1421–1437.

Karbab, E. B., Debbabi, M., Derhab, A., and Mouheb, D. (2016). Cypider: Building Community-Based Cyber-Defense Infrastructure for Android Malware Detection. *ACSAC '16 (32nd Annual Computer Security Applications Conference)*, pages 348–362.

Kudo, N., Yamauchi, T., and Austin, T. H. (2017). Access Control for Plugins in Cordova-Based Hybrid Applications. In *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, number 2, pages 1063–1069. IEEE.

Leng, F., Tan, C. M., and Pecht, M. (2015). Effect of Temperature on the Aging rate of Li Ion Battery Operating above Room Temperature. *Scientific Reports*, 5(1):12967.

Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. *www.bitcoin.org*, page 9.

Nauman, M., Khan, S., Zhang, X., and Seifert, J.-P. (2010). Beyond Kernel-Level Integrity Measurement: Enabling Remote Attestation for the Android Platform. In Acquisti, A., Smith, S. W., and Sadeghi, A.-R., editors, *Trust and Trustworthy Computing*, pages 1–15, Berlin, Heidelberg. Springer Berlin Heidelberg.

Ren, C., Chen, K., and Liu, P. (2014). Droidmarking: Resilient SoftwareWatermarking for Impeding Android Application Repackaging. *29th ACM/IEEE international conference on Automated software engineering*, pages 635–646.

Seigen, Jameson, M., Nieminen, T., Neocortex, and Juarez, A. M. (2013). CryptoNight Hash Function.

Sounthiraraj, D., Sahs, J., Greenwood, G., Lin, Z., and Khan, L. (2014). SMV-HUNTER: Large Scale, Automated Detection of SSL/TLS Man-in-the-Middle Vulnerabilities in Android Apps. In *Proceedings 2014 Network and Distributed System Security Symposium*, number February, pages 23–26, Reston, VA. Internet Society.

Vidas, T. and Christin, N. (2014). Evading Android Runtime Analysis via Sandbox Detection. *Proceedings of the 9th ACM symposium on Information, computer and communications security - ASIA CCS '14*, pages 447–458.

Zhou, W., Zhang, X., and Jiang, X. (2013). AppInk: Watermarking Android Apps for Repackaging Deterrence. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security - ASIA CCS '13*, page 1, New York, New York, USA. ACM Press.