Reconnection-Based Covert Channels in Wireless Networks

Sebastian Zillien(✉) and Steffen Wendzel

Centre of Technology and Transfer, Worms University of Applied Sciences, Worms, Germany
szillien@hs-worms.de

Abstract. In recent years, malware is increasingly applying means of hidden communication. The emergence of network-capable *stegomalware* applies such methods to communication networks. In this paper, we introduce and evaluate two covert channels that utilize reconnections to transmit hidden information in WiFi networks. We implement these covert channels in the 802.11 protocol by abusing the authentication mechanism of these networks. Furthermore, we propose detection methods and countermeasures for both covert channels. Our implementation and quick-start guide are available as open source code on GitHub to aid replicability (https://github.com/NIoSaT/WiFi_Reconnection_CovertChannel).

Keywords: Network covert channel · Wireless networks · Steganography · Information hiding · Network security

1 Introduction

Covert channels are secret and policy-breaking communication channels [14]. In networks, they are increasingly used by steganographic malware (*stegomalware*) to hide the exfiltration of stolen passwords or command and control (C&C) channels. On the other hand, covert channels can also be used to circumvent government censorship. However, the majority of reported covert channel cases are of illegitimate nature and used as part of stegomalware [2,8].

Several techniques to implement such network covert channels have been published so far. These techniques can be organized in a *hiding pattern*-based taxonomy [9,13]. Patterns fall into two broad categories, timing and storage channel patterns. Timing channel patterns transmit their hidden data by manipulating temporal attributes of network traffic, like inter-arrival times or the order of packets. Storage channel patterns transmit their hidden data by manipulating values of the network traffic, like source addresses or the size of packets.

The most recent pattern, PT15, uses artificial reconnections to transmit hidden data [10]. It was only analyzed for the MQTT protocol so far. Pattern PT15 falls into the category of timing channels. Many network types, including wireless networks, have some sort of connection mechanism that can be abused for this type of covert channel.

Published by Springer Nature Switzerland AG 2021
A. Jøsang et al. (Eds.): SEC 2021, IFIP AICT 625, pp. 118–133, 2021.
https://doi.org/10.1007/978-3-030-78120-0_8

In this paper, we show that the artificial reconnections pattern can be used in the widely-deployed 802.11 WiFi networks. We provide two different implementations that exploit the connection establishment mechanism of 802.11 networks to realize a reconnection-based covert channel. When clients connect to a wireless network, they move through 3 states. Each state allows for different frames to be used by the client. A regular use of the network is only possible in state 3. Starting in state 1, a client moves to state 2 with a successful authentication, a deauthentication moves the client back to state 1. From state 2 they move up to state 3 by associating. A disassociation moves the client back to state 2. These steps all happen unencrypted so it is easy for an attacker to observe these steps or even inject spoofed frames to interfere with the connection process.[1] We use these properties to realize our covert channels by disconnecting selected clients in a certain order to transmit a secret message.

Since our covert channels are bound to a physical location, they are not ideal to control a botnet or trojan virus. Our main application scenario is the hidden exchange of data, such as encryption keys, in public places. A sender and (multiple) receiver(s) can exchange such data "over the air" without physically meeting or directly exchanging data between their devices.

The remainder of this paper is structured as follows. We first highlight fundamentals and related work in Sect. 2 and then present our concept in Sect. 3. We discuss the implementation of our two methods in Sect. 4 and 5. Afterwards, we evaluate their performance in terms of robustness and throughput. Next, we present and evaluate passive countermeasures (detection approaches) in Sect. 6, followed by active countermeasures (limitation of channel capacity) in Sect. 7. In Sect. 8, we compare our covert channel with two other publications' wireless covert channels. Section 9 concludes and provides an outlook on future work.

2 Fundamentals and Related Work

To transmit data using PT15, the sender forces clients to reconnect to a network or a server while the receiver monitors the network. The receiver then decodes the ordering of the reconnecting clients to retrieve the hidden data. Since the pattern PT15 is still new, there are no publications on it outside of the initial publication [10]. However, there are several related covert channels.

Kraetzer et al. implement and analyse two different covert channels in 802.11 networks [7]. A covert storage and a covert timing channel. For both covert channels, the covert sender is artificially duplicating frames. Therefore, both channels require other clients in the network that send legitimate data to provide the cover traffic. For the storage channel, the covert sender duplicates a foreign packet but modifies the duplicated packet to include the hidden data. For the

[1] A common attack on this process is the "deauthentication attack", a type of denial of service attack. An attacker continuously sends spoofed deauthentication frames which force a client back into state 1. This effectively disables network access for this client as long as the attacker keeps sending these frames.

timing channel, the covert sender waits until a certain client sends a packet and then the covert sender sends a duplicate of the original packet.

In [3], Classen et al. introduce four approaches for covert channels in wireless networks. The focus of their work is on the physical layer. They encode hidden data by modifying physical aspects of wireless frames, like frequency or phase shift.

HICCUPS (*Hidden Communication System for Corrupted Networks*) [11] was one of the earliest published covert channels for 802.11 networks. The approach functions similar to [7], by modifying header values of wireless packets. HICCUPS also offers a "corrupted frame mode", in which the covert sender embeds the hidden data in the payload part of the packet but sets an intentionally wrong CRC value. The covert receiver knows how to interpret the hidden data, while other clients will ignore the packet as malformed.

In [6], Holloway and Beyah propose a timing covert channel called *Covert DCF* for 802.11 networks that works by manipulating the random back-off time of the CSMA/CA mechanism. To send a secret symbol, the covert sender replaces the random back-off time with a chosen delay. The covert receiver observes the inter-arrival times of the frames and decodes the secret symbol.

In [15], Zhao describes a covert channel based on the 802.11e QoS standard. The channel manipulates 8 bits of the frame header in order to encode the hidden data, while using legitimate network frames as the carrier. Association frames are used to signal the beginning and the end of a message. Zhao also proposes a transmission mode where the entire message body of wireless frames is used for the hidden data to increase the bandwidth of the covert channel.

Recently, Guri showed how data can be exfiltrated from air-gapped systems over WiFi signals using the so-called *AIR-FI* covert channel. Therefore, signals in the 2.4 GHz WiFi frequency bands are generated through DDR SDRAM memory buses; these signals can be recognized by closely located receivers [5].

Additional publications deal with further aspects of network covert channels, see [13,14] (and references therein) for an overview of hiding techniques. For instance, Carrega et al. present methodological approaches to gather data for stegomalware detection as a joint effort of two EU projects [2] while Carrara and Adams discuss non-traditional out-of-band covert channels [1].

3 Concept and Implementation

Both of our covert channels share the same basic approach: Covert sender and covert receiver agree upon which WiFi clients they want to use for the cover channel. We call them $C_1, ..., C_n$. Covert sender and receiver now agree on an encoding schema. Each client C_i is associated with a secret symbol S_i. To send the symbol S_i the covert sender forces the client C_i to reconnect to the network. The covert receiver continuously monitors the network for reconnections. If the receiver recognizes a reconnection of client C_i, it interprets it as the symbol S_i. For instance, to send the secret message $S_1|S_2|S_3|S_1$, the covert sender would

force reconnections from the clients in order of $C_1|C_2|C_3|C_1$. Figure 1 shows the setup and concept of our covert channels.

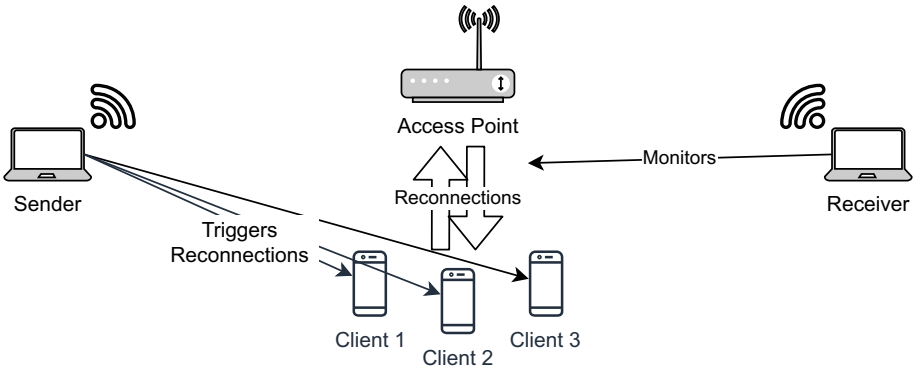


Fig. 1. Concept of the covert channel

Both of our covert channels are *indirect* covert channels. That means the covert sender does not directly communicate with the covert receiver by exchanging network packets. Instead, the covert sender manipulates a third party, called central element, and the covert receiver monitors the reactions of the third party to recognize the secret messages. In a direct covert channel, the covert sender directly sends data to the covert receiver. This makes it easier to find the covert sender and receiver once the covert channel is found.

The indirect approach provides two benefits to our covert channels:

1. *Anonymity:* Both, sender and receiver do not have to "join" (authenticate against) the WiFi network, they can work solely from the "outside". If the covert channel is detected, there is no evidence on who is the covert receiver. It is even possible to use the covert channel while the covert sender does not know who the covert receiver is. Only the covert sender interacts actively with the network but uses frames with a spoofed source address.
2. *Decoupling of Sender & Receiver:* Although our covert channel relies on wireless transmissions, covert sender and covert receiver never have to be in signal range of each other. The covert sender and covert receiver only need to be within range of the used clients $C_1, ..., C_n$. Therefore, our covert channel can almost double the range of a direct wireless communication between sender and receiver. The actual range of the covert channel is largely dependant on the RF environment, the signal power of the clients and the antenna gain of the receiver.

Proof of Concept Encoding. Our proof of concept implementation uses three clients C_1, C_2, C_3. We thus have three possible secret symbols S_1, S_2, S_3. Therefore, we implemented a ternary encoding with which we can encode 27 different

characters (26 letters and space) with 3 digits. To encode a message the sender first converts the message to a list of numbers. Each number is then transformed to a 3 digit ternary code. Each code symbol (0, 1, 2) corresponds to a certain client. As an example we have $26_{10} = 2 \times 3^2 + 2 \times 3^1 + 2 \times 3^0 = 222_3$.

This concept can be expanded with more clients. Only the capabilities of the network hardware limit how many clients can be used. More clients result in a higher covert channel bandwidth as we need less symbols to transmit the same amount of information. If we had 27 clients to work with, we would only need a single symbol for each character instead of the three that we used. If we had 64 clients, each client could signal a symbol from a Base64 encoding.

Hardware Requirements. We used regular "of the shelf" hardware to implement our covert channel. Covert sender and receiver require a wireless adapter that can operate in the *monitor mode* [12]. This mode allows the wireless adapter to do two things that we need for our covert channel: *1)* capture all frames that are sent in a network so that the device does not drop packets not intended for it. This allows the covert receiver to monitor all reconnections that occur within the network. *2)* inject spoofed frames into the network; the covert sender uses this feature to send the forged deauthentication/disassociation frames.

The specific models we used are ALFA AWUS036**NH** (receiver side), ALFA AWUS036**NHA** (sender side), ALFA AWUS036**ACM** Wireless USB Adapter (measurements and recordings), and ESP32 development boards as clients (these are often used in IoT devices).

4 Covert Channel Method 1

Our first method uses a deauthentication attack to force clients to reconnect to the network. This allows us to use any clients that are present in the network.

Scenario. This covert channel could be used to hide the process of exchanging encryption keys between two parties. Both parties would agree on a time and a public place, likely a café or an airport. They also need to agree on a list of clients they want to use beforehand. The two parties would arrive separately at the public place as to not raise suspicion. They exchange secret keys over the covert channel and leave separately.

4.1 Covert Sender

We implemented the sender side in form of a python script with the Scapy framework (https://scapy.net/). An important part of the performance of the covert channel is the correct timing of the clients' reconnections. If a client would be forced to reconnect before finishing the first reconnection, the transmission of the symbol would fail. Therefore we implemented an option `--wait` to add delays between reconnections. We also added the `--delay` option, which adds a

delay after three reconnections, i.e. after each letter. This is used to spread the transmission over a longer time to be stealthier.

Another aspect that needs to be configured is the amount of deauthentication frames that the sender uses to force the reconnection of a client. We found that a single deauthentication frame would not always result in the reliable deauthentication of a client. We noticed that programs like the aircrack suite also use multiple deauthentication frames. Therefore, we also added an option to change the extend of the deauthentication burst.

The sender-side procedure is explained below:

a) The input string (secret message to be transferred) is converted to numbers in the range of 0 to 26. All characters that are not in the lower alpha set are mapped to 0, which is the number for the space character.
b) The numbers are then converted to a three-digit ternary code.
c) The symbols are all concatenated into a single list.
d) The sender iterates over the list and the symbols 0, 1 and 2 are mapped to the clients that are used in the covert channel.
e) Before each reconnection, the sender waits for the duration of the configured delay of the `--wait` option. After the transmission, the additional delay from the `--delay` option is performed.
f) For each symbol, the configured number of deauthentication frames are sent.
***** Steps d), e) and f) are repeated for each symbol that is transmitted.

Our proof of concept sender works on a "best effort" principle. That means the covert sender does not know if a client actually performed a reconnection or if the covert receiver actually received the symbol.

4.2 Covert Receiver

We also implemented the covert receiver in the form of a python script that uses Scapy. Our receiver implementation only needs to know which client corresponds to which code symbol and does not need additional configuration parameters, as the receiver can work with any possible configuration of our sender side script.

Receiving Procedure. Our proof of concept receiver constantly monitors the network and decodes everything that is received using the following procedure:

a) Continuously monitor the network for reconnecting clients.
b) If a known client reconnects, append the corresponding symbol to the receiver list.
c) When the number of symbols in the receiver list is a multiple of three, decode the list to a string.
***** Repeat steps a)–c) until interrupted.

4.3 Evaluation

We evaluated the performance of method 1 in two ways. We tested the reliability and the data throughput of the covert channel.

Reliability. We found three stages where a transmission error could occur. a) from the sender to the client, b) "inside" the client and c) from the client to the receiver. We performed all our tests on an isolated network, while other wireless networks were present. We used a home router (TP-Link WR703N) as access point and three ESP32 boards as clients.

a) The first problem are unreliable deauthentications. To get an overview on how the deauthentication attack performed with various burst sizes, we conducted three test runs. The results of this test are shown in Table 1. As shown, we achieved high success rates with 64 to 80 deauthentication frames. But we never reached a 100% success rate in all three tests. That means we can improve the reliability of step a) but we do not make it completely reliable.

Table 1. Results: deauthentication test

Number of deauthentication frames	Success rate		
	Test 1	Test 2	Test 3
1	0%	3%	2%
10	36%	32%	53%
64	98%	94%	97%
70	98%	93%	96%
80	100%	100%	97%
150	79%	89%	95%

b) The second problem are the reconnection times of different clients. As stated above, clients need a measurable amount of time to reconnect. And if a new reconnection is triggered too early, it will result in a transmission error. Therefore, we added an option for a delay after each reconnection to our code. We performed several tests to find out how different clients react to a deauthentication attack. To do this, we caused 20 deauthentications for each client and recorded the time it took them to reconnect. If a client took longer than 10 s to reconnect or did not reconnect at all, we recorded 10 s as reconnection time.

We found that an iPhone and an Amazon Fire Tablet performed poorly in this test; their reconnection times were slow and inconsistent between 5 and 10 s. The ESP32 boards were significantly faster, but even their reconnection times varied between 0.2 and 2 s. We can use the delay option of our sender to optimize the delay to improve the error source b). But we cannot completely eliminate the problem since the sender does not know if the client is actually done with the previous reconnection.

c) The last step of the transmission, from the client to the receiver, is outside of our control. The client performs a reconnection and exchanges information

with the access point the receiver tries to listen to. But as this step is again wireless, there are several ways that the signals could get corrupted on their way to the receiver. 802.11 networks have multiple mechanisms that make wireless transmissions reliable, but these mechanisms work to make the transmission reliable for the actual participants and not for external listeners. So even if step a) and b) succeed, it is still possible for the transfer to get corrupted in step c).

Running Errors. One problem of our encoding scheme are "running errors". Since we use a three-digit code, all transmissions are aligned in groups of three. If the receiver now misses a single symbol, all following symbols are out of alignment by one step. The same happens when a single spurious symbol is received, all future symbols are then pushed back by one and are out of alignment. Figure 2 visualizes this problem.

Original	123 123 123 123 123 123 123
a)	123 123 123 122 312 312 312 3
b)	123 123 123 131 231 231 23

Fig. 2. Two transmission errors in ternary encoding

Summary. Given that even after above-mentioned optimizations are applied, we do not reach a reliability of 100% (but get close to it), it would be beneficial for the covert channel to implement some sort of error correction or acknowledgement mechanic for real world applications. A simple method would be to split a secret message into multiple smaller messages and stop and restart the transmission with a delay after each part.

Data Throughput. We tested the influence of several configuration parameters on the data throughput. To measure this, we transmitted the same message with different configuration parameters. As we expected, the burst size did not influence the data throughput in a significant way. The frame-sending process itself is fast and the biggest part of the duration comes from artificial delays we had to add. As expected, the two delay options (after each reconnection and after each three reconnections) scaled linearly. The higher the delays the worse the throughput.

We also tested the maximum data throughput that we could achieve. We optimized the settings for our network environment and clients to transmit *HelloWorld* in 20.143 s. That means we transmitted 30 tenary symbols in 20 s (0.5 letters/s).

To encode 27 different characters, we would need 5 binary digits. Therefore, *HelloWorld* would need 50 binary symbols (1 and 0). That means we would transmit 50 bits of information in roughly 20 s, which results in 2.5 $bits/s$.

With additional clients, we could not increase the symbol rate but each symbol could hold more information. If we had 27 clients, we would only need a single symbol for each letter of our message and with 64 clients we could transmit 6 bits worth of data with a single reconnection.

Therefore we could theoretically increase the throughput almost indefinitely, but handling hundreds of clients would be impractical and could result in problems with the access point.

5 Covert Channel Method 2

Our second method uses clients that can be controlled by the sender. Therefore, we no longer need to use a deauthentication attack, instead the clients reconnect themselves when triggered by the sender.

Scenario. This covert channel could be used as a form of number station to distribute new encryption keys to field agents. The sender is set up in a public place like a café with IoT equipment and repeatedly sends out secret data over the covert channel. The agent could then visit the café at a suitable time, receive the keys and leave again without having personal contract with anybody that could reveal their hidden identity. A dissident could also use this method to send illicit information to the press without risking prosecution from the government.

5.1 Covert Sender

The core idea of the sender is still the same as with method 1. The sender reconnects certain clients in a certain order to transmit the hidden data. The difference is that the sender can now directly control the clients and the clients can report their status back to the sender.

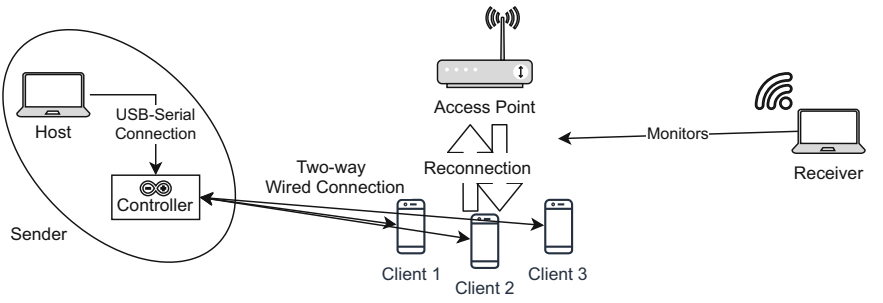


Fig. 3. Technical realization of method 2

Sender-Client Communication. We used a simple connection with two wires between the sender and each client, see Fig. 3. One for data from the sender to the client and one for data from the client to the sender. The client constantly reports its connection status to the sender. This is done via the voltage level of the *status wire*. A high voltage signals that the client is connected to the network and ready for a new reconnection. A low voltage means the client is currently not connected to the network and therefore not ready for a new reconnection. The sender uses the *command wire* to trigger a reconnection of the client. The command wire is normally on a low voltage level and to trigger a reconnection, the sender sends a pulse of high voltage over the command wire. The client detects the rising voltage edge and performs a reconnection.

Hardware Requirements. In addition to the hardware that we used for method 1, we also used the inputs and outputs of an Arduino microcontroller to implement the two-way communication between sender and clients.

Sending Process. The sending process for method 2 is similar to method 1. The main difference is that the sender does not use a guessed delay after each reconnection but instead uses the status information from the clients to time reconnections: First, the input string is converted to a list of clients (see Sect. 4.1 for details). Next, the sender waits until the particular client is ready for a reconnection. Then, the sender sends the reconnection signal to the client and waits for the client to finish the reconnection, before the sender moves on to the next symbol.

The two-way communication between sender and clients allows us to time the reconnections precisely without guessing the needed delays. We still added an additional delay after each symbol sent which can be used to spread the transmission over a longer time.

5.2 Covert Receiver

The receiver is the same one as for method 1, i.e. it monitors the network for reconnections, logs them and decodes the message.

5.3 Evaluation

Robustness. With method 2 we were able to significantly improve the reliability of our covert channel. The two-way communication between sender and client increased the reliability of steps a) and b) (see Sect. 4.3) to 100% in all our tests. The problem with step c) still exists and could not be solved with method 2. To improve step c) we would need some sort of error correction or acknowledgement mechanism.

Data Throughput. The status information from the clients allows us to use tighter timings on the reconnections, so we were able to significantly increase the maximum throughput of our covert channel. We were able to transmit 36 letters in under 21.3 s. With these numbers we send 1.698 letters per second or 5.1 symbols per second. With the same calculations from method 1, we transmit roughly 180 binary bits worth of information in 21.2 s or 8.5 bits/s. In comparison, method 2 is 3.4 times faster than method 1. As before, utilizing more clients would increase the throughput further (again not by increasing the symbol rate but by making each symbol carry more information).

6 Passive Countermeasures: Covert Channel Detection

To get a baseline for the detection of our covert channel, we created reference recordings in multiple network environments. A university building, a home network and our test network. For our detection, we focused on two types of frames: deauthentication and association request frames. Our reference recordings all had similar statistics. Table 2 shows the results of our tests conducted in the eduroam network in a university building. Recording 3 had the most "interesting" frames of all our recordings. The home and testbed recordings were all more similar to recording 1 and 2.

Table 2. Results of the eduroam Measurements

	Recording 1	Recording 2	Recording 3
Recording length	68 min	71 min	60 min
Frames recorded	268.086	575.577	629.203
Deauth. frames	1	0	24
Asso. req. frames	7	4	47

6.1 Detection of Covert Channel Method 1

To generate data for our detection, we performed recordings of transmissions of our covert channel. We used several different configurations – Table 3 presents some of these results. As shown, method 1 creates high numbers of deautentication frames. This is due to the burst size that is required to achieve a reconnection of the client.

For our detection, we used a sliding window approach. That means that we did not look at the complete transmission at once, but instead analyzed it in smaller slices. This allows us to use the approach on a recording or with live traffic. Our detection works as described in Algorithm 1.

Table 3. Method 1: results of the covert channel measurements

	Test 1	Test 2	Test 3
Recording length	56 s	52 s	54 s
Frames recorded	9688	9123	9250
Deauth. frames	6762	6156	6272
Asso. req. frames	30	30	30

Algorithm 1: Detection Algorithm - Method 1

```
Read/Sniff frames for L seconds;
Loop
    C := Count of the deauthentication frames in the window;
    if C >= T then
        raise alarm;
    end
    Discard oldest n seconds of frames;
    Read/Sniff frames for n seconds;
EndLoop
```

6.2 Detection of Covert Channel Method 2

To detect our second covert channel, we recorded multiple transmissions with different configurations. Table 4 shows representative results of our tests. As can be seen, we no longer have deauthentication frames in our recordings – only disassociation and association frames, since the clients produce these frames when they reconnect. Therefore, we slightly modified our detection approach: the detection now considers the number of association requests to differentiate between legitimate and covert channel traffic.

Table 4. Method 2: Results of the Covert Channel Measurements

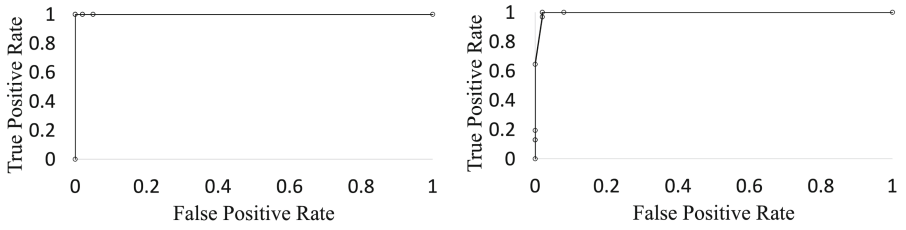
	Test 1	Test 2	Test 3
Recording length	21 s	2.3 min	18.4 min
Deauth. frames	0	0	0
Disas. frames	107	105	108
Asso. req. frames	108	107	108

6.3 Evaluation of Detection Methods

To evaluate our detection methods, we used 100 recordings of legitimate traffic and 30 recordings for each covert method (to achieve a 1:3 ratio as covert traffic

is usually not as present as normal traffic and we wanted to show its detectability under harsh circumstances which would potentially lead to a higher number of false positives).

For method 1, we used delay values from 0 s to 2 s and for method 2 we used delay values from 0 s to 30 s. For both methods, we transmitted multiple different messages. However, as our detection method neither depends on the information *which* clients get disconnected nor their *order*, we do not discriminate between plaintext and randomized messages as they make no difference. Figure 4 shows the ROC curves of our detection approaches with sliding window-based detection threshold.



(a) Method 1 (b) Method 2

Fig. 4. ROC curves - detection of methods 1 and 2

As shown, we achieved a perfect detection for method 1 (AUC = 1.0, optimal with $T = 15$). This is due to the fact that we have to use deauthentication bursts with method 1. Thus, even a single reconnect produces more deauthentication frames than all legitimate recordings had in their entire length.

When we used high thresholds with method 2, we had false negatives, but with smaller thresholds we were able to reliably detect our recordings without adding too many false positives (AUC = 0.9994, optimal with $T = 15$). We can configure method 2 to be more stealthy, but then we would have to sacrifice large amounts of bandwidth, as legitimate traffic has largely different statistics compared with our covert channel.

7 Active Countermeasures: Covert Channel Limitation

We found multiple potential approaches to counter a deauthentication attack and therefore our first method. The most prominent one is the new WPA3 standard that secures the connection process of 802.11 networks. This new standard can, if no backwards compatibility to older standards is enabled, block our first method completely. But our second method would still work in WPA3 only networks.

Therefore, we propose another countermeasure that will also disrupt our second method. So-called spurious disruptions were used already years ago for local covert channels in order to limit their channel capacity [4]. We apply spurious

disruptions by having the access point disconnecting randomly selected clients in randomized intervals. Each reconnection of a client that belongs to the covert channel results in a decoding error for the receiver. We tested this idea for both of our covert channels with good results. As described before in Sect. 4.3, our encoding is susceptible to spurious symbols. This countermeasure introduces many spurious symbols, as the receiver can not distinguish between a reconnection from the covert channel or a reconnection from the countermeasure. In our tests, we were able to successfully disrupt all messages that we sent, by reconnecting a client every 10 s. Depending on the timing, the first letters of a message were transmitted correctly, but no message was transmitted completely without disruption.

A countermeasure like this does also affect legitimate users of the network. We performed empirical tests to estimate the effects on other network users. To do this, we connected an iPhone and an Amazon Fire Tablet to our test network and used the devices to surf the internet and stream videos, while periodically disconnecting the devices from the network. We did not notice any problems from an end user standpoint. Videos kept playing and websites were loading at regular speeds. Other devices with other usage scenarios might experience recognizable problems.

The covert channel on the other hand could defend itself by adding error correction to the transmission. With added error correction, it is a matter of trading bandwidth for reliability. Since both covert channels do not provide a high bandwidth, they are vulnerable to such countermeasures.

8 Comparison with Other Covert Channels

In this section, we compare the performance of our covert channels with two other covert channels that exploit characteristics of wireless networks.

8.1 Kraetzer et al.: WLAN Steganography

In [7], Kraetzer et al. also analyzed their covert channels based on bandwidth, reliability and detectability. For their covert storage channel, which duplicates wireless packets and replaces the payload with the covert data, they calculated a theoretical bandwidth of 4 bytes per second in their chosen configuration. In practice, they were able to achieve 2.1 bytes per second, as the sending process was slowed down since the sender had to wait for appropriate packages to duplicate.

Their covert timing channel was not evaluated with a real-world test, but they gave a bandwidth of 0.2 bytes per second. This lower bandwidth was expected, as their timing channel only transmits a single bit per packet.

Our covert channel delivered 0.3125 bytes per second (method 1) and 1.06 bytes per second (method 2). Our tests were conducted with 3 clients. With additional clients, we could increase the throughput. With 64 clients we could theoretically achieve 1.1 bytes/s and 3.8 bytes/s.

As for the robustness, Kraetzer et al. also found that their covert channel was not 100% reliable. They encountered transmission errors in form of duplicated letters. They were not able to find the source of these errors, which matches our difficulties with unreliable wireless transmissions. The authors did not provide test results for the detection of their covert channels in form of a ROC curve or a confusion matrix. But they found their storage channel to be easily detectable, by comparing the payloads of retransmitted frames.

8.2 Zhao: Covert Channels in 802.11e Wireless Networks

The covert channel described in [15] manipulates bits of the frame header in order to encode the hidden data. The covert channel has a bandwidth of 8 bits per frame. The author did not provide any test results on the real-world bandwidth of the covert channel. But since 802.11 networks operate at high frame rates, a generally higher bandwidth seems reasonable.

The covert channel can use the acknowledgement functionality of 802.11 networks to increase the reliability of the covert channel.

Zhao's covert channel creates additional association frames, as in case of our covert channel. Zhao argues that frequent association and reassociations are more common in a QBSS (*QoS Basic Service Set*), which would make the covert channel harder to detect. But the author does not provide measurements to support this claim.

Other than the increase in association frames, the covert channel does not create abnormal traffic, which could provoke errors in regular network participants. The author did not propose a statistical detection approach.

9 Conclusion

We introduced the first WiFi-based realization of covert channels that exploit artificial reconnections. The indirect nature of our two covert channels provides them with two benefits: range and anonymity of the participants. This is amplified by the usage of wireless networks as they are, by nature, broadcast networks.

Both of our covert channels suffer from similar problems, as they both cannot guarantee that a signal actually reaches the receiver, which is a common problem with blindly operating network covert channel senders.

With the detection approaches that we developed, we were able to detect both of our covert channels with minimal and none errors, respectively. But our covert channels do not interfere with the network usage of regular users, so it is unlikely that an administrator will actively look for them. More crowded networks might show different statistics, that hinder detection.

In future work, we plan to investigate whether networks show drastically different reconnection behavior when there are more devices in the network, which could change the detection results. We also plan to add the error-correcting and -detecting functionality of our covert channels and to investigate additional detection methods.

Acknowledgements. The authors like to thank Laura Hartmann for providing her comments on this paper.

References

1. Carrara, B., Adams, C.: Out-of-band covert channels-a survey. ACM Comput. Surv. (CSUR) **49**(2), 1–36 (2016). https://doi.org/10.1145/2938370
2. Carrega, A., Caviglione, L., Repetto, M., Zuppelli, M.: Programmable data gathering for detecting stegomalware. In: 2020 6th IEEE Conference on Network Softwarization (NetSoft), pp. 422–429 (2020). https://doi.org/10.1109/NetSoft48620.2020.9165537
3. Classen, J., Schulz, M., Hollick, M.: Practical covert channels for WiFi systems. In: 2015 IEEE Conference on Communications and Network Security (CNS), pp. 209–217, September 2015. https://doi.org/10.1109/CNS.2015.7346830
4. Fadlalla, Y.: Approaches to Resolving Covert Storage Channels in Multilevel Secure Systems. Ph.D. thesis, University of New Brunswick (1996)
5. Guri, M.: AIR-FI: Generating covert Wi-Fi signals from air-gapped computers. arXiv/CS.CR (2020)
6. Holloway, R., Beyah, R.: Covert DCF: A DCF-based covert timing channel in 802.11 networks. In: 2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems, pp. 570–579. IEEE (2011)
7. Krätzer, C., Dittmann, J., Lang, A., Kühne, T.: WLAN steganography: a first practical review. In: Proceedings of the 8th Workshop on Multimedia and Security. MM&Sec 2006, New York, NY, USA, pp. 17–22. Association for Computing Machiner (2006). https://doi.org/10.1145/1161366.1161371
8. Mazurczyk, W., Wendzel, S.: Information hiding: challenges for forensic experts. Commun. ACM **61**(1), 86–94 (2017). https://doi.org/10.1145/3158416
9. Mazurczyk, W., Wendzel, S., Cabaj, K.: Towards deriving insights into data hiding methods using pattern-based approach. In: Proceedings of the 13th International Conference on Availability, Reliability and Security. ARES 2018. ACM (2018). https://doi.org/10.1145/3230833.3233261
10. Mileva, A., Velinov, A., Hartmann, L., Wendzel, S., Mazurczyk, W.: Comprehensive analysis of MQTT 5.0 susceptibility to network covert channels. Comput. Secur. 104 (2021). https://doi.org/10.1016/j.cose.2021.102207
11. Szczypiorski, K.: HICCUPS: hidden communication system for corrupted networks. In: International Multi-Conference on Advanced Computer Systems, pp. 31–40 (2003)
12. The aircrack project: airmon-ng monitor mode (2019). https://www.aircrack-ng.org/doku.php?id=airmon-ng&s[]=monitor
13. Wendzel, S., Zander, S., Fechner, B., Herdin, C.: Pattern-based survey and categorization of network covert channel techniques. ACM Comput. Surv. **47**(3) (2015). https://doi.org/10.1145/2684195
14. Zander, S., Armitage, G., Branch, P.: Covert channels and countermeasures in computer network protocols. IEEE Commun. Mag. **45**(12), 136–142 (2007). https://doi.org/10.1109/MCOM.2007.4395378
15. Zhao, H.: Covert channels in 802.11e wireless networks. In: 2014 Wireless Telecommunications Symposium, pp. 1–5 (2014). https://doi.org/10.1109/WTS.2014.6834991