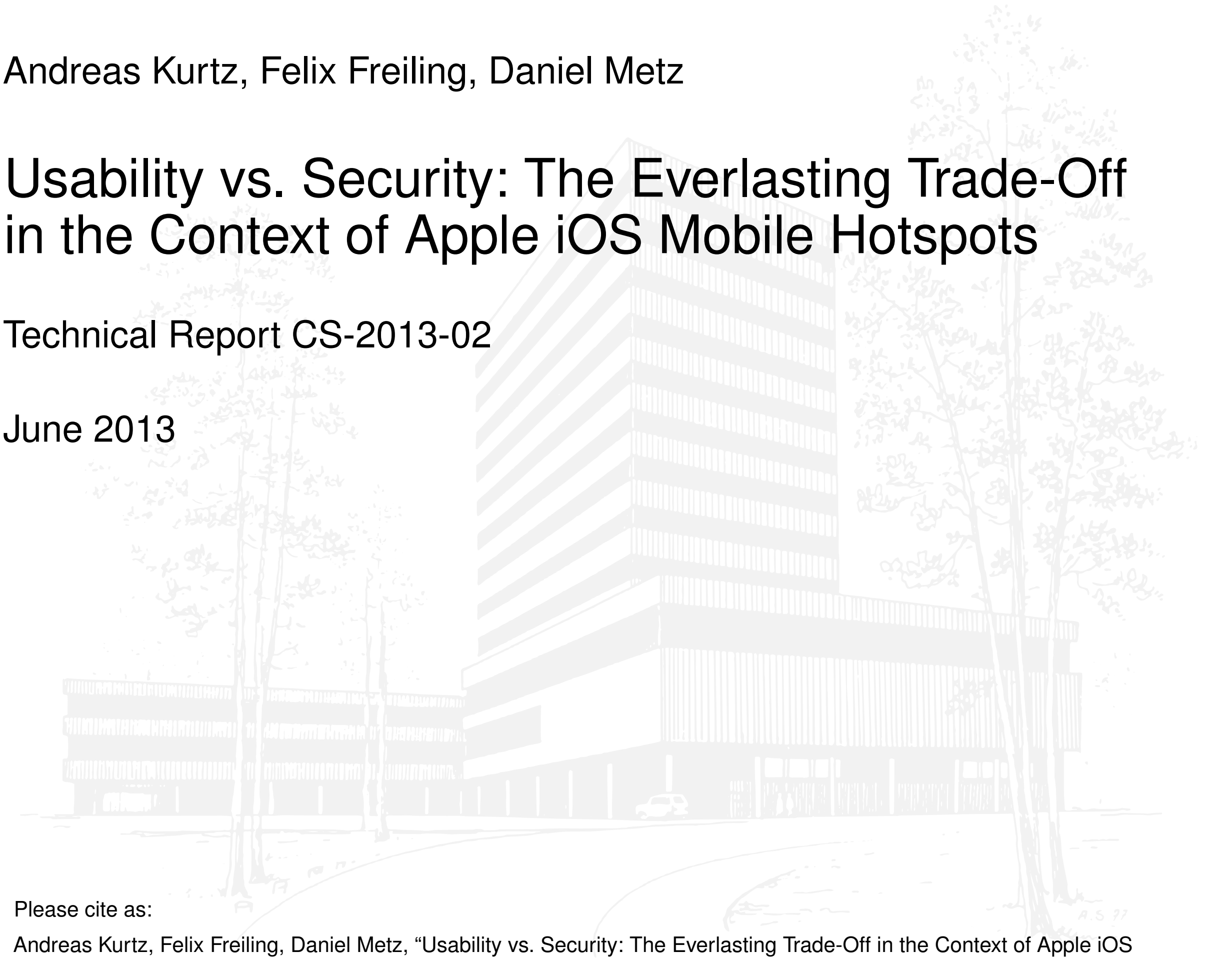
Department Informatik

Technical Reports / ISSN 2191-5008



Andreas Kurtz, Felix Freiling, Daniel Metz

Usability vs. Security: The Everlasting Trade-Off in the Context of Apple iOS Mobile Hotspots

Technical Report CS-2013-02

June 2013

Please cite as:
Andreas Kurtz, Felix Freiling, Daniel Metz, “Usability vs. Security: The Everlasting Trade-Off in the Context of Apple iOS Mobile Hotspots,” University of Erlangen, Dept. of Computer Science, Technical Reports, CS-2013-02, June 2013.

Usability vs. Security: The Everlasting Trade-Off in the Context of Apple iOS Mobile Hotspots

Andreas Kurtz, Felix Freiling, Daniel Metz
Dept. of Computer Science, Friedrich-Alexander-University, Erlangen, Germany
{andreas.kurtz,felix.freiling}@cs.fau.de, daniel.metz@studium.fau.de

Abstract—**Passwords have to be secure and usable at the same time, a trade-off that is long known. There are many approaches to avoid this trade-off, e.g., to advice users on generating strong passwords and to reject user passwords that are weak. The same usability/security trade-off arises in scenarios where passwords are generated by machines but exchanged by humans, as is the case in *pre-shared key* (PSK) authentication. We investigate this trade-off by analyzing the PSK authentication method used by Apple iOS to set up a secure WPA2 connection when using an iPhone as a Wi-Fi mobile hotspot. We show that Apple iOS generates weak default passwords which makes the mobile hotspot feature of Apple iOS susceptible to brute force attacks on the WPA2 handshake. More precisely, we observed that the generation of default passwords is based on a word list, of which only 1.842 entries are taken into consideration. In addition, the process of selecting words from that word list is not random at all, resulting in a skewed frequency distribution and the possibility to compromise a hotspot connection in less than 50 seconds. Spot tests show that other mobile platforms are also affected by similar problems. We conclude that more care should be taken to create secure passwords even in PSK scenarios.**

I. Introduction

It is well known that security is severely influenced by the usability of security enforcing mechanisms. One particularly striking example is the use of secure passwords for authentication since weak passwords are a major entry point for attackers.

The main reason why password based authentication schemes are often easy to break is a result of human memory limitations. As it is well-known in cognitive psychology [1], the capacity of the human brain is limited to around seven (plus or minus two) items that can be hold in memory continuously. In addition, chunks that are familiar to a human, can be remembered more easily than any random fragments. This is why password based authentication always involves a trade-off: strong passwords that are difficult to crack, can hardly be remembered.

This trade-off is even an issue in *mutual* authentication using shared keys. A prominent example is setting up a secure wireless connection between two devices using a *pre-shared key* (PSK). A common scenario for PSK authentication is *Internet tethering* where an existing Internet connection is shared by turning a smartphone into a portable Wi-Fi hotspot. Thus, electronic devices in the vicinity of the hotspot can connect to the Internet using the cellular data connection of the smartphone. This option is particularly popular, because mobile hotspots are most often used during traveling or business trips. Using the Wi-Fi hotspot, users are able to conveniently connect their computers to the Internet, avoiding the need to carry cumbersome cables. The wireless connection is secured by entering the same passphrase on all devices.

While previously users were allowed to use any passphrase in PSK setup, today systems propose passwords that should be both easily memorizable and secure. A standard way to generate such passwords is to combine words from a list of sufficient size with random numbers in a suitable way. While this can increase the entropy of the generated passphrase in theory, it is unclear how secure this approach is in practice.

In this report, we study the security of PSK based authentication in the context of the mobile hotspot feature of Apple iOS version 6 and below. We show that the generation of default passwords in iOS is flawed. We describe an attack to compromise Apple iOS mobile hotspots within seconds to underline the practical impact of the problem. Consequently, an attacker is able to abuse the existing Internet connection or attack any connected notebook from within the trusted local network. Furthermore, even services running on the smart device disclosing personal data might be accessed via the hotspot connection and thus violating the privacy of a smart device owner. We believe that other mobile platforms are also affected by similar problems and therefore conclude that more care should be taken to create secure passwords in PSK scenarios.

A. Related Work

Within a few studies on password based authentication, system users were surveyed about their experiences in dealing with passwords [2], [3]. Thereby, the authors have identified a considerable number of usability issues with password mechanisms and concluded that the primary impact on password design is memorability. Based on these results, Yan et al. [4] investigated how to help users to choose strong passwords. Therefore, they performed a controlled trial on the effects of giving users different kinds of advice. They have also confirmed that users have difficulties remembering random passwords.

The literature on breaking WPA/WPA2 protected wireless networks is relatively sparse. In 2008, Beck and Tews described the first practical attack on WPA protected networks, besides launching a dictionary attack when a weak PSK is used [5]. This attack has been improved by Tomcscany and Lueg [6] by using not only data gathered from the 4-way handshake but also some known-plaintexts. Although these weaknesses have demonstrated some minor weaknesses in the WPA/WPA2 protocol, the most practical way of breaking into a WPA secured wireless network today is still attacking the PSK.

B. Contributions

Within this report we make the following contributions:

- We discuss the security issues that arise from the usability/security trade-off in PSK authentication.
- We reveal serious security defects in the procedure of generating default passwords in the mobile hotspot feature of Apple iOS version 6 and below. We describe an attack to compromise Apple iOS mobile hotspots within seconds and introduce the iOS app *Hotspot Cracker* to assist those attacks.
- We give suggestions on how to remedy this problem and how to generate passwords that are both usable and secure.

C. Roadmap

This report is structured as follows: In Section II, we explain the idea behind our attack and provide some relevant background information. Furthermore, we outline the additional attack surface induced by mobile hotspot features and discuss possible risks. The results and attack details to compromise Apple iOS based mobile hotspots are presented in Section III. We conclude and discuss possible mitigations in Section V.

II. Background

A. Wireless Security

Wireless networks provided by mobile hotspot solutions are usually protected by security mechanisms specified within the IEEE 802.11i [7] standard, which is widely recognized as *Wi-Fi Protected Access version 2* (WPA2). WPA2 supports two different authentication mechanisms, either using a RADIUS server or a shared key. In the context of mobile hotspots, authentication and encryption is always based on a passphrase. This passphrase is used to derive a pre-shared key (PSK) using the *Password-Based Key Derivation Function 2* (PBKDF2) [8]. However, the PSK itself is never used for encryption purposes. Instead, the PSK is used within a 4-way handshake to derive a hierarchy of temporary keys, which are used for encryption and integrity checking purposes. In more detail, during authentication the PSK is used as a so-called *Pairwise Master Key* (PMK). This PMK itself is never used for data encryption, but to derive a temporary encryption key, the *Pairwise Transient Key* (PTK). This key encloses several other temporary keys which are used for data encryption and for message authentication purposes. It should be noted that all generated keys are only valid for the lifetime of a single session and that generation of those keys only relies on the PSK. This implies that the security level of the whole mobile hotspot depends on the quality of the passphrase.

Unlike the outdated security algorithm *Wired Equivalent Privacy* (WEP), which suffers from subtle statistical flaws and is known to be broken [9]–[12], the only practical way to break into WPA2 protected networks so far, is by attacking the PSK based authentication mechanism. An attacker may attempt to discover a PSK by systematically trying every possible combination of letters, numbers and symbols until a correct passphrase is identified (brute force). For this, an attacker needs to capture a 4-way handshake between a Wi-Fi enabled device and the mobile hotspot. Afterwards, brute force or targeted dictionary attacks can be conducted to determine the PSK within offline computations.

B. Apple iOS

iOS (previously iPhone OS) is an operating system used on mobile devices by Apple. iOS is derived from Apple's desktop operating system OS X, which is mostly based on Darwin. Darwin again is an open-source operating system developed by Apple which consists of several different software parts including BSD. The core of the Darwin operating system is based on XNU, a hybrid kernel that combines various parts of BSD and the Mach microkernel. While some components of the iOS kernel are open-source due to historical reasons, the main parts of the iOS operating system are closed and information on it is not publicly available.

The functionality of the iOS operating system can be extended by applications (so-called apps), that can be build using a software development kit (SDK) based on Objective-C. In order to interact with the operating system, iOS provides an extensive application programming interface (API) organized in several different abstraction layers: the Core OS layer, the Core Services layer, the Media layer, and the Cocoa Touch layer. While lower layers provide fundamental system services on which all apps rely on, the higher-level layers provide object-oriented abstractions for lower-level constructs [13].

Most of the iOS system interfaces are provided in special packages, so-called frameworks. A framework consists of a dynamic shared library and corresponding resources (such as header files). Frameworks provide the relevant interfaces needed to build software for the iOS platform. Basically, there are two types of frameworks: *public frameworks* and *private frameworks*. Public frameworks are those recommended by Apple to built third-party apps. Private frameworks are intended to be used only by iOS itself within its system apps.

In order to interact with the underlying kernel, the IPC mechanism of the Mach kernel is used. This mechanism is different from monolithic kernels as messages are sent to a specific port provided by the kernel (so-called Mach messages), instead of calling kernel functionality via a system call or a trap.

C. Mobile Hotspot Attack Surface

When it comes to discussing the attack surface of a smart device, its multiple connectivity options are often named first [14], [15]. A smart device provides plenty of wireless interfaces, like cellular radios (GSM/CDMA/LTE), Wi-Fi, Bluetooth, NFC or RFID. These aggregated communication abilities are a unique characteristic of smart devices, compared to ordinary desktop computers, and they increase the attack surface in many different ways [16].

What has been neglected so far, is the ability of a smart device to be turned into a mobile hotspot. The purpose of this is to share a cellular data connection to other Wi-Fi enabled devices. When the hotspot feature is switched on, a software-based wireless access point is started on the smart device. This access point allows other wireless devices to connect using a pre-shared key and forwards data packets to the Internet.

As these mobile hotspots are often used in public and can be accessed by any surrounding devices, some new hotspot-specific threats are arising, which are discussed in the following.

1) Abuse of the Internet Connection: If a mobile hotspot is compromised, an attacker will instantly gain access to the existing Internet connection. This poses a special risk, as the registered smartphone holder is responsible for the exchanged data [17]. If a mobile hotspot is used to conduct any illegal activities by a malicious party, a remote compromise can hardly be proven, as no relevant log files are recorded. Furthermore, most data plans are subjected to wide restrictions: If a certain data volume is exceeded, the bandwidth might either be reduced until the end of the month or extra fees might be charged.

2) Exposed Services: Access to a mobile hotspot also results in access to services running on a smart device. There are different services, which might be exposed:

In the recent past, it became quite popular to transfer files from a computer to a smart device, in order to access them during travel. To make this procedure more convenient, dedicated apps are often used to turn a smart device into a wireless flash drive (like e.g. AirDrive HD [18]). These apps usually provide services to exchange files over the air using a web browser, without requiring any cable-based synchronization. Therefore, these file transfer apps spawn a HTTP service, providing a web-based interface to upload and manage personal data files. As these file sharing services are bound to the wireless interface, they can also be accessed through a mobile hotspot connection. Consequently, any malicious party compromising a mobile hotspot might also have access to those stored files.

Furthermore, even system services are accessible through a hotspot connection. During the process of removing the restrictions of a mobile operating system in order to install software not authorized by the vendor (iOS Jailbreaking, Android Rooting), often additional services are installed. For instance, many jailbroken iOS devices provide remote access via an unapproved Secure Shell daemon (SSHD). As the well-known factory default password (*alpine* [19]) for the root user remains often unchanged, a smart device might be compromised through a mobile hotspot connection.

Furthermore, not only the smart device itself, but also services running on the connected computers might be attacked from within the trusted local network. If any of the connected computers provide file-sharing services, or have known vulnerabilities within outdated operating system services, personal data might be compromised.

3) Eavesdropping: Finally, an attacker might intercept messages passing between connected devices and the mobile hotspot using a man-in-the-middle attack. As a cellular data connection is often considered more trustworthy than any foreign wireless network connection, mobile hotspot users might underestimate the risk of eavesdropping.

III. The PSK Mechanism of Apple iOS Hotspots

Within this section we describe our results of reverse engineering Apple iOS based mobile hotspots. First, we found out that hotspot default passwords consist of 4 to 6 characters, followed by a four-digit number. As this scheme enables only a very limited number of possible password combinations, the limited search space already makes the mobile hotspot feature of Apple susceptible to brute force attacks on the WPA handshake.

When having a closer look on the default passwords of iOS mobile hotspots, it is noticeable that those passwords do not only comply to the limited scheme introduced above, but are also easily to remember as familiar words are used. After retrieving several hotspot passwords by manually resetting the hotspot settings, we revealed a word list on the Internet, which contained all our collected samples. This list consists of around 52.500 entries and was originated from an open-source Scrabble crossword game [20]. Using this inofficial Scrabble word list within offline dictionary attacks, we already had a 100% success rate of cracking any arbitrary iOS hotspot default password[1]. However, it still took us around 49 minutes to cycle over all 52.500 entries (∼525 mio. permutations according to the 4 digit suffix) by cracking on a AMD Radeon HD 6990 GPU.

[1]There was no evidence indicating that Apple uses this Scrabble word list to generate default passwords. As it will turn out later, both Apple and the game developers might have referred to the same base word list.

Thinking of a real-life scenario where, e.g., business travelers should be attacked during their trip, that time frame would have been excessively too long. For this reason, we aimed to optimize the cracking procedure in order to provide more realistic attacks.

Since our preliminary word list was not precise enough and contained many entries, not complying with the underlying password scheme, reduction of the search space was our primary goal. Therefore, we reverse engineered the relevant parts of the Apple iOS operating system, to extract the exact word list which is used during the iOS setup procedure to create hotspot default passwords.

As most parts of the iOS operating system are proprietary, we combined static and dynamic analysis techniques to reverse engineer the iOS operating system. Therefore, most of the reverse engineering effort was spent on manually analyzing the ARM disassembly of the relevant iOS frameworks. In addition, our static analysis was supplemented by dynamic analysis using the GNU Debugger.

Using those techniques, we determined that the official *Preferences* system app is responsible for generating default hotspot passwords during the first launch. We found out, that every time a new hotspot password is generated an English-language dictionary file is accessed from the file system. Consequently, we monitored all accesses to the file system by intercepting all `open()` system calls to the iOS kernel and analyzed the corresponding backtrace of the method calls that caused this file access. Listing 1 shows an excerpt of the relevant backtrace when the dictionary file is accessed.

Within one of the first steps, the method `generateDefaultPassword()` of the `WifiPasswordController` class triggers the overall initialization (Frame #17 in Listing 1). This method invokes a frontend method of the Apple spell checking service, located in the *UIKit* framework. The method `suggestWordInLanguage()` of the class `UITextChecker` (Frame #16) is normally used by Apple to enhance the user experience and to predict user input. Whenever a user enters some text, words are suggested appearing above the keyboard based on the letters a user already typed in. This method is re-used to generate easy pronounceable hotspot passwords that can be easily remembered. Next, the *ProofReader* framework, a private framework and the core of the iOS spell checking service is utilized to pick a single word in a specific length range from this English-language dictionary file (Frames #15 to #13). According to the relevant disassembly output, only words in the length of 4 to 6 characters are selected.

```
#0  0x36b26dc4 in open ()
...
#13 0x3076bd1e in -[AppleSpell
       databaseConnectionForLanguage:] ()
#14 0x307d7bf4 in -[AppleSpell(Guessing)
        spellServer:
       suggestWordWithMinimumLength:
       maximumLength:language:] ()
#15 0x3077be12 in -[AppleSpell
       spellServer:
       suggestWordWithLengthInRange:
       language:] ()
#16 0x353fc342 in -[UITextChecker
       suggestWordInLanguage:] ()
#17 0x33b1e874 in +[Wi-
       FiPasswordController
       generateDefaultPassword] ()

...
0x2fe5d6a4: "/System/Library/
       PrivateFrameworks/ProofReader.
       framework/English.lproj/Dictionary.
       dat"
```

Listing 1: Stack backtrace while debugging the Apple iOS Preferences app revealing that an English-language spell checking dictionary is used to generate hotspot default passwords.

During static analysis, we revealed that a four digit random number is appended to the word picked from the dictionary. Finally, the private *MobileWiFi* framework is instrumented to deliver the newly generated default password to the Wi-Fi daemon (*wifid*) using a Mach message call. The wifid is responsible for managing the Wi-Fi based connectivity and stores the default password into the iOS keychain (*Account: _AppleWi-FiInternetTetheringSSID_*). The password is retrieved from the keychain, everytime the mobile hotspot is switched on.

Although, we revealed the exact location of the dictionary file, we were still not able to acquire the relevant entries. As it turned out, the dictionary used by Apple's spell checking service is highly encoded and copyrighted by Lernout & Hauspie Speech Products, a former provider of speech and language technology products. To not violate any copyright laws or licenses, we avoided to reverse engineer the dictionary format. Instead, we preferred a dynamic approach to retrieve qualified words from the dictionary and to build up a hotspot cracking word list. We repeatedly invoked the *suggestWordInLanguage()* method of the spell checking framework (100 mio. times) and stored the derived words in a database for further analysis.

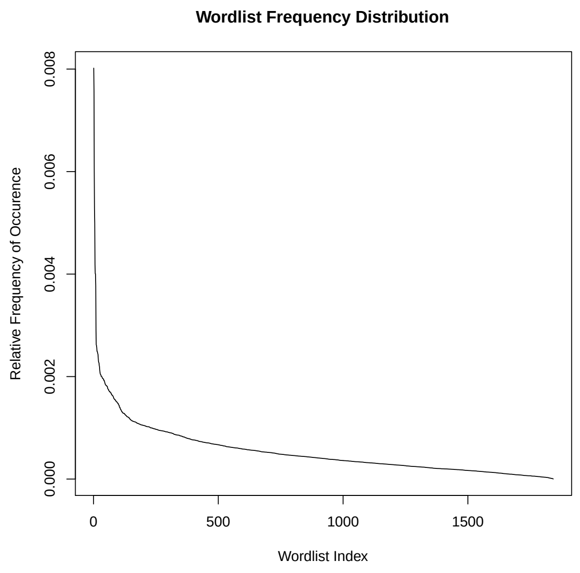


Fig. 1: Skewed Frequency Distribution: Words from the Top 10 are ten times more likely selected to be used as a default password than any other words.

As it turned out, the list of retrieved passwords was stabilized after around 250.000 invocations of the method above and only 1.842 different entries of that dictionary are taken into consideration. Consequently, any default password used within an arbitrary iOS mobile hotspot, is based on one of these 1.842 different words. This fact reduced the search space of our initial brute force attack by more than 96% and thus increased the overall cracking speed significantly. Compared to ∼525 mio. permutations at the beginning, the total amount of possible candidates which have to be tried out during a dictionary attack was reduced to ∼18.4 mio.

In addition, the process of selecting words from the spell checking dictionary was found to be not random at all, resulting in a skewed frequency distribution (see Figure 1). Although we did not reverse engineer the dictionary mechanism in detail, it seems that these abnormalities are caused by inner optimizations in the processing of the dictionary format.

Table I lists the most commonly used hotspot passwords ordered by their relative frequency of occurrence. Our analysis has shown that words from this Top 10

Top 10	Word	RF
1.	suave	0.80%
2.	subbed	0.76%
3.	headed	0.61%
4.	head	0.53%
5.	header	0.50%
6.	coal	0.41%
7.	ohms	0.40%
8.	coach	0.40%
9.	reach	0.38%
10.	macaws	0.29%

TABLE I: Top 10 of the most commonly used iOS hotspot passwords ordered by relative frequency (RF) of occurrence.

list are ten times more likely to be selected as a default password than other words. This further speeds up the attack, as we process our word list according to the relative frequency of occurrence of a single word.

IV. Attacking iOS PSK Authentication

Another well-known way to improve brute force attacks, is to speed up the actual cracking process and to increase the number of guesses per second by using distributed cracking methods. Instead of using only a single GPU, we balanced the cracking load on several GPUs and compared the cracking times of each cluster. As shown in Table II, a GPU cluster composed of four AMD Radeon HD 7970 was able to cycle through around 390.000 guesses per second. Using our reduced and optimized search space, which consists of only around 18.5 million ($= 1.842 \cdot 10^4$) possible candidates, such a GPU cluster will crack an arbitrary iOS hotspot default password in less than 50 seconds.

A. Practical Attack Scenarios

To attack arbitrary hotspot users, an attacker simply has to monitor the traffic and to wait for a wireless client to connect to a mobile hotspot. To accelerate the process, or to perform a targeted attack against a specific hotspot, an attacker might deauthenticate an existing wireless client. By forcing clients to reauthenticate, the probability of capturing a WPA handshake is increased. Within both attack types, iOS based hotspot targets can be identified by their vendor-specific MAC addresses.

The whole process of *(1)* identifying iOS targets, *(2)* deauthenticating wireless clients, *(3)* capturing WPA handshakes and *(4)* cracking the hotspot default password can be automated easily, using freely available tools [21], [22]. To automate the process of word list generation, we built the iOS app *Hotspot Cracker* [23]. This app assists in generating an iOS hotspot cracking word list, which might be used in subsequent attacks on other hotspot users. The app also gives explanations and hints on how to crack a captured WPA handshake using well-known password crackers. Future releases might also automate the process of capturing and cracking hotspot passwords. As computing power on smart devices is limited, one solution is to involve online password cracking services like CloudCracker [24], to crack hotspot passwords on-the-fly.

B. Brief Comparison to other Platforms

Other mobile platforms might be affected by these deficits as well. Although, we did not analyze other platforms in detail, spot-checks have revealed that default passwords in *Windows Phone 8* consist of only 8-digit numbers. As this results in a search space of 10^8 candidates, attacks on Windows-based hotspot passwords might be practicable. Moreover, while the official version of *Android* generates strong passwords[2], some vendors modified the Wi-Fi related components utilized in their devices and weakened the algorithm of generating default passwords. For instance, some Android-based models of the smartphone and tablet manufacturer HTC are even shipped with constant default passwords consisting of a static string (*1234567890*) [26]. However, future studies will be necessary to evaluate the security level of mobile hotspots on other platforms in more detail.

V. Countermeasures and Conclusion

As the mobile hotspot feature is probably most often used while being on travel, on conferences, or hotel stays, an attacker will only have a limited amount of time to succeed in breaking into a mobile hotspot. Therefore, a very limited cracking time frame is the main requirement for such an attack to be practically relevant. Taking our optimizations into consideration, we are now able to show that it is possible for an attacker to reveal a default password of an arbitrary iOS hotspot user within seconds. For that to happen, an attacker only needs to capture a WPA2 authentication

[2]The method `setDefaultApConfiguration()` within the Android `WifiApConfigStore` class [25] is responsible for creating random WPA2 passwords based on Java's universally unique identifiers (*UUID*).

# GPUs	Hardware	Cycles per Second	ACT
2x	Nvidia Tesla C2075	46.600	3m 18s
1x	AMD Radeon HD 6990	180.000	52s
4x	AMD Radeon HD 7970	390.000	24s

TABLE II: Average cracking time (ACT) of an arbitrary iOS hotspot default password using different GPU clusters.

handshake and to crack the pre-shared key using our optimized dictionary.

As it is always a good advice to replace initial default passwords by user-defined strong and secure passwords, this becomes particular relevant on mobile hotspots passwords. Therefore, users of mobile hotspots, especially of iOS-based mobile hotspots, are advised to change their passwords. In addition, some mobile platforms (like Apple iOS) display the number of connected clients on the lock screen. Therefore, it is a good advice to periodically check that screen for any conspicuous activity. Finally, hotspot capabilities of smart devices should be switched off every time when they are no longer needed, to keep the overall attack surface as minimal as possible.

Vendors of mobile hotspot solutions should improve their way of generating initial default passwords. System-generated passwords should be reasonably long and should use a reasonably large character set. Consequently, hotspot passwords should be composed of completely random sequences of letters, numbers and special characters. It can be neglected that increased randomness could have a negative impact on the memorability of the passwords. Particularly, in the context of mobile hotspots there is no need to create easily memorizable passwords. After a device has been paired once by typing out the displayed hotspot password, the entered credentials are usually cached within the associating device and are re-used within subsequent connections.

Summing up, the results of our analysis have shown that the mobile hotspot feature of smart devices increases the attack surface in several ways. As the default password of an arbitrary iOS hotspot user can be revealed within seconds, attacks on mobile hotspots might have been underestimated in the past and might be an attractive target in the future.

Acknowledgment

The authors would like to thank Tobias Klein (*NESO Security Labs GmbH*), Simon Streicher (*Heilbronn University*) and Markus Troßbach (*University of Heidelberg*).

References

[1] G. Miller, "The magical number seven, plus or minus two: Some limits on our capacity for processing information," *The psychological review*, vol. 63, pp. 81–97, 1956.

[2] A. Adams and M. A. Sasse, "Users are not the enemy," *Communications of the ACM*, vol. 42, no. 12, pp. 40–46, 1999.

[3] M. A. Sasse, S. Brostoff, and D. Weirich, "Transforming the weakest link - a human/computer interaction approach to usable and effective security," *BT technology journal*, vol. 19, no. 3, pp. 122–131, 2001.

[4] J. Yan, A. Blackwell, R. Anderson, and A. Grant, "Password memorability and security: Empirical results," *Security & Privacy, IEEE*, vol. 2, no. 5, pp. 25–31, 2004.

[5] E. Tews and M. Beck, "Practical attacks against WEP and WPA," in *Proceedings of the second ACM conference on Wireless network security*. ACM, 2009, pp. 79–86.

[6] D. P. Tomcsanyi and L. Lueg, "CCMP known-plain-text attack," 2010.

[7] IEEE, *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, Amendment 6: Medium Access Control (MAC) Security Enhancements (IEEE Std 802.11i-2004)*, Institute of Electrical and Electronics Engineers, Inc.

[8] B. Kaliski, *PKCS #5: Password-Based Cryptography Specification Version 2.0(IETF RFC 2898)*, The Internet Society, Sep. 2000.

[9] S. R. Fluhrer, I. Mantin, and A. Shamir, "Weaknesses in the Key Scheduling Algorithm of RC4," in *Revised Papers from the 8th Annual International Workshop on Selected Areas in Cryptography*, ser. SAC '01. London, UK, UK: Springer-Verlag, 2001, pp. 1–24.

[10] N. Cam-Winget, R. Housley, D. Wagner, and J. Walker, "Security flaws in 802.11 data link protocols," *Communications of the ACM*, vol. 46, no. 5, pp. 35–39, 2003.

[11] A. Klein, "Attacks on the rc4 stream cipher," *Designs, Codes and Cryptography*, vol. 48, no. 3, pp. 269–286, 2008.

[12] A. Bittau, M. Handley, and J. Lackey, "The final nail in WEP's coffin," in *Security and Privacy, 2006 IEEE Symposium on*. IEEE, 2006, pp. 15–pp.

[13] iOS Technology Overview. [Online]. Available: http://developer.apple.com/library/ios/#documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html

[14] C. Guo, H. J. Wang, and W. Zhu, "Smart-phone attacks and defenses," in *HotNets III*, 2004.

[15] C. Miller, "Mobile attacks and defense," *Security & Privacy, IEEE*, vol. 9, no. 4, pp. 68–70, 2011.

[16] M. Becher, F. C. Freiling, J. Hoffmann, T. Holz, S. Uellenbeck, and C. Wolf, "Mobile Security Catching Up?Revealing the Nuts and Bolts of the Security of Mobile Devices," 2011.

[17] R. Hale, “Wi-Fi Liability: Potential Legal Risks in Accessing and Operating Wireless Internet,” *Santa Clara Computer and High Technology Law Journal*, vol. 21, p. 543, 2005.

[18] iOS App AirDrive HD, Version 1.6.0. [Online]. Available: https://itunes.apple.com/us/app/airdrive-hd-wireless-flash/id484724740?mt=8

[19] iOS Default Root Password. [Online]. Available: http://cydia.saurik.com/password.html

[20] Scrabble Word List. [Online]. Available: http://www.badgehungry.com/scrabble-word-list/

[21] J. Steube. Hashcat Advanced Password Recovery. [Online]. Available: http://hashcat.net/oclhashcat-plus/

[22] Aircrack-ng. [Online]. Available: http://www.aircrack-ng.org/

[23] iOS App Hotspot Cracker. [Online]. Available: http://www1.cs.fau.de/hotspot/

[24] M. Marlinspike. (2013, Apr.) CloudCracker, an online password cracking service. [Online]. Available: https://www.cloudcracker.com

[25] Google, Android Developer Documentation. WifiApConfigStore class. [Online]. Available: https://android.googlesource.com/platform/frameworks/base/+/master/wifi/java/android/net/wifi/WifiApConfigStore.java

[26] Verizon. (2013, Apr.) Support Page on the Mobile HotSpot Settings of some HTC Models. [Online]. Available: http://support.verizonwireless.com/clc/devices/knowledge_base.html?id=35523