Authentication via User Profiling

Extended Abstract

André Faustino
Instituto Superior Técnico
andre.faustino@tecnico.ulisboa.pt

ABSTRACT

Online authentication using only a username and a textual password has been proven flawed over years. Some solutions have been adopting different types of passwords or adding more authentication factors.

Due to sometimes requiring expensive hardware or demanding several actions from the user, some of these solutions are unusable in mobile devices.

The solution described in this work is an additional authentication factor that uses the smartphone characteristics and its user configuration in order to create user profiles that will be used to authenticate the user, requiring no actions from the user.

This way, the proposed solution provides a fast authentication that does not requires the user to perform any actions. The proposed solution is also as secure as existing solutions.

Since mobile devices are subject to changes such as the installed applications, memorized networks and accounts, operating system upgrades among others, profile verification must be made allowing small changes.

Therefore, the Authentication Server evaluates whether the freshly received profile is similar enough to the stored profile using defined thresholds. If static attributes are changed, or if a large number of attributes have been changed, then the verification fails immediately, activating the fallback mechanism defined by the third-party application.

KEYWORDS

Online authentication, Two-Factor Authentication, mobile devices, user profile, installed applications

1 INTRODUCTION

Online authentication has been done over the years using a username and a textual password. This system is relatively simple and the user doesn't need to carry extra devices, only needing to remember his username and password.

However, over the years this model has become vulnerable and easy to exploit. Short passwords are easy to crack using brute force attacks or even discover through phishing attacks, as Coombs [8] pointed out, Kaspersky estimated that a phishing gang stole one billion dollars from several banks in 2014.

Social Engineering attacks are also a threat when the password is something personal related to the user. In those cases an attacker may discover the password by knowing pieces of information related to the user or finding that information in social networks.

Long randomly generated passwords are harder to guess and crack but they are also harder to remember, especially if the user uses different passwords for different services [8].

In the particular case of online authentication in mobile devices, there are several approaches but not a definite solution. The most common approaches consist of requiring additional authentication factors, such as biometrics or the presence of some object.

However, these solutions have some problems. Once biometrics are falsified the user may never use them again, since it's very hard or even impossible to replace them. Solutions that require objects being carried by the user are not ideal either, since the object can be lost, stolen or forgotten.

The most commonly used system consists on sending a one-time password (via SMS or a specific application) that the user must input to verify his identity. However, this solution requires the user to perform an undesirable set of actions, including checking the received code, copying or memorizing it and inputting it when asked.

Proposed Solution

The proposed solution solves the problem of online authentication in mobile devices using an authentication factor that is secure and requires minimum user interaction. The used authentication factor is an adaptive unique profile that is composed of unique features of the smartphone and its user configuration like memorized networks, installed applications, memorized accounts, among others.

The solution consists of an authentication server and a smartphone application that will collect the relevant information. Whenever a user logs in the desired third-party application, the information collector application is called, retrieving the needed information and creating the profile.

For each retrieved attribute a hash is calculated, with each profile being a set of the calculated hashes. An HMAC of the profile is calculated using a symmetric AES key shared between device and sever that is stored safely in the smartphone's TEE. Both the profile and its HMAC are sent through

a secure channel to the authentication server alongside with a digital signature of the created profile, to prevent the message from being tampered.

The authentication server will then verify if the digital signature is valid and the integrity of the received information by checking if the received HMAC matches the received profile. If the received information is valid, the server checks if it matches the stored information contained in the user profile.

Due to smartphones being customizable, the proposed solution must deal with changes in the profile. If the profile is too strict, whenever a user downloads a new application, for example, a new profile would be required.

Part of the profile is composed of static attributes, information that is impossible to change, and the other part is composed of dynamic attributes which may suffer changes in a regular use of the smartphone. If a static attribute or a significant part of the dynamic attributes is changed, the user must acquire a new profile, activating the fallback mechanism of the third-party application.

Whenever the server accepts the profile of a user, it will update the stored profile with the new information, making sure that the stored profile always contains the updated information of the smartphone.

Objectives

The objectives of this work are to introduce a new Two-Factor Authentication scheme that is simultaneously as secure as existing solutions, fast and that requires few actions from the user when compared to existing solutions.

Considering that resources, namely CPU, memory, network and battery, are limited in smartphones, the solution should avoid using more resources than necessary.

Contributions

The major contributions for the online authentication problem of the solution described in the section above are the following:

- First solution known so far to use a profile composed of device characteristics and user configuration to authenticate the user.
- Two-Factor Authentication solution that requires no explicit interaction from the user in the best cases and requires the same actions as any other solution in the worst case.

2 RELATED WORK

This section contains an overview of the related work presented in the Thesis document, namely the sections of Two-Factor Authentication, User and System Profiling and Attacker Models.

Two-Factor Authentication

Christian Holz and Frank Bentley in [12] proposed a system that uses fingerprints introduced in the smartphone as a second authentication factor for desktop environments, pointing out that the increased complexity introduced by a temporary code is what keeps most users from using Two-Factor Authentication systems.

Aloul et al [4] proposed a system that use smartphones as security tokens. The system consists on generating a One-Time Password using factors like the smartphone's IMEI and IMSI number, a username, a pin and a timestamp.

Acharya et al [2] proposed a similar system that uses different factors. The factors used by Acharya et al consist on the IMSI number, a pin, a timestamp, a username provided by the server owner and the date of birth of the user.

Harini et al [11] developed a system which they called 2CAuth that uses QR codes in an OTP authentication protocol instead of SMS messages, claiming that distributing One-Time Passwords using SMS is not practical due to the SMS transmission delay.

Dmitrienko et al [10] conducted a study on the security of Two-Factor Authentication implementations of worldwide internet service providers like Facebook, Google, Twitter and Dropbox. Dmitrienko et al found that the major problems consisted on low-entropy One-Time Passwords or its distribution, suggesting that the use of the smartphone's Trusted Execution Environment could overcome these problems.

Schrittwieser et al [14] studied the security of Android messaging applications. Considering the Authentication Mechanism and Account Hijacking attack vector, Schrittwieser et al concluded that most of the evaluated applications have flaws that allow attackers to easily hijack accounts, such as generating the code in the client side, which allows Man-in-the-Middle attacks.

Rob Coombs [8] described how ARM Trustzone can be used with FIDO to replace textual passwords. Coombs started to note that smartphones are vulnerable to malware, allowing information to be read or written outside of its application sandbox. ARM Trustzone aims at mitigating these attacks by storing information and executing code inside its Trusted Execution Environment.

User and System Profiling

Ali et al [3] proposed a system to perform user profiling through browser fingerprinting, by looking at the characteristics or attributes of a browser like plug-ins, time zone, fonts and many other features, making it possible to track users even if they erase their cookies.

Boda et al [5] studied a cross-browser fingerprinting by identifying common attributes of all browsers.

Stöber et al [15] proposed a system capable of performing smartphone fingerprint based on application behavior and the traffic generated by applications, using data mining algorithms and the traffic generated by applications to identify one user.

Dai et al [9] proposed a system to automatically fingerprint Android applications constructing a user profile with that information.

Conti et al [6] proposed a system that aims to identify users based on their actions on Android applications by using machine learning algorithms over the encrypted traffic generated by these applications.

Attacker Models

Cooijmans et al [7] developed a comparison study between the Android Keystore implementation for Android API Level 18 and a third-party cryptography library for Java called Bouncy Castle, defining the three following attacker models, ordered by increasing order of power:

- Malicious app attacker - attacker that tries to attack the Keystore using an application installed on the device.
- Root attacker - attacker that has root credentials and is able to run apps under root permissions and inspect the file system.
- Intercepting root attacker - attacker with the same abilities as the Root attacker but also capable of capturing user-input in real time.

R. Loftus and M. Bauman [13] analyzed if known attacks to Android Operating System were still possible in the version 7. Particularly, Loftus and Bauman note that Brute Force Attacks and Cold Boot Attacks are not possible in Android 7 but Evil Maid Attacks and Fingerprint Bypass Attacks are still possible if the attacker has root capabilities.

3 SOLUTION OVERVIEW

System Architecture

The objective of the proposed solution consists on creating a unique user profile based on unique features of Android smartphones and features regarding user behavior like: installed applications, memorized networks or memorized accounts. The unique user profile will be used as a second authentication factor in an authentication system. This way, the proposed system does not require the user to perform additional tasks to prove his identity since the profile will provide by itself everything the system needs to authenticate the user.

The unique user profiles will be stored in a server that is also responsible for verifying the similarity between two profiles, deciding whether a received profile should be accepted or rejected. If a profile is rejected the third-party's fallback mechanism is activated.

These situations need to be analyzed with caution because they correspond to users logging in different smartphones or users that changed a huge part of their smartphone information. This way, this system is an improvement to the state of the art authentication processes since this system requires fewer user actions in most of the cases and behaves exactly like the state of the art systems in the worst cases, introducing no overhead.

The architecture of the system described by the proposed solution is the following:

- Profiling Server - the server that contains all the information regarding the created user profiles. This server is responsible for receiving the profiles, storing them and verifying whether the received profiles are similar enough to the stored ones.
- Third-party Server - third party server that contains all the information and functionalities provided by the third-party. This server is also responsible for defining the fallback mechanism used when the received profiles are not accepted.
- Information Collector Application - the application is responsible for collecting the information, creating the profile and communicating with both servers.

When implementing the system, an SMS code verification was defined as the fallback mechanism of the third-party part of the application. This mechanism was chosen due to being the most commonly used mechanism by banking applications nowadays, facilitating the comparison between the proposed solution and the most used existing solution.

The system has three main phases: the Registration phase, the Bootstrap phase and the Login phase. The Registration and Bootstrap phases occur only once and have the objective of registering the user in both the Third-Party Server and in the Profiling server, initializing all the required information.

The Login phase is the phase that occurs every other time and has the objective of verifying if the information sent by the Information Collector Application is stated as valid by the Profiling Server, confirming this way that the login is performed by the rightful user.

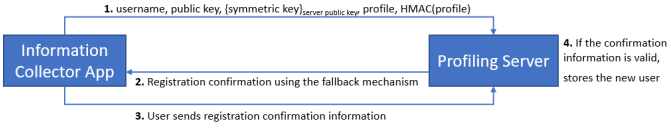

Figure 1: Registration phase of the system

Registration phase can be seen in detail in Figure 1. The Registration phase consists on the user registering himself in the Profiling Server. The Information Collector Application starts by generating an asymmetric RSA key pair and a symmetric AES key to be used to create the profile and sends this

information to the Profiling Server alongside the username, the freshly acquired profile and the calculated HMAC of the profile.

The server sends an SMS code (since it is the defined fallback mechanism) to the smartphone to make sure that the rightful user is registering in the system. If the user replies with the correct code, the server will then store all the received information, registering the user.

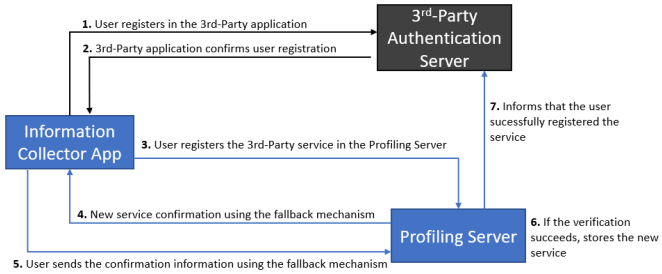

Figure 2: Bootstrap phase of the system

The Bootstrap phase can be seen in detail in Figure 2. This phase occurs when the user joins a new service, provided by the Third-Party. The user starts by registering himself in the Third-Party Server, receiving a confirmation of his registration. Then the user notifies the Profiling Server that he is now using the stored profile in the new service.

Similarly as in the Registration phase, the Profiling server will then send an SMS code to the smartphone to confirm the registration of the new service. If the user replies with the correct code, the server will then store the new service and notify the Third-Party Server that the service was successfully registered.

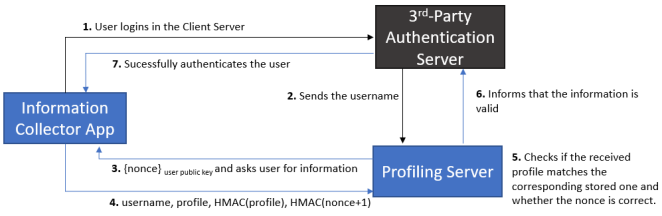

Figure 3: Login phase of the system

The Login phase can be seen in detail in Figure 3. The Information Collector Application starts to communicate with the Third-Party Authentication Server, sending the user credentials. If the credentials have a match in the storage of the Third-Party Authentication Server, the Third-Party will communicate the username to the Profiling Server.

The Profiling Server will send to the user the nonce that should be used to ensure the freshness of the login. The user will be asked if the login request is valid. If the user validates it, the Information Collector Application will acquire the most recent profile and sends it to the Profiling Server alongside the username, the HMAC of a known variation of the most recent received nonce and the HMAC of the profile.

The Profiling server will verify if the received profile is similar enough to the stored one and verify if the received nonce is the most recent one or if the message was a replay of a previous one. If all these parameters check, the server has now authenticated the user.

The Profiling Server notifies the Third-Party Authentication Server that the user is valid. The Third-Party Authentication Server will then successfully authenticate the user, allowing the user to use the Third-Party application.

All the communications are done using HTTPS to ensure confidentiality and integrity of the messages. Each message is also sent along with its digital signature to ensure mutual authentication and non-repudiation.

Profile Verification

In this work, a profile is a set of attributes, where each attribute corresponds to a different feature of the smartphone. Since Android is a customizable Operating System and certain information may change over time, three types of attributes were identified with, with each type requiring a different type of verification. The three types of attributes are the following:

- Static Attributes - information that mostly concerns the smartphone itself and therefore cannot be changed.
- Dynamic Attributes - information that concerns the OS of the smartphone and the SDK version or the SIM card of the user, while some of this information is very unlikely to change over time, change is still a possibility.
- Dynamic List Attributes - attributes that do not consist of a single instance of information but relate to a set of different values. An example is the attribute *Installed Applications* that consists on a set of package names of each installed application in the smartphone.

Due to privacy questions as well as facilitating the profile verification, each attribute consists of the hash of the real value of the attribute. Since lists are exceptional cases, each list attribute consists of a list of hashes. This way, only the smartphone of the user deals with the real information, making it impossible to know the real values of each attribute.

To improve the security of the system, an HMAC of the whole profile is calculated and send alongside the profile, preventing an attacker from generating a similar profile or brute forcing the hashes to know the real information. The symmetric key used to generate the calculated HMACs is securely stored in Android's Keystore, which stores the keys in

Attribute	Median number (per user)
Google Accounts	1
Input Methods	2.42
Installed Applications	53
Memorized Accounts	4.5
Memorized Networks	27.26

Table 1: Mean number for each attribute after First Test Phase

the smartphone's TEE, whenever the smartphone is capable of storing information in the TEE.

Regarding profile verification, the main idea is quite simple: static attributes cannot change. The system should accept changes in dynamic attributes and store the most updated information whenever a profile is accepted. When a big part of the dynamic attributes is changed it is more likely that we are in the presence of a different device than in the presence of the registered device. This way, it is necessary to define a certain threshold to determine whether the number of changed attributes could be accepted or not.

List attributes are once again a special case. It is very frequent that a user changes only a small part of the list instead of the full list, therefore it is necessary to define thresholds that will help to determine whether two lists are similar enough for the profile to be accepted or not.

With all these ideas in mind, the verification algorithm is the following:

- If two profiles are exactly equal verification succeeds.
- If static attributes are changed the verification fails.
- The threshold for dynamic attributes is defined as 75%. This means that considering the 16 dynamic attributes (including lists) the server will only accept up to 4 changed attributes.
- If the verification has not failed in some other points, lists will be verified against thresholds to determine whether the verification could be accepted. If a list fails this verification, the whole verification of the profile is failed. List verification is done considering the common elements and the new elements. The percentage of common elements must be above the threshold and the percentage of new elements must be below *1 - threshold*. The threshold for Installed Applications, Input Methods and Memorized Networks is 75% and the threshold for Google Accounts and Memorized Accounts is 66%.

The threshold for the lists was defined after a preliminary test phase with around 16 users. In this test phase, the median number of each list was calculated, the results can be seen in Table 1.

Given these numbers, considering an example of a user with 50 installed applications, 25 memorized networks and 5 memorized accounts, the same user must have at the next login 37 installed applications out of a potential 62 installed applications, 19 networks out of a potential 31 networks and 3 accounts out of a potential 6 accounts.

Finally, whenever a profile is accepted by the verification the server will update the stored information replacing the old attributes by the changed ones. This way, the stored profile always corresponds to the information sent in the most recent successful login, making sure that a changed value is only processed once and that whenever the changes are not significant enough the system will be able to authenticate the user by itself, not requiring the user to perform further actions, accomplishing that way the objectives of this work.

Note that even though an attacker may try to slowly change the attributes that compose the profile, whenever the rightful user logs in his information will always prevail, which means that attributes changed by an attacker would change to the real value every time the rightful user logs in.

Implementation of the Solution

The two servers described in the System Architecture section were implemented as two HTTPS Java servers that use MySQL databases to store all the relevant information in a persistent way. The decision to implement the servers in Java was done based on both servers being relatively simple, based on how easy it is to implement and test an HTTPS server in Java 8 and since Android runs in Java and some libraries could be used either in the application side and in the server side.

Unit tests were designed using JUnit. The objective of these tests was to test the storage and the server's connection to the storage and test whether the server handlers would give the proper responses to the requests made by the Android application.

The Information Collector application was created using Android 4.4 as the minimum version of the Operating System but an Android 6 application was also done with few adaptations from the original work. The reasoning behind choosing Android 4.4 as the minimum version is that a huge percentage of Android users still use Android 4.4, Android 5 or Android 5.1.

The collection of information is implemented by an Android service that is started by the application whenever the user logs in or registers and the service is stopped whenever the application is dismissed or closed.

Both the asymmetric key pair and the symmetric key used are stored using the Android Keystore, which securely stores the mentioned keys in the device's TEE whenever possible. In this case, all the operations performed using these keys are performed inside the device's TEE and all the key material

generated during these operations are stored in the device's TEE as well.

This means that the operations and the key material used are not stored in normal world memory at the anytime, which means that an attacker is not able to obtain information even if he is capable of reading the device's memory.

Problems found during the Implementation phase

After doing a preliminary test phase with real users it was possible to find three major problems that stood out. The first one was detected while calculating the median number of memorized accounts, it was immediately noticed that only few users had memorized accounts stored in the database. After noticing that some of the users without stored accounts corresponded to users using Android 6, it was found that the application lacked asking for user consent to use the permission GET_ACCOUNTS, which is required in Android from version 6 onwards to access stored accounts in the smartphone, among other information.

The second major problem was a misimplementation of the fallback mechanism on the server. Whenever the retrieved profile did not match the stored one the server and the user asked the application to retrieve a new profile, the server would try to find a stored profile with the same IMEI as the new profile acquired that should be stored. Since some minor bugs were being fixed whenever they were found and the users were installing the application multiples times during the preliminary test phase, the keys stored in the TEE would be overwritten with every new installation making it impossible for the server to match the fresh profile's IMEI with an existing profile, preventing that way users from updating their profile. This situation was particularly problematic since it would also prevent users from having more than one profile stored and in the eventually that the keys are erased from the TEE (which may happen if the user clears the application data, for example) the user would also be prevented of creating a new profile.

The third major problem was the reason of most logins failing. While looking at the logs to understand what made some logins fail it was possible to notice that some users would send a list of memorized networks in one login and send an empty list in the following login. After talking to some of these users trying to understand if they were removing some memorized networks and adding then a list of different ones, users claimed to rarely delete some networks and when asked they also claimed that some logins were done using wireless networks and another logins were done using mobile data. After looking in the documentation of Android, it began clear to understand that whenever users logged in using mobile data with their smartphone's wireless function turned off the application was unable to retrieve the list of memorized networks. Since it is possible in all Android versions so far to change the WiFi state using the WiFi Manager, all it took to fix the problem was adding the permission CHANGE_WIFI_STATE to the Manifest file and then using the WiFi Manager to turn on the WiFi functionality (if it was turned off) right before retrieving the profile, turning it back off again if the user was not using it whenever the users closes the application.

The application also suffered a substantial change in the workflow regarding rejected profiles. This change was a design decision to improve the workflow of the application, requiring fewer actions from the user and helping the users relate more the application with the existing alternatives. Previously, whenever a profile was rejected, the user would be presented with a button that would allow the application to collect a new profile, send it to the server and requiring the user to confirm the creation of the new profile by correctly introducing a code that would be received via SMS. In order to make the application work more closely to existing alternatives, the button phase was eliminated. Whenever a profile is rejected the user receives a code via SMS that the user should input. The application will then acquire the new profile and send it to the server alongside the inputted code. If the code matches the expected one, the server will then store the new user profile. In the existing alternatives the user receives an SMS code in each login without being required to press buttons for it to be sent.

Attacker Models

Taken into account the works of Cooijmans et al [7] and Loftus and Bauman [13], the proposed solution is not vulnerable to common attacks since it does not require any input from the user or flawed biometric authentication such as fingerprints.

As described by Android [1] and further confirmed by Cooijmans et al [7], the keys stored in the TEE are also secured against being extracted from the device, even though they can be extracted from the corresponding application sandbox.

Taken into account the attacker models introduced by Cooijmans et al [7] the only relevant attack vector is an attacker who extracts the keys into his own malicious application. It was only considered in the solution the Android Keystore to securely store the keys and extracting keys directly from the device is impossible, therefore not considered.

Therefore, the model of the attacker considered in the security evaluation of the system is the attacker described by Cooijmans et al [7] as the "Root Attacker". This is a type of attacker who has root capabilities and is able to run applications with root permissions. This attacker is also able of inspecting the Android file system. It is assumed that the device was previously rooted or the attacker has gained root

capabilities through malware capable of Privilege Escalation attacks.

4 EVALUATION

In order to test the application with real-world users, it was selected a group of users that showed availability in downloading the application and using it on a regular basis.

The first test phase occurred from June 2017 to July 2017. The second test phase occurred from the end of July 2017 until mid-September 2017. The comparison between the results of both phases can be seen in Figure 4 5.

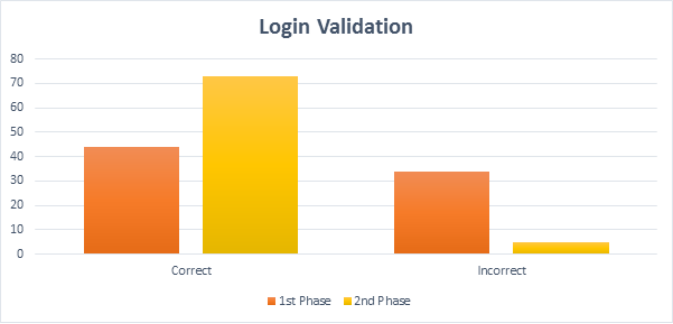

Figure 4: Comparison results of Login Validation in both phases

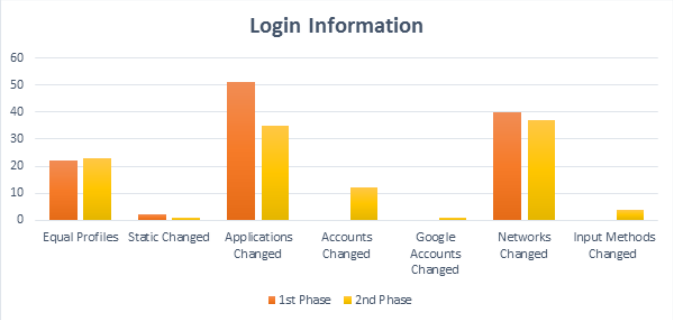

Figure 5: Comparison results of Login Information in both phases

First Test Phase

In the First Test Phase, 19 users participated performing a total of 78 logins. The results are the following: out of the 78 recorded logins, only 44 were considered correct by the system, whereas 34 logins failed in profile verification.

Regarding the types of changed attributes, it is important to note that half of the correct logins corresponded to profiles exactly equal to the expected profile. Both the logins that failed due to a static attribute being changed corresponded to a change in the Software Version, which can be considered as the attribute being wrongly classified, since it is somehow frequent to change.

Another important remark is the fact that there wasn't recorded a single change on accounts, which is explained by the application not asking the required user consent to use the permissions needed to retrieve the stored accounts.

As explained in Section 3, after fixing the problems the value of the defined thresholds was also revisited taking into account the estimated medians calculated after the First Test Phase. The threshold of memorized accounts and input methods was changed from 75% to 66% to accept a bigger number of changes in both lists.

Second Test Phase

In the Second Test Phase, only 12 users chose to participate, which is rather unfortunate since the testing period was bigger than the period of the first phase and the application was not displaying any bugs or problems.

Similarly to the first phase, it was recorded 78 logins, but, in this test phase, only 5 were considered incorrect by the system, one of them with a failed static attribute, which means that the calculated probability of the fallback mechanism being activated is only 6.41%. This translates into a huge advantage when compared to existing Two-Factor Authentication schemes that require lots of actions from the user.

The 5 failed logins had the following problems: one of the failed logins corresponded to a change with the MAC address attribute. It is hard to understand how and why did the user changed his MAC address.

The remaining four failed logins consisted on a user that lost the key used to create the registration profile which can happen if the user clears the application data, two users that changed a large number of memorized networks and a user that changed his stored Gmail account.

Out of the 73 correct logins, only 23 were exactly the same as the expected profile, which is a lower percentage than in the first phase.

STRIDE Analysis

A STRIDE analysis was conducted to identify the major security flaws of the developed system. The results are the following:

Spoofing Identity. Since the profile is created using a symmetric key stored in the smartphone, an attacker wishing to spoof the identity of a rightful user must be able of creating fake profiles for a given user or, in alternative, replay old messages send from the respective user to the server.

Since the server generates a different nonce for every user action and checks if the received messages contain the right nonce, replay attacks are not possible in the system.

Regarding the creation of fake user profiles, an attacker needs access to the symmetric key stored in the user's smartphone and also needs to use user information to make the forged profile match the stored profile.

The attacker may obtain the required information about the user through malware installed in the rightful user's smartphone or through brute force attacks if the symmetric key is known by the attacker.

Even though the symmetric key is stored in the Android Keystore, some attacks may still be possible, especially if the smartphone is rooted. These attacks will be explained in detail later, taking into account realistic attacker models.

Tampering Data. Tampering Data attacks are not possible in the system, since the communication between user and Profiling server is done through HTTPS, which protects against message tampering. The messages exchanged in all communications are also signed to guarantee the authenticity of the message, which also makes it easy to detect tampered messages.

Repudiation. Repudiation is not possible in the system since, as said above, all messages exchanged in every communication between user and Profiling server are signed by the sender.

Information Disclosure. Information disclosure attacks is mitigated in the system, since the profile is a set of hashes and all communications are done through HTTPS. An attacker would need to break the HTTPS encryption to obtain the profile and then break the hash function.

In the case where the attacker obtains access to a profile, he would need the symmetric key to either brute force the profile or break the HMAC function.

The Profiling server stores all the necessary in an encrypted MySQL database which also mitigates information disclosure attacks against the server storage. Further attacks against the server are also not considered, with the server being assumed secure.

Denial of Service. It is possible to flood the Profiling server with fake requests or replays of old HTTPS requests which delays or even denies the server from responding to rightful requests made by the users.

Elevation of Privilege. Elevation of Privilege attacks are possible in the Android Operating System and those attacks are particularly threatening to the system since they may allow an attacker of obtaining access to the symmetric key used to create profiles.

Security Evaluation of the System

Considering the Attacker Models defined in Section 3 and the Threat Analysis done in this chapter, the major security threat to the proposed solution consists on an attacker with root capabilities who is able to extract the keys from the Android Keystore and use it on his own malicious application.

In that case, given that the attacker would have the key and the capability of reading the smartphone's information, the attacker would become able to create false profiles.

The most impactful attack that can be performed with this information is an attacker which is somehow able to emulate the victim's static smartphone attributes and uses the key earned through the attack to impersonate the real user.

However this attack is extremely powerful it is unclear how easy it is for an attacker to create malware capable of retrieving keys stored in the Android Keystore and emulate another device.

Due to Android Keystore being restructured in Android version 6.0 and the lack of recent studies regarding the mentioned vulnerabilities it is unclear whether these vulnerabilities have been fixed. Even though it is recommended that root should be disabled unless the user really needs root capabilities, since it can be exploited by attackers to perform a wide range of attacks against the device.

Evaluation Summary

Considering the objectives defined in Section 1, it is possible to conclude that:

- Fast Authentication that requires few actions - Considering the best case, the system is able to successfully authenticate the user in few seconds without requiring any action from the user. Using the results of the Second Test Phase, the best case happens 93.6% of the times, which means that 9 out of 10 times the proposed solution is better than the existing solution in the number of required actions.
- Security - The only possible attack is a targeted malware that uses root permissions to retrieve the keys used to create the profiles. Despite this attack being extremely impactful, existing solutions are also vulnerable to several types of malware especially if root permissions are available to the malware. Therefore, the proposed solution is at least as secure as the existing alternatives.
- CPU and Memory usage - The system doesn't store anything in memory since the keys are stored in the TEE and the cryptographic operations done there as well. CPU usage is also very low since the application only calls defined Android APIs to retrieve all the necessary information.
- Network usage - The system only exchanges a couple of HTTPS messages with the server, which reduces the introduced overhead in the network.

- Battery usage - Since the application only runs the collector service when needed and stops it when the user exits the application, the overhead in the battery is minimal. Location is not calculated using GPS which also helps in not reducing the battery life.

5 FUTURE WORK

Considering the scope of the work and the work that was done, there are three major directions to follow in the future:

- Ask for parts of the profile instead of the whole profile - it consists in defining some fixed attributes that should be send all the time and a set of attributes that may be or not asked by the system. This idea is based on the premise that a subset of the 20 retrieved factors would be able to uniquely authenticate the user in a secure way and would further increase the security of the entire system since an attacker that is able to break one message once does not get an idea of the whole profile but only a small part of it.
- Machine Learning algorithms to support the decision of verifying a profile - it provides a different type of verification that does not rely only on counting the common elements from profile to profile but a different type of verification that would take into account the profile of the user. A possible use case would be the system considering not only the name of each installed application but also its type, based on the premise that a user that has different applications of a type is more likely to install or uninstall applications from the same type than from a different one.
- Extend the solution to accept profiles from multiple devices - an interesting and challenging approach would be to store only one profile and use multiple devices to authenticate the user. The main idea would be to extract key user configuration features in order to check whether these features are present in a different smartphone or not. This idea is based on the premise that a user is highly likely to install the same set of applications in the smartphones he regularly uses, similarly memorizing a set of accounts and networks as well that is present in all the smartphones as well.

6 CONCLUSION

Online authentication has been done through the years using a username and a textual password. While this system is relatively simple, it has been proven insecure due to problems regarding textual passwords.

Short passwords are easy to crack using brute force attacks and in some cases easy to guess through social engineering attacks. Long passwords are harder to guess and crack but they are also harder to remember.

In mobile devices, some solutions to this problem include the generation of a One-Time Password that is send to the user through SMS or through an application. Users in general avoid using this type of authentication due to being required to perform additional undesired actions.

The solution described in this work consists in a new Two-Factor Authentication scheme that uses smartphone characteristics and its user configuration to create unique user profiles that will be used to authenticate the user.

In the solution a profile is a set of hashes calculated for each individual attribute. The profile is then used to calculate an HMAC using a symmetric key stored in the device's TEE. That way, the privacy of the profile is guaranteed since the client is the only one who can really know what the value of the attribute, since there is no way of inverting a hash function.

There are two types of attributes: static attributes and dynamic attributes. Static attributes are attributes related to the device's hardware or with the device itself and are not supposed to change. Dynamic attributes are attributes related to the device configuration made by the user and can change.

Profile verification fails whenever static attributes are changed since it means that the device is not the same of the device used to register the profile. Regarding dynamic attributes, there is a defined threshold of 75% which means that in every login, 11 out of the 16 dynamic (12 dynamic attributes plus 4 lists) attributes should remain constant.

Since lists like are special dynamic attributes as they are more likely to suffer changes, its verification must be adapted as well. Instead of just checking if the lists are equal, the solution checks if 75% of the list remains constant and whether the percentage of new information is not greater than 25% of the size of the new list.

If the profile verification is passed, the new profile is stored in the server, making sure that the server is always updated with the most up-to-date information. If the verification fails, the fallback mechanism of the third-party application is activated and the user will need to provide information to ensure his identity. The selection of the fallback mechanism is defined by the third party, being out of the scope of this work.

After two testing phases with smartphone users the correctness of the solution was evaluated. The first phase was extremely helpful to detect several problems which made the verification fail more than it should, accounts were not being retrieved in Android 6 onwards and the list of memorized networks was being send empty whenever the user was using mobile data with the WiFi capability turned off.

The results after the second test phase were really positive, out of 78 logins made by 12 users over a month, only 5 of those logins failed the verification. This introduces an

improvement in terms of requiring actions performed by the user, since in only about 1 login out of 10 the user is required of using the fallback mechanism.

Regarding the security of the system, after analyzing all the threats, the only identified attack is targeted malware that abuses root capabilities to steal the key used to create the profile. Since there is no definite solution against malware and the security of the solution is dependent on how Android secures the keys in the TEE, the solution described in this work is at least as secure as the existing alternatives.

The solution also accomplishes the objectives of using the resources of the smartphone wisely, the solution does not make an abusive use of the CPU or the Memory since information is stored in the TEE and some secure operations are done there as well and the overhead introduced in the network is minimal since the application only exchanges a couple of messages with the server.

REFERENCES

[1] [n. d.]. ([n. d.]). https://source.android.com/security/trusty/

[2] Sagar Acharya, Apoorva Polawar, and P Pawar. 2013. Two factor authentication using smartphone generated one time password. *IOSR Journal of Computer Engineering (IOSR-JCE)* 11, 2 (2013), 85–90.

[3] Murad Ali, Zubair Shaikh, Muhammad Khan, and Taha Tariq. 2015. User Profiling Through Browser Finger Printing. In *International Conference on Recent Advances in Computer Systems*. Atlantis Press.

[4] Fadi A. Aloul, Syed Zahidi, and Wassim El-Hajj. 2009. Two factor authentication using mobile phones. In *The 7th IEEE/ACS International Conference on Computer Systems and Applications, AICCSA 2009, Rabat, Morocco, May 10-13, 2009*, El Mostapha Aboulhamid and José Luis Sevillano (Eds.). IEEE Computer Society, 641–644. https://doi.org/10.1109/AICCSA.2009.5069395

[5] Károly Boda, Ádám Máté Földes, Gábor György Gulyás, and Sándor Imre. 2011. User Tracking on the Web via Cross-Browser Fingerprinting. In *Information Security Technology for Applications - 16th Nordic Conference on Secure IT Systems, NordSec 2011, Tallinn, Estonia, October 26-28, 2011, Revised Selected Papers (Lecture Notes in Computer Science)*, Peeter Laud (Ed.), Vol. 7161. Springer, 31–46. https://doi.org/10.1007/978-3-642-29615-4_4

[6] Mauro Conti, Luigi V. Mancini, Riccardo Spolaor, and Nino Vincenzo Verde. 2015. Can't You Hear Me Knocking: Identification of User Actions on Android Apps via Traffic Analysis. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy, CODASPY 2015, San Antonio, TX, USA, March 2-4, 2015*, Jaehong Park and Anna Cinzia Squicciarini (Eds.). ACM, 297–304. https://doi.org/10.1145/2699026.2699119

[7] Tim Cooijmans, Joeri de Ruiter, and Erik Poll. 2014. Analysis of secure key storage solutions on Android. In *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*. ACM, 11–20.

[8] Rob Coombs. 2015. Securing the future of authentication with ARM TrustZone-based trusted execution environment and Fast Identity Online (FIDO). *ARM White Paper* (2015).

[9] Shuaifu Dai, Alok Tongaonkar, Xiaoyin Wang, Antonio Nucci, and Dawn Song. 2013. NetworkProfiler: Towards automatic fingerprinting of Android apps. In *Proceedings of the IEEE INFOCOM 2013, Turin, Italy, April 14-19, 2013*. IEEE, 809–817. https://doi.org/10.1109/INFCOM.2013.6566868

[10] Alexandra Dmitrienko, Christopher Liebchen, Christian Rossow, and Ahmad-Reza Sadeghi. 2014. SECURITY ANALYSIS OF MOBILE TWO-FACTOR AUTHENTICATION SCHEMES. *Intel Technology Journal* 18, 4 (2014).

[11] N Harini, TR Padmanabhan, et al. 2013. 2CAuth: A new two factor authentication scheme using QR-code. *International Journal of Engineering and Technology* 5, 2 (2013), 1087–1094.

[12] Christian Holz and Frank R. Bentley. 2016. On-Demand Biometrics: Fast Cross-Device Authentication. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, San Jose, CA, USA, May 7-12, 2016*, Jofish Kaye, Allison Druin, Cliff Lampe, Dan Morris, and Juan Pablo Hourcade (Eds.). ACM, 3761–3766. https://doi.org/10.1145/2858036.2858139

[13] Ronan Loftus and Marwin Baumann. 2017. Android 7 File Based Encryption and the Attacks Against It. (2017).

[14] Sebastian Schrittwieser, Peter Frühwirt, Peter Kieseberg, Manuel Leithner, Martin Mulazzani, Markus Huber, and Edgar R Weippl. 2012. Guess Who's Texting You? Evaluating the Security of Smartphone Messaging Applications.. In *NDSS*.

[15] Tim Stöber, Mario Frank, Jens B. Schmitt, and Ivan Martinovic. 2013. Who do you sync you are?: smartphone fingerprinting via application behaviour. In *Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks, WISEC'13, Budapest, Hungary, April 17-19, 2013*, Levente Buttyán, Ahmad-Reza Sadeghi, and Marco Gruteser (Eds.). ACM, 7–12. https://doi.org/10.1145/2462096.2462099