Advanced Wi-Fi Attacks Using Commodity Hardware

Mathy Vanhoef
iMinds-DistriNet
KU Leuven
Mathy.Vanhoef@cs.kuleuven.be

Frank Piessens
iMinds-DistriNet
KU Leuven
Frank.Piessens@cs.kuleuven.be

ABSTRACT

We show that low-layer attacks against Wi-Fi can be implemented using user-modifiable firmware. Hence cheap off-the-shelf Wi-Fi dongles can be used carry out advanced attacks. We demonstrate this by implementing five low-layer attacks using open source Atheros firmware. The first attack consists of unfair channel usage, giving the user a higher throughput while reducing that of others. The second attack defeats countermeasures designed to prevent unfair channel usage. The third attack performs continuous jamming, making the channel unusable for other devices. For the fourth attack we implemented a selective jammer, allowing one to jam specific frames already in the air. The fifth is a novel channel-based Man-in-the-Middle (MitM) attack, enabling reliable manipulation of encrypted traffic.

These low-layer attacks facilitate novel attacks against higher-layer protocols. To demonstrate this we show how our MitM attack facilitates attacks against the Temporal Key Integrity Protocol (TKIP) when used as a group cipher. Since a substantial number of networks still use TKIP as their group cipher, this shows that weaknesses in TKIP have a higher impact than previously thought.

1. INTRODUCTION

Wireless networks based on the 802.11 standard have had an enormous success, ranging from use in common households to large scale deployments in critical infrastructures. As new standards push the boundary of transmission speed and functionality, the capabilities of wireless chips have increased accordingly. This opens new possibilities where commodity devices can be used to implement state of the art attacks, previously thought only possible on expensive hardware such as Universal Software Radio Peripherals (USRPs). To demonstrate this we implement several low-layer attacks using off-the-shelf Wi-Fi dongles. In particular we modify the firmware of Atheros AR7010 and AR9271 chips. We conjecture that other devices, which also load user-modifiable firmware after power up, can execute similar attacks.

ACSAC '14, December 08–12, 2014, New Orleans, LA, USA.

ACM 978-1-4503-3005-3/14/12 ...$15.00.
http://dx.doi.org/10.1145/2664243.2664260 .

Our first modifications give the client an unfair share of the bandwidth. We use this to experimentally explore the behaviour of selfish stations. Based on this, to the best of our knowledge, we are the first to suggest that contending selfish stations will exploit the capture effect to increase throughput. This is the phenomenon where, in case of a collision, the frame with the strongest signal and lowest bitrate is received correctly [13]. Surprisingly, contending selfish stations end up lowering their bitrate to obtain a higher throughput. We also bypass systems designed to prevent selfish behaviour.

We continue by turning our dongle into a continuous and selective jammer. The former endlessly transmits noise, while the latter jams specific frames already in the air. Previously, selective jamming required either a costly USRP setup [7], or a WiFi adapter which allows modifications to the microcode handling medium access and de/encoding operations [4]. Though it shows selective jamming is possible, the URSP setup is non-trivial and costs more than $3600. Additionally, being able to change medium access algorithms in commodity devices is rare in practise. We show that these jamming attacks can be implemented using off-the-shelf USB dongles (costing only 15$) without requiring low-level microcode access. It is surprising this is possible by using only the default (and limited) API of the wireless chip (i.e., without modifying its medium access algorithms).

With as goal to enable (or facilitate) attacks on higher-layer protocols, we also present a channel-based MitM attack against any type of encrypted network. Our goal is not to decrypt traffic, but to be able to reliably intercept and manipulate it. Normally obtaining a MitM position in an encrypted network is not trivial, since the station and Access Point (AP) verify each other's MAC address. Therefore an attacker cannot set up a rogue access point with a different MAC address to subsequently intercept and forward all traffic. We demonstrate that cloning the AP on another channel bypasses the MAC address checks. Customizable firmware makes the implementation easier and more efficient, in particular when targeting multiple stations simultaneously.

Finally we use the MitM position to attack TKIP when used as a group cipher, and show how to decrease the execution time of the attack by targeting multiple stations simultaneously. Since most routers by default use TKIP as their group cipher for backwards compatibility, a significant amount of networks are vulnerable to our attack.

Our selective jammer, channel MitM, and TKIP attacks are available for download [1]. The unfair channel usage and continuous jammer can disrupt network activity without usable countermeasures being available. Therefore we do not

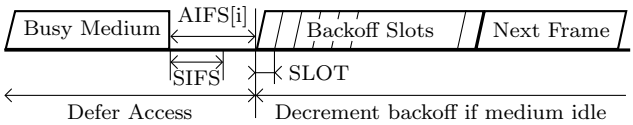

Figure 1: EDCA Timing Relations.

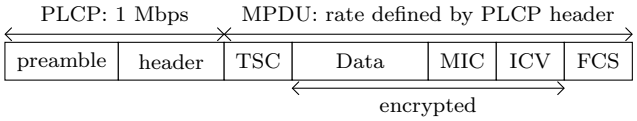

Figure 2: Simplified format of 802.11b TKIP frames.

release them. To summarize, our main contributions are:

- We study (contending) selfish stations using off-the-shelf Wi-Fi dongles, taking into account capture affect, and bypass systems designed to detect selfish users.
- We show how continuous and selective jamming can be implemented using off-the-shelf Wi-Fi dongles.
- We present a channel-based Man-in-the-Middle attack against encrypted networks, enabling reliable interception and manipulation of all encrypted traffic.
- We attack TKIP when used as a group cipher, and additionally devise a technique to decrease the execution time by targeting multiple stations simultaneously.

The remainder of this paper is organized as follows. Section 2 explains the relevant aspects of the 802.11 standard. In Section 3 we analyse, implement, and measure the impact of selfish stations. Section 4 shows how continuous and selective jamming can be implemented on commodity devices. A novel channel-based Man-in-the-Middle attack is presented in Section 5, and in Section 6 we use this to attack TKIP when used as a group cipher. Finally, we summarise related work in Section 7 and conclude in Section 8.

2. THE 802.11 STANDARD

In this section we present the basics of the 802.11 MAC and physical (PHY) layer [11], explain the TKIP encryption protocol, and introduce the `ath9k_htc` firmware.

2.1 Medium Access Control (MAC)

The MAC service is responsible for exchanging MAC Protocol Data Units (MPDUs) using carrier sense multiple access with collision avoidance (CSMA/CA). A station wishing to transmit first monitors the medium for a given Inter Frame Space (IFS), using a carrier sense mechanism. If the medium is busy the station defers its transmission using a random backoff procedure. The original standard offered the Distributed Coordination Function (DCF) to control this process. The 802.11e amendment extends DCF with Quality of Service (QoS) enhancements, resulting in enhanced distributed channel access (EDCA). With EDCA four different Access Classes (AC) are defined, allowing the prioritization of traffic. For example, the access class of voice traffic has a higher priority than that of background traffic. We use the terms QoS channel and priority as synonyms of AC.

The time a station must wait before it can transmit depends on several parameters and inter frame spaces (see Figure 1). The shortest is the Short Interframe Space (*SIFS*) and is the time between successfully receiving a frame and sending an acknowledgement (ACK). Other frames must wait longer than *SIFS*, giving ACKs the highest priority. When a station wants to transmit a data frame of access class AC it first monitors the medium for a period equal to

$$AIFS[AC] = SIFS + AIFSN[AC] \cdot SLOT \qquad (1)$$

where $SLOT$ is the duration of one slot interval, and with $AIFSN$ dependent on the access class. If the medium is idle during this time the station can transmit immediately. In case the medium was busy the contention window $CW[AC]$ is set to $CW_{min}[AC]$ and the random backoff procedure is initiated. This procedure initializes the backoff counter to a value uniformly distributed over the interval $[0, CW[AC]]$. Once the medium has been idle for $AIFS[AC]$, the backoff counter is decremented for each slot where the medium is idle. When the counter is zero the station transmits the frame. Since AC is clear from context it will no longer be explicitly included. The values assigned to $AIFSN$, CW_{min}, and CW_{max} determine the priority of an access class, and are advertised by the AP in the EDCA parameter set element in beacons and probe responses. Beacon frames are periodically transmitted to advertise the presence of a network, while probe requests and responses are used to actively seek networks. The value of *SIFS* and *SLOT* depend on the physical layer being used (see Section 2.2).

If the Frame Check Sequence (FCS) at the receiver is correct it will wait *SIFS* time and send an ACK. If the sender receives an ACK it resets CW to CW_{min} and enters a post-backoff stage by setting the backoff counter to a value uniformly distributed over the interval $[0, CW]$. When the FCS is wrong the frame is discarded. When the sender receives no ACK it retransmits the frame by updating CW to $2 \cdot CW + 1$ and repeating the backoff procedure. A frame is retransmitted at most 7 times, and CW is limited by CW_{max}.

Assuming all stations cooperate, analytical models of DCF show that it provides fairness [5]. Hence identical access classes in EDCA also provide this fairness. However, in practice stations can act selfishly and increase their throughput at the cost of others [25, 22]. In Sect. 3 we will study in detail how selfish stations can increase their throughput.

2.2 Physical Layer (PHY)

Wi-Fi can operate in the 2.4 and 5 GHz bands. A channel number defines the center frequency on which the radio operates. Senders can choose between multiple transmission rates: low rates are used for bad connections, and high for good connections. For backwards compatibility beacons and probes are generally sent using the lowest rate of 1 Mbps.

When an MPDU is transmitted a Physical Layer Convergence Protocol (PLCP) preamble and header is added (see Figure 2). The preamble allows the receiver to synchronize to the transmission, and the header defines the modulation used for the MPDU. We focus on 802.11g long preamble mode, where (slightly simplified) the overhead of the PLCP is 23 μs, *SIFS* is 10 μs, and *SLOT* is 9 μs [11, §18.3].

An important property of 802.11 radios is their susceptibility to the capture effect, the phenomenon where the strongest frame is received correctly in case of a collision. This goes against the common assumption that both frames are lost. More precisely, a collision results in one of three cases [13]: (1) both frames are lost; (2) if the stronger

frame is received first it will be decoded correctly; or (3) if the stronger frame is received second, yet within the PLCP preamble of the first one, the second frame will be decoded correctly. The capture effect has been observed on different chipsets [13, 8], occurs more frequently at low bitrates [14], and favours nearby stations [13]. In Section 3 we show that selfish stations can abuse it to contend with other stations.

2.3 Temporal Key Integrity Protocol

The Temporal Key Integrity Protocol (TKIP) is an improvement of the broken WEP protocol [27]. It was designed so old WEP-compatible hardware can support TKIP with only firmware updates. Nowadays TKIP is deprecated, and the more secure (AES-)CCMP is recommended. However, networks which support both TKIP and CCMP simultaneously, generally use TKIP as their group cipher. This is done for backward compatibility, so both old TKIP clients and newer CCMP clients can decrypt broadcast data. Since most routers by default allow both TKIP and CCMP, and thus use TKIP as their group cipher, TKIP is still widely used. For example, recently it has been shown that 66% of wireless networks in residential areas in Belgium use TKIP as their group cipher [29]. Note that WPA1 and WPA2 are certifications handed out by the WiFi Alliance. The difference between them is that WPA1 mandates TKIP support and optionally allows CCMP, while the reverse is true for WPA2. We use the term WPA1/2 to refer to both.

When a station connects to a secured network it begins by negotiating session keys using a 4-way handshake. This results in a pairwise transient key (PTK) and a group temporal key (GTK). These keys depend, among other things, on the MAC address of the station and AP. The PTK protects unicast traffic, while the GTK protects broadcast traffic. When using TKIP as the group cipher, the GTK is used to derive a 128-bit encryption key, and a 64-bit Message Integrity Code (MIC) key. Note that a client transmitting a broadcast frame first sends it to the AP (as a unicast frame), after which the AP broadcasts it to all stations using the group cipher. All keys are renewed after a user defined interval, commonly set to 1 hour.

When a station transmits a TKIP protected frame, it first calculates the MIC over the data field using its 64-bit MIC key (see Figure 2). Then it calculates a CRC over the data and MIC, called the Integrity Check Value (ICV). These fields are encrypted using a per packet key derived from the TKIP Sequence Counter (TSC) and 128-bit encryption key. The TSC is a replay counter which is incremented after successfully transmitting a frame. The receiver drops frames with an old TSC or bad ICV. However, if a frame has a valid TSC and ICV, but a wrong MIC, the TKIP countermeasures are activated:

- Clients send a MIC failure report to the AP. Additionally, when a client itself detects two failures within one minute, it will disconnect from the network.
- An AP silently logs the MIC failure. If two failures occur within one minute, it will halt all TKIP traffic for one minute. After this minute clients can reconnect.

Several weaknesses have been discovered in TKIP [27, 29, 18, 28, 10, 17, 21, 20]. The first known attack, and the one we built on, is the Beck and Tews attack [27]. It decrypts a packet byte by byte and requires that QoS is supported, which is the case for most modern devices [18, 29]. The attack works by guessing the last byte of an encrypted packet and then removing this byte. The ICV of the (encrypted) shortened packet can be corrected if the guess was correct. Because TKIP uses a unique TSC for each QoS channel, we can inject the corrected packet on a QoS channel with a lower TSC. If the guess was wrong, the ICV will be wrong, and the receiver drops the packet. However, if the guess was correct, the receiver will respond with a MIC failure. Hence the last byte of a packet can be found by trying all possible values and detecting a correct guess using the MIC failure report. This technique is then recursively applied to the shortened packet. To avoid the TKIP countermeasures at most one byte can be decrypted per minute. In the Beck and Tews attack this technique is used to decrypt an ARP packet. From the decrypted packet, the MIC key for that communication direction can be derived. This attack, and its variations, have previously only been designed to attack unicast traffic [27, 18, 28, 10, 29, 20].

For a more detailed background on TKIP, including practical attacks, we refer the reader to our earlier work [29].

Listing 1: Code which sets EDCA parameters for a specific access class in the ath9k_htc driver.

```
1 int ath_htc_txq_update(...):
2   qi.tqi_aifs = qinfo->tqi_aifs;
3   qi.tqi_cwmin = qinfo->tqi_cwmin / 2; /* XXX */
4   qi.tqi_cwmax = qinfo->tqi_cwmax;
```

2.4 Atheros Firmware Implementation

The open source `ath9k_htc` firmware runs on AR7010 and AR9271 chips, and is sent to the device after power up. The AR7010 is a system-on-chip that generally uses PCIe to connect to an external AR9280 wireless chip, while the AR9271 is a single-chip solution with onboard wireless chip. The AR7010 supports both the 2.4 GHz and 5 GHz band, but can only use its built-in 2x2 MIMO antenna. On the other hand, the AR9271 can use an external antenna, but only supports the 2.4 GHz band.

The wireless chip in both devices is controlled using memory mapped registers. For example, the `AR_QTXDP` register contains a pointer to the transmit queue, and writing to the `AR_Q_TXE` register will start the transmit process.

3. UNFAIR CHANNEL USAGE

In this section we study (contending) selfish stations using off-the-shelf Wi-Fi dongles. This is done by implementing several strategies and measuring the resulting throughput. Finally we bypass system designed to detect selfish users.

3.1 Experimental Setup

In order to implement selfish strategies we modified the `ath9k_htc` driver to get direct control over MAC layer parameters. While doing this we found that the original driver wrongly divided CW_{min} by two, while other parameters were properly configured (see Listing 1 line 3). This leads to an unfair advantage, and motivates other stations to act selfish as well. The driver was patched to properly set CW_{min}.

A Linksys WAG320N with firmware v1.00.08 is used as the AP. It is connected to a HP 8510p running Linux 3.7.2

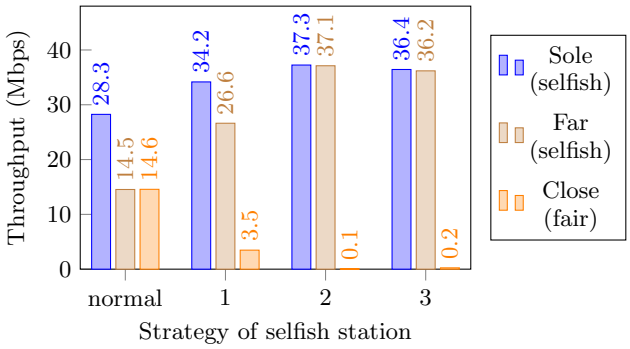

Figure 3: Throughput in case of one selfish station for the strategies in Sect. 3.2. Close and far denote contending stations, with far being the selfish one. Sole represents only one connected station. Standard deviation of all results were below 0.28 Mbps.

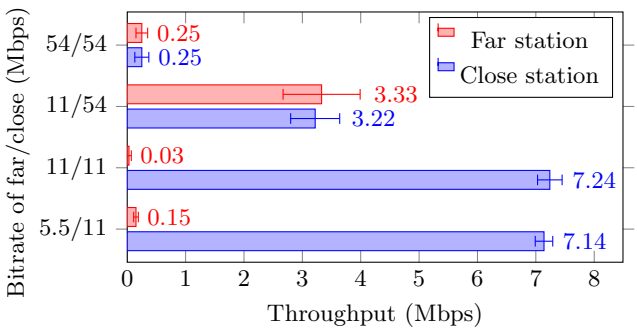

Figure 4: Impact of bitrate on contending selfish stations. Y-axis shows bitrate of far and close station, respectively. Error bars denote standard deviation.

using a gigabit ethernet connection. For the clients we created two identical VMWare Player instances, each having 2GB RAM and running Linux 3.7.2. One client uses a TL-WN722N placed 10 cm from the AP, while the other uses a AWUS036nha placed 1 meter from the AP (close distances reduce packet loss, making measurements more accurate). We refer to them as the close and far station, respectively. Reported results are over 10 runs of iperf 2.0.5 in UDP mode, using the maximum payload of an ethernet frame (1500 bytes minus 28 bytes for the IP and the UDP header), in the form bandwidth $\pm$ standard deviation. We use 802.11g, a channel width of 20 MHz, do not use the RTS or CTS protection mechanism, and do not use encryption. We assume the AP to be trusted and only consider selfish clients.

3.2 One Selfish Station

We study the behaviour a single selfish station in two phases. First when the selfish station is the only (sole) device connect to the network, and then when other fair stations are also connected to the network. For the first phase we let the close station be the only (sole) selfish station connected to the network. We implemented the following selfish strategies to determine whether they are effective or not:

1. Disabling backoff using the `AR_DMISC` register.
2. Setting *AIFSN* to zero in the `AR_DLCL_IFS` register.
3. Decreasing *SIFS* in the `AR_D_GBL_IFS_SIFS` register.

The result of these (incremental) strategies, including normal throughput, are shown in Figure 3 under "Sole (selfish)". Only results where *SIFS* was lowered to 6 are shown (our conclusion is the same for other values). We verify these results theoretically. It takes 218 μs to send 1472 bytes of UDP payload at 54 Mbps (the highest 802.11g bitrate), and 23 μs to send the long PLCP preamble and header. We have an overhead of a 34 bytes MAC header, a 20 byte IP header, a 8 byte UDP header, and a 4 byte FCS. At 54 Mbps this results in a overhead of 33 μs for each frame encapsulating a 1472 byte UDP payload. Since an ACK is 14 bytes and sent at 24 Mbps, it takes 38 μs to transmit when including the LPCP and *SIFS* timeout. Before sending the data frame, we wait on average $SIFS + (AIFSN + \frac{CW_{min}}{2}) \cdot SLOT$. Normally, when *SIFS* is 10 μs, *SLOT* is 9 μs, *AIFSN* is 3, and CW_{min} is 15, this is equal to 104.5μs. Hence it takes on average 393 μs to send 1472 bytes of data, resulting in a theoretical throughput of 29.96 Mbps. For the three selfish strategies we get 36.18 Mbps, 39.45 Mbps, and 40.53 Mbps, respectively. Our results are slightly lower because a beacon is transmitted every 102.4 ms and not all packets arrive successfully. Notice that reducing *SIFS* may negatively impact frame delivery. Hence a sole selfish station will disable backoff and set *AIFSN* to zero, but will not change *SIFS*.

The experiments were repeated when a fair station was also present. Since a malicious station may be unable to move arbitrarily close to the AP, we pick the far station as the selfish one. The results are shown in figure 3 under "Close (fair)" and "Far (selfish)". Again the optimal strategy of the selfish station is to keep the default value of *SIFS*, disable backoff, and set *AIFSN* to zero. We found this strategy works independent of the distances between stations.

3.3 Multiple Selfish Stations

We experimentally investigate the behaviour of contending selfish stations, by letting both the close and far station act selfishly. Because selfish stations generate so much traffic they hinder the arrival of beacons, it was necessary to modify the stations so they would not disconnect when too many beacons were lost. We found two strategies to increase throughput, both with as goal to exploit the capture effect:

1. Reducing the bitrate of transmissions.
2. Increasing transmit power.

These strategies work because, in case of a collision, the receiver tends to decode the frame having the lowest bitrate and the highest signal strength. Since collisions are now desired selfish stations will disable carrier sense. This can be done using the `AR_DIAG_SW` register. To avoid the station from lowering its bitrate after a failed transmission, we force both stations to use a fixed bitrate in each experiment.

First we performed experiments when contending selfish stations reduce their bitrate. We found that stations keep lowering their bitrate until this no longer provides any advantages (see Figure 4). In particular, both stations start with a 54 Mbps bitrate. Due to collisions at high bitrates most frames are lost. Assuming the far station lowers its bitrate, while the close station still uses 54 Mbps, the bitrate resulting in the maximum throughput for the far station was 11 Mbps. In response the close station will maximize its

own throughput by using 11 Mbps as well. This arms race continues until lowering the bitrate no longer provides any advantages. We conjecture that the final bitrates depend on the devices being used and on the environment. In our setting the close station ends up using 11 Mbps while the far station uses 5.5 Mbps. At this point neither station could increase throughput. Hence contending selfish stations will lower their bitrate in order to get a higher throughput.

A higher transmit power can also increase throughput, though it is more sensitive to distances. If the close station is 10 cm from the AP, the far station could not increase its throughput using a higher transmit power. However, when placing the close station 70 cm from the AP the situation changes. Assuming both stations act selfish and use 11 Mbps, the throughputs of the close and far station are 0.50 ± 0.52 and 2.10 ± 1.05 Mbps, respectively. When the far station uses an amplifier as external antenna, its throughput increases to 3.66 ± 2.08 Mbps while the close station gets 0.44 ± 0.28 Mbps. Hence selfish stations will use a higher transmit power in an attempt to increase their throughput.

At higher distances both strategies still work. However, as the distance increase, it gets harder for a selfish station to beat other stations and increase its own throughput.

3.4 Defeating Countermeasures

We now investigate mechanisms which are designed to detect (and prevent) selfish behaviour. Unfortunately we found that even advanced detection mechanisms such as DOMINO [25] have weaknesses. Though they can reliably detect manipulations of interframe spaces and the backoff procedure, defending against stations which jam others appears difficult. The goal of jamming others is to increase their contention window (backoff counter), increasing the chance the cheater can access the channel. Frames of other stations can be jammed by a selective jammer (see Sect. 4.2).

An attacker only jams frames of other clients. It detects such frames based on the addresses in the MAC header. Hence detection mechanisms assume the header of frames remain valid, and use this to count the number of failed (jammed) transmissions of stations [25]. If a station has a significantly lower failed transmission count than all others, it is assumed to be jamming the other stations, and is punished (e.g. thrown of the network). The flaw in this technique is that the authenticity of jammed (corrupted) frames cannot be guaranteed. An attacker can forge jammed frames and fool the mechanism into thinking a station is jamming others. The targeted station is then wrongly punished.

We created a tool to forge jammed frames. In particular we used the `AR_DIAG_SW` register to force the wireless chip to append a bad FCS to every transmitted frame. To target a victim we forge jammed frames which appear to come from all other stations. As a result the victim will appear to have a significant lower number of failed transmissions compared to the others, and will be wrongly punished.

We conclude that existing systems tend to underestimate the power of attackers, and hope our work gives a better overview of both the capabilities and behaviour of attackers.

4. JAMMING

In this section we show how to implement continuous and selective jamming on commodity Wi-Fi devices. To the best of our knowledge we are the first to accomplish this.

Register	Description
`AR_DIAG_SW`	Disable carrier sense, abort ongoing reception, append bad FCS
`AR_D_GBL_IFS_SIFS`	*SIFS*
`AR_D_GBL_IFS_SLOT`	*SLOT*
`AR_DLCL_IFS`	CW_{min}, CW_{max}, and *AIFSN*

Table 1: Important registers and their functionality.

Type	Wireless Chip	Device
USB	Atheros AR9271	Alfa AWUS036nha
	Atheros AR9271	TP-Link WN722N
	Atheros AR9280	Netgear WNDA3200
	Realtek RTL8187L	Alfa AWUS036h
	Ralink RT2720	Belkin F5D8053 v3
	Ralink RT3070	Digiflex aw-u150bb
PCI	Atheros AR5112	Wistron NeWeb CM9
	Broadcom bcm4334	Galaxy i9305
	Intel jc82535rde	Intel 4965AG
	Intel wg82541rde	Intel 5100AGN
Access Point (PCI)	Atheros AR2413	Sagem F@st 3464
	Broadcom bcm2050	WRT54g v2
	Broadcom bcm4322	Linksys WAG320N
	Broadcom bcm5356	Asus RT-N10

Table 2: Devices and their wireless chips. Monitor mode is supported by all USB and Intel PCI devices.

4.1 Continuous Jamming

A continuous jammer transmits noise for an indefinite amount of time. The noise can consist of random energy pulses or data resembling the protocol being jammed. Generally one of these options is preferred. For instance, injecting random Wi-Fi frames might trigger warnings from intrusion detection systems, while random noise will be seen as non-Wi-Fi devices using the same frequency.

To turn a commodity Wi-Fi device into a continuous jammer we need to be able to carry out the follow tasks:

1. Disable carrier sense.
2. Reset all interframe spaces and disable backoff.
3. Prevent the chip from waiting for an ACK.
4. Queue a large number of frames for transmission.

Achieving the first two points is explained in Sect. 3. The most important registers we used are summarized in Table 1.

To address the last two points we need to know how the wireless chip is instructed to transmit frames. The `ath9k_htc` firmware maintains a linked list of frames to be sent. Each frame is encapsulated by a transmit descriptor, which includes metadata describing how it should be transmitted. Among the metadata is the `HAL_TXDESC_NOACK` flag. When set, the wireless chip does not wait for an ACK and will not retransmit the frame. To queue an infinite number of frames we turn the linked list into one self-linked element.

We have implemented the continuous jammer by modifying the `ath9k_htc` firmware and created a tool to send commands to the firmware [1]. Since the AR7010 chip supports 2.4 and 5 GHz we can jam both bands on any channel. It is

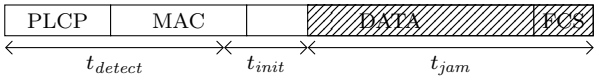

Figure 5: Timing requirements of selective jamming. After a frame header is detected and decoded, the jammer is initiated to jam the remaining content.

straightforward to turn our continuous jammer into a periodic jammer, which transmits only during certain intervals. An advantage of a periodic jammer is a lower consumption.

We tested our implementation using a WNDA3200 and AWUS036nha. Both devices are able to jam any channel in the 2.4 GHz band, with the WNDA3200 also able to jam channels in the 5 GHz band. All devices in Table 2 were susceptible to the attack: once the jammer was activated they all lost their connection to the AP. When monitoring the channel we observed that only the first frame injected by the jammer was visible. All devices in table 2 supporting monitor mode displayed this behaviour. This highlights an important difference between our jammer and the unfair channel usage described in Sect. 3. Though the high load of a selfish node could at times cause a fair station to disconnect, all its traffic would be visible in monitor mode.

The effectiveness of the jammer depends on how it is blocking frames. They can be blocked by triggering the carrier sense mechanism of the transmitter (preventing it from sending frames) or by mangling the frame at the receiver. Generally it is easier to silence a transmitter, since less power is required to trigger the carrier sense mechanism compared to mangling incoming frames. To establish an upper bound on the effectiveness of our jammer we tested the maximum distance for which the AWUS036nha could silence a TL-WN722n. This was done with a clear line-of-sight between the victim and the jammer. Testing whether a device was silenced was done by letting it inject a frame, and using a second device to monitor whether it was transmitted. With an AWUS036nha as the jammer, it was effective up to roughly 80 meters. When using an amplifier as external antenna the range was extended to roughly 120 meters.

Cheap and readily available jammers for the 2.4 and 5 GHz bands can have a high impact since many devices operate in these bands (e.g., security webcams, baby monitors, etc).

4.2 Selective Jamming

Selective jammers are arguably the most sophisticated and efficient. By targeting only selected frames an attacker can stealthy block specific frames from reaching their destination. In its simplest form a selective jammer emits noise whenever radio activity is detected. More sophisticated ones decode a prefix of a frame and then decide whether to jam the remaining content. In other words, they use higher-layer information to decide whether to jam it. The downside is that selective jammers can be difficult to implement, since they must adhere to strict timing requirements. This is illustrated in Figure 5 where t_{detect} is the time it takes to detect and decode the frame header, t_{init} the time needed to start the jammer, and t_{jam} the time when the frame is being jammed. To successfully jam a frame $t_{detect} + t_{init}$ must be lower than the time it takes to transmit the frame.

To implement selective jamming on commodity devices we must be able to accomplish the following tasks:

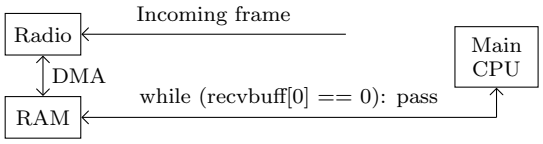

Figure 6: Using commodity devices to detect and process frames while they are still in the air.

1. Read the decoded header of a frame still in transit.
2. Abort receiving the current frame.
3. Immediately jam the channel.

Point 3 can be done by disabling carrier sense, setting all interframe spaces to zero, disabling the backoff procedure, and then injecting a dummy frame (see Sect. 4.1). Point 2 can be accomplished using the `AR_DIAG_SW` register.

There is no support to read the already decoded bytes of a frame still in transit. Nevertheless, one can still accomplish this if the wireless chip uses Direct Memory Access (DMA) to write already decoded bytes to memory. In that case we can monitor the receive buffer using the main CPU (see Figure 6). We first initialize the receive buffer with values that do not occur in valid 802.11 frames. Once these bytes are being modified, we know a frame is being received, and can read the already decoded bytes.

4.2.1 Implementation

We implemented the selective jammer by modifying the `ath9k_htc` firmware [1]. Currently it is designed to jam beacons and probe responses from a particular AP. The user can specify which SSID or MAC address to target and for how long to attack it. It is straightforward to modify our implementation to target different frames. In particular, the following fields can be used to target other frames: (1) MAC addresses in the header; (2) type and priority of frame; (3) sequence and fragment number. One can also use the first bytes of data in the packet, though this may make it difficult to jam the frame in time. Once decided to jam the frame, we inject a dummy frame transmitted at 1 Mbps to create a collision and mangle the frame. This will cause the FCS of the targeted frame to be invalid making the receiver drop the frame. To prevent the injected frame from being retransmitted we use the `HAL_TXDESC_NOACK` flag. When targeting unicast frames it is possible to inject an ACK immediately after jamming the frame. This prevents the sender from retransmitting the frame.

The selective jammer can also be modified to assure an attacker is the first to reply to certain frames. For example, an attacker can detect a probe request while it is still being transmitted, and immediately queue the transmission of a probe reply. To assure the reply is transmitted before any other station we disable backoff and set *AIFSN* to zero. A proof of concept of this attack has been implemented. The tool detects probe requests, jams them, and replies with a custom probe response. In Section 5 we use this to send probe responses with a modified channel number. Being able to reply faster than the legitimate source is known to facilitate other attacks as well. For example, it can also be used to forge DNS replies in an (unencrypted) network.

We conjecture that, with sufficient efforts, all operations can be implemented on other commodity devices as well.

4.2.2 Experiments

We selectively jammed beacons with the victim at 70 cm from the AP and the jammer at 1 m (all devices were located on a single line). The victim was put in monitor mode to track how many beacons were malformed. In each experiment we carried out the attack for 2 minutes. When an AWUS036nha jams a WNDA3200, on average 52% of the beacons are malformed. However, when an AWUS036nha is jamming a TL-WN722N, 99% of the beacons are malformed. Hence the effectiveness depends on the device of the victim. More precisely, the type of antenna, radio chip, internal noise filters, etc. can all influence how vulnerable a victim is. The device of the attacker also plays an important role. As mentioned, when an AWUS036nha uses its default antenna to jam a WNDA3200, on average 52% of the beacons are malformed. But when connecting an amplifier to the AWUS036nha and then jamming the WNDA3200, 92% of all beacons are jammed. The location of the victim and attacker also plays an important role. When a WNDA3200 jams another WNDA3200 in the 2.4 GHz band, 51% of beacons were jammed. If we switch the location of the attacker and victim, all beacons were jammed. This is expected, since the signal strength of the AP is now lower for the victim, meaning it becomes easier for the attacker to overpower it. More precisely, the attacker must transmit a signal powerful enough to overcome capture effect, otherwise the victim still correctly receives the original frame. Finally we tested our jammer in the 5 GHz band by using two WNDA3200's. In this case all beacons were jammed.

With beacons sent at 1 Mbps in the 2.4 GHz band, both the AWUS036nha and WNDA3200 start mangling bytes at position 52 of the beacon. When beacons are sent at 6 Mbps in the 5 GHz band, the WNDA3200 starts mangling bytes at position 88. This is sufficient to jam beacons, probe requests, probe responses, and other packets of similar size. Currently the jammer is limited by the time it takes the wireless chip to write the first decoded bytes to RAM. In particular we observed that the wireless chip writes to RAM only after it has decoded the first 48 bytes. Further reverse engineering may reveal a technique to force to chip to write to RAM earlier, resulting in faster reaction times.

An interesting observation was made when selectively jamming an AR5112 chip. When the signal power of the jammer was ± 16 dB higher than that of the AP, the AR5112 returned a prefix of the beacon, followed by the dummy frame injected by jammer. This was caused by the Message in Message (MiM) support of the AR5112. Devices that support MiM can resynchronise to a stronger frame while receiving a weaker frame, even if the stronger frame arrives after the preamble of the weaker frame [16]. Hence our selective jammer is an ideal tool to test whether a device supports MiM. For comparison, Lee et al. had to resort to more tedious experiments to show the AR5112 supports MiM [14].

4.2.3 Discussion

One limitation of our jammer is that it does not have access to the PLCP header of a frame still in transmit. In particular this means we are unable to access the length field in the PLCP header, and thus do not know how long the frame is. Normally the `AR_DataLen` field in the receive descriptor contains the length of the frame. However, for frames still in transmit, it contains the number of bytes written to RAM so far. We found no other data from which the length can be derived, though more analysis may overcome this obstacle.

Essentially our cheap selective jammer is capable of deterministically creating collisions. Apart from being used as a jammer, it can also be used to test new collision detection techniques, experiment with Message in Message radios, etc. In short, a cheap and easily available selective jammer not only shows that jamming poses a more serious threat than previously thought, it also facilitates research experiments.

5. CHANNEL-BASED MITM

In this section we present a man-in-the-middle attack on WPA1/2 secured networks. Our goal is not to decrypt traffic, but to reliable intercept and manipulate it. In Section 6 we use this to attack TKIP when used as a group cipher.

5.1 Background

If the goal of an attacker is to reliably sniff and manipulate traffic, merely monitoring the channel is insufficient. Inevitably packets will be missed, and it is difficult to block and manipulate traffic. Though a selective jammer can block certain packets, it is not reliable enough (see Sect. 4.2.2). Another problem is that selective jammers inherently do not have access to the complete frame, giving an attacker limited information to decide whether to block it. All these limitations disappear when having a MitM position.

Establishing a MitM position in a WPA1/2 secured network is difficult due to the 4-way handshake. This is because the generated session keys depend on the MAC address of the AP and client. Therefore, if we use a rogue AP with a different MAC address, the handshake will fail. Using the same MAC address as the real AP is not possible since the client and AP would simply communicate with each other.

5.2 Intercepting Encrypted Traffic

To intercept all traffic we will clone the AP on a different channel and forward all traffic to the real AP. This requires two Wi-Fi dongles: one operating on the channel of the real AP, and one cloning the AP on a different channel. Because both Wi-Fi dongles are physically close to each other they can receive each others frames, even though they operate on different channels. Hence, blindly forwarding packets between both channels may create an infinite loop. To avoid this we keep track of recently forwarded frames using their sequence numbers, and only forward new frames. To advertise our rogue AP we transmit a beacon every 102.4 ms and reply to probe requests using custom probe responses.

Once our rouge AP is started, we want to force clients to connect to it. Unfortunately, injecting probe responses containing the channel of the rogue AP does not cause clients to switch channels. Selectively jamming all beacons and probe responses is also not reliable, as some frames will inevitably be missed. Additionally we found that if a client was recently connected to an AP, it will simply assume it is still present, and immediately send an authentication frame. This technique allows a mobile device to quickly reconnect to a network. The authentication message is only 34 bytes long and too short to be selectively jammed. Hence selective jamming cannot be used to force clients connect to the rogue AP. To avoid all these issues we continuously jam the channel of the real AP (see Sect. 4.1). This forces the clients to switch to our channel and connect to our rogue AP. Once the clients have switched, we can stop the jammer, and start forwarding frames between the clients and the real AP.

5.3 Implementation

We implemented the attack in a command-line tool [1]. It allows a user to specify which AP to clone, whether to jam the channel of the real AP until clients connect to our AP, and whether to write all intercepted traffic to file.

The firmware was modified so injected frames are retransmitted in case of failure, and so ACKs are generated when frames are sent to us. Retransmitting frames is done by disabling the `HAL_TXDESC_NOACK` flag. Generating ACKs on the device cloning the AP is straightforward, as it only needs to listen on the MAC address of the AP. However, the device on the channel of the real AP must listen to the MAC addresses of all connected clients. To accomplish this we rely on virtual interface support, a hardware technology enabling a single device to listen on multiple MAC addresses. When a client is trying to connect to the rogue AP, our modified firmware simulates the addition of a new virtual interface.

We had to patch the driver and firmware to prevent modifications to forwarded frames. In particular we made the driver treat injected data frames as management frames. In the firmware we marked injected frames as CF-Poll frames to prevent the sequence number from being overwritten.

After establishing a MitM position an attacker will reliably capture all traffic. Additionally, packets can be blocked by not forwarding them. It is straightforward to update the tool so the attack can be executed even when the target is far away from the AP by forwarding frames over the internet.

5.4 Experiments

We performed several experiments to measure the reliability and impact of the attack. For the victim we used a Latitude E6500 running Linux 3.7.2, for the AP a Linksys WAG320N using firmware v1.00.08, and as attack machine a VMWare player instance having 2 GB RAM and running Linux 3.7.2 with our modified drivers and firmware. We used a TL-WN722N and WNDA3200 to intercept and forward traffic, and an AWUS036nha as the jammer. The AP was configured to operate in 802.11g mode.

When using continuous jamming to force clients to connect to our rogue AP, we consistently established a MitM position. Even clients already connected to the real AP switched to the rogue AP. We then measured the impact on latency, bandwidth, and web page loading times. When connected to the real AP the victim had a latency of 3.82 ± 10.4 ms. This increased to 7.45 ± 12.9 ms when connected to our rogue AP. Hence latency is doubled, which is expected since every packet must be forwarded by our rogue AP, and is thus transmitted twice. To test the impact on throughput we let the victim download a 100 MB file. Under normal conditions this takes place at 18.6 Mbps. When being attacked the speed is lowered to 8 Mbps. Finally we tested the impact on page load times when surfing the web. Under normal conditions a page was loaded in 0.76 ± 0.07 ms, which increased to 0.83 ± 0.08 ms when under attack. Since this is only a 9% slowdown users are unlikely to notice this and will not realise they are under attack.

These results can be optimized by implementing a rate adaptation algorithm when forwarding packets, and by using unfair MAC parameters as described in Sect. 3.

5.5 Countermeasures

There are legitimate reasons for an AP to change channel, for example to switch to a less occupied one. In the 5 GHz band, switching channels is even required to avoid interference with radars [11, §4.5.5.3]. Hence storing the channel of an AP you previously connected to is not feasible.

The attack can be detected by including the channel of the AP in the 4-way handshake. Unfortunately this requires a change in the existing protocol, making this difficult to realise in practise. Ideally the impact of the attack is reduced by using a secure encryption protocol such as CCMP.

6. TKIP AS A GROUP CIPHER

In this section we attack TKIP when used as a group cipher, which is the default security setting for most routers.

6.1 Attack Details

Previously TKIP was only investigated when used to protect unicast traffic (see Sect. 2.3). Our goal is to show that TKIP can also be attacked when used as a group cipher, i.e., when used to protect broadcast and multicast frames.

Directly applying the Beck and Tews attacks on broadcast packets fails when multiple clients are connected to the AP. In that case all clients will simultaneously send a MIC failure report (instead of only the targeted client as in the unicast scenario). Hence the AP immediately starts the TKIP countermeasures. Recall that the countermeasures disable all TKIP traffic for one minute, after which a new GTK is generated and clients can reconnect. Thus we would only be able to decrypt one byte. Our selective jammer cannot be used to block the MIC failure reports, because they are detected based on their unique length. More troublesome, MIC failures are generally sent at high bitrates, meaning even more advanced selective jammers are unable to reliably jam them. Instead we use the MitM attack from Sect. 5, allowing us to block MIC failure reports by not forwarding them. Note that we must still assure that individual clients do not send more than one MIC failure report every minute. Otherwise the client will disconnect from the network, even if the AP did not start the TKIP countermeasures.

Once a MitM position has been achieved we wait until a small broadcast packet is transmitted. We prefer small packets as these take less time to decrypt. Since ARP requests are small and sent when a client (re)connects to a network, they are an ideal candidate. After capturing an ARP request we decrypt its ICV and MIC using the Beck and Tews method, and guess the remaining content. Since we do not forward MIC failures to the AP, the TKIP countermeasures will not be actived when multiple clients send a MIC failure report. Once the packet has been decrypted we learn the keystream corresponding to the sequence counter (TSC) of the ARP request, and we can derive the MIC key for broadcast traffic [27]. With this we can inject 3 to 7 small packets to any client connected to the AP (the amount depends on the number of supported QoS channels). Furthermore, existing attacks relying on knowing the MIC key can be modified to work on broadcast packets. For example, by modifying the attacks presented in [29], we can abuse fragmentation to inject an arbitrary amount broadcast of packets, and we can efficiently decrypt arbitrary broadcast packets.

We continue by speeding up the attack. There are several techniques to accomplish this. First, if we do not detect a MIC failure after trying all possible 256 values, we know the target did not receive the packet with the correct guess. If it did, it would retransmit the MIC failure until we responded with an ACK. Hence we can immediately retry guessing all

values without fear of generating two MIC failures within one minute. In contrast, previous attacks on TKIP had to wait one minute because the attacker might have missed the MIC failure report while the AP did receive it.

If we are able to send broadcast packets to only one client we can further speed up the attack. We start the attack by sending guesses to one particular client. Once it sends a MIC failure report, we can immediately continue the attack by targeting another client. Each client will only see one MIC failure every minute, and the AP will see none, hence the TKIP countermeasures are never activated. We keep targeting different clients while assuring that an individual client will never send more than one MIC failure report every minute. The difficulty is in sending a broadcast packet to a specific client. Though it is possible by cloning the AP on multiple channels and spreading the clients out over these channels, such an approach is cumbersome and requires multiple Wi-Fi devices. Instead we rely on the observation that most devices do not process their own broadcast packets. That is, most devices will drop broadcast packets with as source address their own MAC address. Assuming we are attacking two clients simultaneously, we can target one client by using the source address of the other client. Finally we do not have to wait one minute when guessing the last remaining bytes. Though a client may send more than one MIC failure report within one minute, and therefore disconnect from the network, the AP will not see these MIC failures. Hence the AP will not activate its countermeasures.

6.2 Implementation and Experiments

We implemented the attack by extending the MitM implementation (see Sect. 5). To attack two clients we rely on the fact that a client will not process broadcast packets with as source its own MAC address. Once the clients are connected to our rogue AP, and completed the 4-way handshake, we wait for an ARP request. The ARP request is decrypted byte by byte, and finally the MIC key is derived.

Our experimental setup is the same as in Sect. 5.4, except that we now also use a Samsung Galaxy i9305 running Android 4.3 as a second victim. During the experiment we first let the two victims connect to the real AP, after which we started the attack. The execution time over 10 runs was 7.3 ± 0.6 minutes. This is the time from launching the tool to decrypting the APR reply and deriving the MIC key. Note that the execution time can be further reduced by targeting more stations simultaneously. Program output and network traces of the attack are available for download [1].

As a comparison we measured the execution time of the original Beck and Tews attack. The attacker used a TL-WN722N and the victim an AWUS036h. They were placed close to each other to reduce packet loss, an hence represents an optimal case. We used the tkiptun-ng tool from aircrack-ng. The execution time over 10 runs was 14.6 ± 1.1 minutes, more than twice the time of our attack.

6.3 Countermeasures

To prevent the interception of MIC failure reports, APs should securely acknowledge them. If the client does not receive the acknowledgement within time, it may be under attack, and should activate its TKIP countermeasures. This prevents an attacker from blocking MIC failure reports and attacking multiple stations simultaneously.

The attack can be mitigated by using a short rekeying timeout of 2 minutes or less. Ideally the attack is prevented by using a more secure encryption protocol such as CCMP.

7. RELATED WORK

Atheros drivers are popular for low-layer control over experiments, relevant examples are [25, 14, 8]. Custom firmware has also been used for low-layer control [3, 4, 9].

A significant amount of research on the behaviour of selfish stations rely only on analytic models and simulations [23, 6, 24]. Raya et al. perform experimental tests when a single node manipulates the contention window of the backoff procedure [25]. Pelechrinis et al. also perform experiments, but only consider stations which modify their CCA threshold [22]. However, an attacker can easily change additional parameters. We are not aware of any works doing a detailed experimental study on the behaviour of selfish stations, where multiple strategies are tested, and the capture effect is taken into account. The capture has been observed for 802.11 on Prism and Atheros chipsets [13, 8]. In [8] the authors attempt to reduce the unfairness caused by this effect. Lee et al. perform a detailed study on when the capture effect takes places, and found that the Atheros AR5112 not only exhibits capture effect but also supports Message in Message mode [14].

Bayraktaroglu et al. required a USRP to continuously jam the 2.4 GHz band [2]. Kim et al. modify the MadWifi driver of the Atheros 5212 to continuously emit meaningless frames in the 2.4 GHz band [12]. Noubir et al. used one USRP1 and two RFX2400 boards to build a selective jammer [19]. Unfortunately the reaction time of their setup was too slow to reliably jam frames [19, §6.1.4]. Cassola et al. required two USRP2s and two RFX2400 boards, totalling more than $3600, to selectively jam frames in the 2.4 GHz band [7]. Their jammer starts mangling frames sent using 1 Mbps at byte 38, while our jammer does this at byte 52. However, our dongle costs only $15, can jam both frequency bands, and is easier in use due to it's smaller size. In concurrent and independent work, Berger et al. used custom microcode to have control of medium access and de/encoding operations, and used this to implement selective jamming in the 2.4 GHz band [4]. The advantage of our approach is that we do not require such low-level control over medium access operations performed in the chip. Continuous and selective jammers have also been created to target other wireless protocols [30, 31]. Our selective Wi-Fi jammer is unique because it is both cheap and portable. Jamming can also be used as a defensive mechanism [6, 4, 30].

Beck and Tews found the first practically exploitable vulnerability in TKIP, which required QoS enhancements to be enabled [27]. Ohigashi and Morri used a MitM position to execute the attack even if QoS was not enabled [20]. In particular they assumed the client was not in range of the AP, and that the attacker acted as a repeater by forwarding all frames between the client and AP. In contrast, our MitM attack does not require that the client and AP are out of range. Over the years additional improvements of the Beck and Tews attack have been found [10, 18, 28, 29]. Several more theoretical attacks on TKIP have also been published [17, 26, 21]. Finally, the closest related work to our channel-based MitM attack we are aware of is the AirJack tool [15]. It clones an unencrypted network on a different channel and forwards all traffic over ethernet. Hence it cannot clone encrypted networks, whereas our attack can.

8. CONCLUSION

We were able to implement several low-layer attacks on Wi-Fi using open source Atheros firmware. This is surprising, since we only have access to a limited API to control the radio (e.g. we cannot transmit arbitrary signals, do not have access to raw signal data, don't modify medium access algorithms, etc). Additionally we bypassed systems designed to prevent some of these attacks, indicating that previous works have underestimated the capabilities of attackers. We also showed that our low-layer attacks facilitate attacks on higher-layer protocols, by attacking TKIP when used as a group cipher. Since TKIP is used significantly more as a group cipher than as a unicast cipher, this demonstrates that weaknesses in TKIP are still of high practical value.

Finally, the selective jammer, channel MitM, and TKIP attacks are available for download [1]. We hope these results aid in the creation of better countermeasures, and motivate people to only use the more secure (AES)-CCMP.

9. ACKNOWLEDGEMENTS

This research is partially funded by the Research Fund KU Leuven, and by the EU FP7 project NESSoS. With the financial support from the Prevention of and Fight against Crime Programme of the European Union (B-CCENTRE). Mathy Vanhoef holds a Ph. D. fellowship of the Research Foundation - Flanders (FWO).

10. REFERENCES

[1] `http://modwifi.bitbucket.org/`.

[2] E. Bayraktaroglu, C. King, X. Liu, G. Noubir, R. Rajaraman, and B. Thapa. On the performance of IEEE 802.11 under jamming. In *INFOCOM*, 2008.

[3] J. Bellardo and S. Savage. 802.11 denial-of-service attacks: real vulnerabilities and practical solutions. In *Proc. of the 12th USENIX Security Symp.*, 2003.

[4] D. S. Berger, F. Gringoli, N. Facchi, I. Martinovic, and J. Schmitt. Gaining insight on friendly jamming in a real-world IEEE 802.11 network. In *WiSec*, 2014.

[5] G. Berger-Sabbatel, A. Duda, O. Gaudouin, M. Heusse, and F. Rousseau. Fairness and its impact on delay in 802.11 networks. In *GLOBECOM*, 2004.

[6] M. Cagalj, S. Ganeriwal, I. Aad, and J.-P. Hubaux. On selfish behavior in CSMA/CA networks. In *INFOCOM*, 2005.

[7] A. Cassola, W. Robertson, E. Kirda, and G. Noubir. A practical, targeted, and stealthy attack against wpa enterprise authentication. In *NDSS Symp.*, Apr. 2013.

[8] S. Ganu, K. Ramachandran, M. Gruteser, I. Seskar, and J. Deng. Methods for restoring MAC layer fairness in IEEE 802.11 networks with physical layer capture. In *REALMAN*, 2006.

[9] D. Halperin, W. Hu, A. Sheth, and D. Wetherall. Tool release: Gathering 802.11n traces with channel state information. *ACM SIGCOMM CCR*, 2011.

[10] F. M. Halvorsen, O. Haugen, M. Eian, and S. F. Mjølsnes. An improved attack on TKIP. In *14th Nordic Conf. on Secure IT Systems (NordSec)*, 2009.

[11] IEEE Std 802.11-2012. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 2012.

[12] Y. S. Kim, P. Tague, H. Lee, and H. Kim. Carving secure wi-fi zones with defensive jamming. In *ASIACCS*, 2012.

[13] A. Kochut, A. Vasan, A. U. Shankar, and A. Agrawala. Sniffing out the correct physical layer capture model in 802.11b. In *ICNP*, 2004.

[14] J. Lee, W. Kim, S.-J. Lee, D. Jo, J. Ryu, T. Kwon, and Y. Choi. An experimental study on the capture effect in 802.11a networks. In *Proc. of the 2nd ACM Intl. Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization*, 2007.

[15] M. Lynn and R. Baird. Advanced 802.11 attack. In *Black Hat Briefings*, 2002.

[16] J. Manweiler, N. Santhapuri, S. Sen, R. Roy Choudhury, S. Nelakuditi, and K. Munagala. Order matters: Transmission reordering in wireless networks. In *MobiCom*, 2009.

[17] V. Moen, H. Raddum, and K. J. Hole. Weaknesses in the temporal key hash of wpa. *Mobile Computing and Comm. Review*, 2004.

[18] M. Morii and Y. Todo. Cryptanalysis for rc4 and breaking wep/wpa-tkip. *IEICE Trans.*, 2011.

[19] G. Noubir, R. Rajaraman, B. Sheng, and B. Thapa. On the robustness of ieee 802.11 rate adaptation algorithms against smart jamming. In *WiSec*, 2011.

[20] T. Ohigashi and M. Morii. A practical message falsification attack on wpa. In *Joint Workshop on Information Security (JWIS)*, 2009.

[21] K. G. Paterson, B. Poettering, and J. C. Schuldt. Plaintext recovery attacks against wpa/tkip, 2013.

[22] K. Pelechrinis, G. Yan, S. Eidenbenz, and S. Krishnamurthy. Detecting selfish exploitation of carrier sensing in 802.11 networks. In *INFOCOM*, 2009.

[23] O. Queseth. The effect of selfish behavior in mobile networks using CSMA/CA. In *Proc. of the 61st IEEE Vehicular Technology Conf.*, 2005.

[24] S. Radosavac, J. S. Baras, and I. Koutsopoulos. A framework for MAC protocol misbehavior detection in wireless networks. In *Proc. of the 4th ACM workshop on Wireless security*, WiSe '05, 2005.

[25] M. Raya, J.-P. Hubaux, and I. Aad. DOMINO: a system to detect greedy behavior in EEE 802.11 hotspots. In *MobiSys*, 2004.

[26] P. Sepehrdad, S. Vaudenay, and M. Vuagnoux. Statistical attack on rc4 distinguishing wpa. In *EUROCRYPT*, 2011.

[27] E. Tews and M. Beck. Practical attacks against WEP and wpa. In *WiSec*, 2009.

[28] Y. Todo, Y. Ozawa, T. Ohigashi, and M. Morii. Falsification attacks against wpa-tkip in a realistic environment. *IEICE Trans.*, 2012.

[29] M. Vanhoef and F. Piessens. Practical verification of wpa-tkip vulnerabilities. In *ASIACCS*, 2013.

[30] M. Wilhelm, I. Martinovic, J. B. Schmitt, and V. Lenders. Wifire: A firewall for wireless networks. In *SIGCOMM*, 2011.

[31] W. Xu, W. Trappe, Y. Zhang, and T. Wood. The feasibility of launching and detecting jamming attacks in wireless networks. In *Proc. of ACM MobiHoc*, 2005.