Security Evaluation of the Z-Wave Wireless Protocol

Behrang Fouladi
SensePost UK Ltd.
behrang@sensepost.com

Sahand Ghanoun
research@sahand.net

***Abstract*— The Z-Wave wireless communication protocol has been widely used in home automation and wireless sensors networks. Z-Wave is based on a proprietary design and a sole chip vendor. There have been a number of academic and practical security researches on home automation systems based on ZigBee and X10 protocols, however, no public vulnerability research on Z-Wave could be found prior to this work.**

In this paper, we analyze the Z-Wave protocol stack layers and design a radio packet capture device and related software named Z-Force to intercept Z-Wave communications. This device enables us to decode different layers of the Z-Wave protocol and study the implementation of encryption and data origin authentication in the application layer. We then present the details of a vulnerability discovered using Z-Force tool in AES encrypted Z-Wave door locks that can be remotely exploited to unlock doors without the knowledge of the encryption keys.

***Index Terms*— Z-Wave, Home Automation, Encryption, Data Authentication, Protocol Analysis**

I. Introduction

Home automation systems provide a centralized control and monitoring function for heating, ventilation and air conditioning (HVAC), lighting and physical security systems. The central control panel and various household devices such as security sensors and alarm systems are connected to each other to form a mesh network over wireless or wired communication links and act as a "smart home". The first generation of home automation systems such as X10 used existing power lines in the buildings as the communication medium, but as well as limited bandwidth, they were susceptible to signal loss and electrical interference. Wireless home automation systems overcome these limitations of the power line systems and provide easier expansion and interconnectivity of different devices. However, as the radio packets can be easily intercepted or injected by the attackers, protecting the confidentiality and integrity of wireless communication is of great importance in these systems. As such, security services such as key establishment, encryption, frame integrity protection and device authentication were included in the specifications of open wireless protocols such as ZigBee. Although these security services are built on top of the recognized cryptographic algorithms such as Advanced Encryption Algorithm (AES), successful attacks against them have been demonstrated that exploit the implementation vulnerabilities or insecure key management practices [1] [2].

Z-Wave protocol developed by Sigma Designs, Inc. provides packet encryption, integrity protection and device authentication services and is gaining momentum against ZigBee protocol with regards to home automation. This is partly due to interoperability of devices and shorter time to market on the vendor side. Another benefit is that it is less subjected to signal interference compared to the ZigBee protocol, which operates on the widely populated 2.4 GHz band shared by both Bluetooth and Wi-Fi devices. The protocol specification and software development kit (SDK) [4] are not open and only available to the device manufacturers (OEMs) who have signed an NDA with Sigma Designs. The SDK costs between 1500 to 3500 US dollars and the NDA prevents OEMs from disclosing the content of the SDK publically. The aim of this research is to build a low cost Z-Wave packet capture and injection tool which facilitates the security testing of home automation systems as well as aides vulnerability discovery of the Z-wave security products such as access control systems and door locks.

II. Z-Wave Protocol Stack Analysis

Z-Wave operates in the Industrial, Scientific and Medical radio frequency band (ISM). It transmits on 868.42 MHz (Europe) and 908.42 MHz (United States) frequencies designed for low-bandwidth data communications in embedded devices such as security sensors, alarms and home automation control panels. An open source implementation of Z-Wave protocol stack, open-zwave [4], is available but it does not support the security services as of yet.

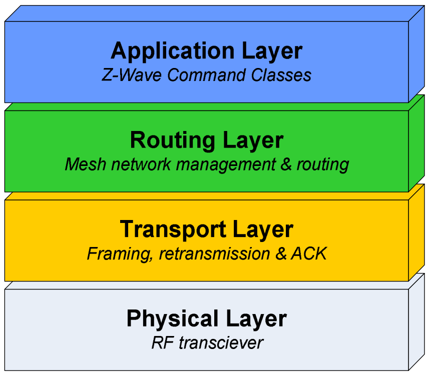


Figure 1 - Z-Wave protocol layers

The open-zwave software uses a Z-Wave controller device as the radio modem to communicate with the network nodes. The controller devices can only manage one Z-Wave network at a given time identified by a unique 32 bits Home ID. The Home ID value is written to the controller's Z-Wave chip by the manufacturer and cannot be changed by the controller software. This prevents controller devices from tuning into neighbor Z-Wave networks. Furthermore, the firmware of controller devices did not support promiscuous mode to intercept Z-Wave network packets of its own home network. These limitations led us to build our own Z-Wave radio modem for the purpose of this research as outlined in the following section.

A. Physical Layer

The ITU-T recommendation G.9959 [5] contains physical and MAC layer specifications for sub GHz radio communication including the Z-Wave protocol. It also outlines some aspects of the Z-Wave transport layer such as frame formats and Beam control which is necessary to communicate with Z-Wave door locks.

We used the Texas Instruments CC1110 radio transceiver kit [6] that comes with SmartRF Studio [7] software to determine the radio configuration parameters required by the CC1110 chip in order to receive and transmit Z-Wave packets with 9.6 Kbps and 40 Kbps data transfer rates.

9.6 kbps (Europe) Configuration	
Data rate	9.6 kbps
Symbol rate	19.2 kBaud
Center frequency	868.42 MHz
Modulation scheme	FSK
Coding	Manchester
Separation	40 KHz

Table 1 - 9.6 Kbps radio configurations

40 kbps (Europe) Configuration	
Data rate	40 kbps
Symbol rate	40 kBaud
Center frequency	868.40 MHz
Modulation scheme	FSK
Coding	NRZ
Separation	40 KHz

Table 2- 40 Kbps radio configurations

The above parameters were used to implement Z-Wave physical layer in CC1110 chip firmware and to develop a low cost Z-Wave packet interception and injection tool, which we named as Z-Force. It is worth noting that there were some discrepancies between RF signal properties specified in ITU.T G.9959 and those listed in the table above. In order to communicate with the devices available to us, we observed the modulation scheme for 9.6 kbps data rate needs to be G-FSK. In addition to that we noticed the symbol polarity is inverted in 40 kbps profile which means all the frame bits have to be inverted before transmission or after receipt of a frame. Some

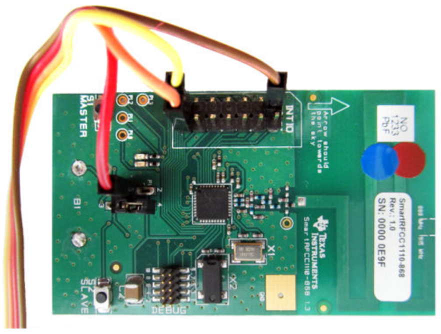


Figure 2 –Z-Force radio transceiver

other Texas Instrument transceivers can automatically perform this action in hardware. For more information on the individual CC1110 register settings, refer to Z-Force documentation. A screenshot of Z-Force user interface is included in Appendix B.

B. Transport Layer

Z-Wave transport layer is mainly responsible for retransmission, packet acknowledgment, waking up low power network nodes (Beaming) and packet origin authentication. Each Z-Wave frame in this layer contains the 32 bits Home ID that identifies the associated Z-Wave network, 8 bits source node ID, frame header that defines frame type (single-cast, multi-cast, routed) and control flag such as low power transmission, 8 bits payload length followed by the payload and the 8 bits frame checksum value. Transport layer relies on a frame checksum value to detect and discard erroneous frames. The Z-Wave protocol uses the following checksum algorithm from the ITU-T G.9959 standard:

```
BYTE GenCheckSum(BYTE *Data,BYTE Length){
   BYTE CheckSum = 0xFF;
   for (; Length > 0; Length--){
   CheckSum ^= *Data++;}
return CheckSum;}
```

An overview of Z-Wave transport frame format and fields is shown in figure 2. Frame retransmission occurs when an acknowledgement frame was not received from the destination node before the frame expiration time. Beam frames that are used to wake up battery powered Z-Wave nodes are controlled by the transport layer. Some battery powered Z-Wave devices including door locks needs to listen for the incoming commands from the network controller, but keeping their radio on will drain the battery quickly. In order to preserve the battery power, the device enters sleep mode and periodically turns on its radio, looking for beam frames. The transmitting node sends several back to back beam frames in 100ms intervals to ensure that the sleeping device will notice one of those frames when waking up and therefore it will keep its radio on to receive subsequent transmissions.

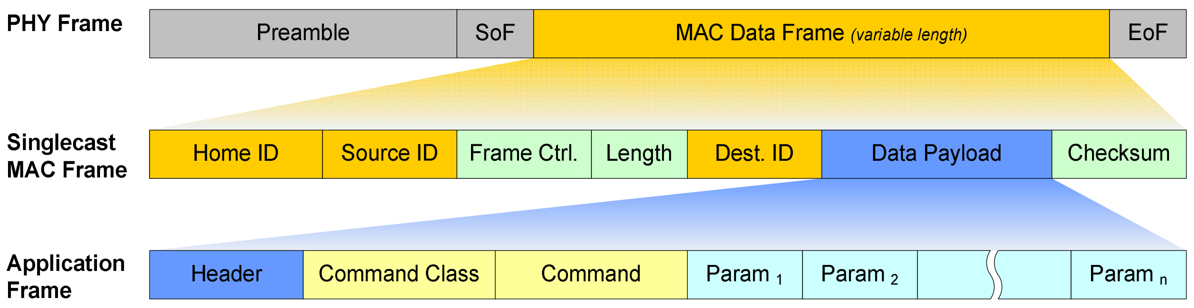


Figure 3 –Z-Wave frame format in different layers

When in secure transmission mode, an 8-byte frame authentication header is added to the end of the frame just before the frame checksum filed. No public specifications for the Z-Wave origin authentication header were available before our research. By using Z-Force tool and performing binary code analysis of a Z-Wave controller appliance firmware we found the cryptographic algorithm and parameters to calculate this header value. Z-Wave data origin authentication and encryption algorithms are discussed in detail in section III.

C. Network Layer

Z-Wave protocol forms a mesh network with one primary controller device and up to 232 nodes each of which can act as a packet repeater – with the exception of battery powered nodes – to route Z-Wave data even when the two communicating parties cannot establish a direct radio link between each other. In order to determine the best route to a destination node, each device in the Z-Wave network maintains a network topology that indicates all other devices in proximity. When device locations at home changes or they are removed from the network, this topology can become wrong and cause routing issues in the network. The Z-Wave protocol supports automatic topology discovery and healing to detect new network location and routes and optimize the routing tables. Although, Z-Wave routing mechanism and topology discovery might be subjected to attacks such as unauthorized modification of routing tables by rouge nodes [8], we did not perform security tests in network layer, as our research was focused on the encryption and origin authentication that are handled by transport and application layers.

D. Application Layer

This layer is responsible for parsing the frame payload and decoding the Z-Wave commands and supplied parameters. If the node was a Z-Wave controller device the decoded command and associated parameters will be forwarded to the controller software running on the home control computer or appliance. Otherwise they will be processed by the device firmware that is developed using Z-Wave SDK and running on the Z-Wave chip. As demonstrated in figure 2, the payload frame starts with one byte command header specifying that the command is single/multi or broadcast followed by the command class. Z-Wave command classes define device functionality such as door lock, alarm sensor, binary sensor, heating thermostat and etc. Each command class can consist of multiple commands. For instance, COMMAND_CLASS_ALARM (defined as 0x71) includes CMD_GET (0x04) and CMD_REPORT (0x05), the first of which is sent by the controller to the alarm to get the current state of the alarm and the second one is sent to the controller when the alarm is triggered. The open-zwave project has listed command class and command codes for various Z-Wave devices. It's important to note that this list does not contain the commands processed by Z-Wave firmware such as network topology discovery or network inclusion and exclusion commands.

Using the Z-Force tool we noticed that when home network nodes communicate over secure Z-Wave, the frame payload is encrypted and followed by 8 bytes authentication field. The feature list of the Z-Wave door lock stated that it was using 128 bits AES encryption, but we observed that the encrypted frame payload length is less than the cipher block length (128 bits). This suggested that the device has AES algorithm in one of Cipher Feedback (CFB) or Output Feedback (OFB) modes that can convert block ciphers to variable length stream ciphers. In the following section we validate this hypothesis as well as the frame authentication algorithm.

III. VULNERABILITY ANALYSIS

In order to discover design or implementation vulnerabilities in Z-Wave secure communication, it was essential to uncover the details of frame encryption and authentication algorithms. We analysed the Z-Wave radio frames and firmware binary of a home automation appliance in the following scenarios:

a) Door lock inclusion into the Z-Wave network for the first time: encryption key exchange takes place between the controller appliance and the door lock to establish a shared symmetric key.

b) Sending lock/unlock commands to the Z-Wave door lock: the command is encrypted using the established encryption key and authentication value is also appended to the frame.

c) Door lock inclusion after controller appliance factory rest: The factory reset will erase previously established key from the appliance but the door lock will still hold the old encryption key.

Analysis of scenario (a) indicated that the encryption key is not exchanged in clear text. This key is generated using the hardware based pseudo random number generator (PRNG) on the Z-Wave chip and then is encrypted by using a hard coded temporary default key in chip's firmware before being sent to the door lock. The value of this temporary key was found to be 16 bytes of zero. Although an attacker could intercept the encrypted key exchange frame, and decipher it using the hard-coded key this attack vector was not interesting to us, as the key exchange only happens at system initial setup time or re-installation that limits the attack time window. Furthermore Z-Wave devices can switch their radio to low power transmission mode during key exchange process to make packet interception attack much more difficult.

After successful exchange of the network key (K_n) between the controller appliance and the door lock, they both derive two new 128-bit keys: frame encryption key (K_c) and data origin authentication key (K_m) by using AES encryption in ECB mode as following:

$$K_c = AES\text{-}ECB_{K_n}(Passwd_c)$$
$$K_m = AES\text{-}ECB_{K_n}(Passwd_m)$$

$Passwd_c$ and $Passwd_m$ values were found to be 16-byte values hardcoded into the Z-Wave firmware as highlighted in Appendix A.

Z-Wave data origin authentication is based on the cipher block chaining message authentication code (CBC-MAC) technique that can calculate a message authentication code (MAC) from a block cipher algorithm such as AES. The generated MAC value ensures that the Z-Wave frame is not tampered with or corrupted during the transmission (data integrity) and that it has been sent by the node claiming to be the message source (origin authentication). In order to prevent packet replay attacks, 64-bit nonce values generated by the Z-Wave chip's PRNG are used during MAC calculation as described by the following formula:

$$MAC = AES\text{-}CBCMAC_{K_m}(\mathrm{IV}, \mathrm{SH}||\mathrm{SRC}||\mathrm{DST}||\mathrm{LEN}||\mathrm{C})$$

Initialization vector (IV) is 16 bytes long with bytes zero to seven set by the PRNG and bytes 7 to 15 set to the nonce value received from the destination node. The security header (SH) is a one byte value that determines the type of secure messages: nonce request (0x40), nonce reply (0x80) and encrypted data (0x81). SRC and DST are the source and destination node IDs, LEN is the encrypted payload length in bytes and finally C is the encrypted payload bytes that are generated by using AES algorithm in OFB mode:

$$C = AES\text{-}OFB_{K_c}(\mathrm{IV}, \mathrm{P})$$

P is the plain text variable size payload that contains Z-Wave command header, class, ID and parameters.

With the above knowledge of Z-Wave encryption and authentication, we developed a door lock key exchange module for the Z-Force tool that enabled us to control all steps of the key exchange protocol (Figure 3), run it with our own network key and observe the responses from the door lock. After running a few tests including scenario (c) that was mentioned earlier in this section, we identified a critical protocol implementation vulnerability that could allow an attacker to reset the established network key on a target Z-Wave door lock to a known value of his choice and then issue unauthorised commands.

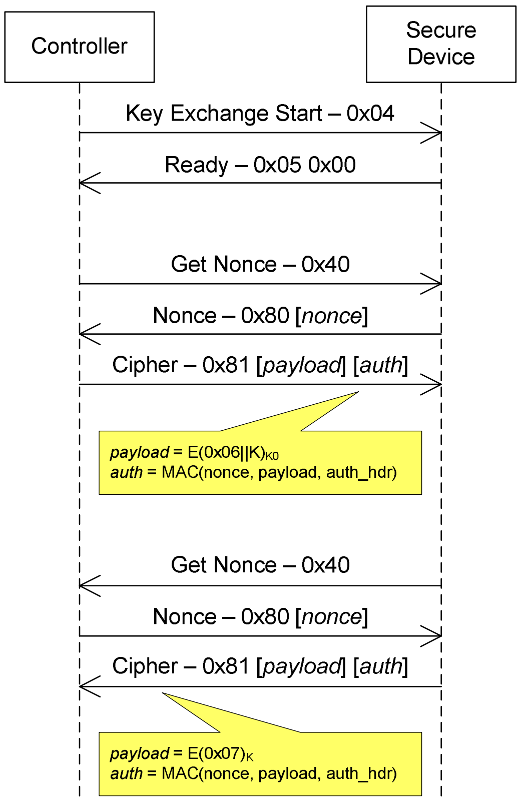

Controller
Secure Device
Key Exchange Start – 0x04
Ready – 0x05 0x00
Get Nonce – 0x40
Nonce – 0x80 [nonce]
Cipher – 0x81 [payload] [auth]
payload = E(0x06||K)K0
auth = MAC(nonce, payload, auth_hdr)
Get Nonce – 0x40
Nonce – 0x80 [nonce]
Cipher – 0x81 [payload] [auth]
payload = E(0x07)K
auth = MAC(nonce, payload, auth_hdr)

Figure 4 – Key exchange protocol

The root cause of this issue was lack of state validation in the key exchange protocol handler programmed in the Z-Wave door lock firmware. This handler code is called when the door lock receives the key exchange start packet which payload is 0x98 0x04 from the home controller device. At this point, the handler function needs to load a shared encryption key so that it can decrypt the rest of key exchange packets received from the controller and be able to encrypt its response packets. However before using the hardcoded temporary key (16 bytes of zero) for this purpose, the door lock should check the state of the current shared key in its EEPROM and load the previously provisioned network key if one already exists. On contrary, we discovered that the Z-Wave door lock does not perform this important state validation. As a result, a remote attacker who has detailed knowledge of the key exchange protocol and is in possession of a Z-Wave packet injection tool similar to Z-Force can force the target Z-Wave door lock to overwrite its current shared network key with that of the attacker. This would enable the attacker to send secure Z-Wave messages to perform unauthorised actions such as unlocking the door or changing users' PIN codes. We successfully demonstrated this attack against a 128 bit AES encrypted door lock using Z-Force kit that was configured to operate in 868.42 MHz frequency and 40Kbps data transfer rate.

I. Impact

Successful exploitation of the aforementioned vulnerability can enable an attacker to take full control of the affected Z-Wave door locks and possibly other secure devices in a building. If the compromised door lock attempts to send "Door is Open" event to the controller after being unlocked by the attacker, the received packet will contain an invalid frame authentication field and will be discarded by the controller software. Therefore, the home residents or building manager will not be alerted about the intrusion.

II. Conclusion

We have analysed Z-Wave proprietary protocol and uncovered the details of its encryption, authentication and key exchange protocols. Based on this knowledge we developed a low cost Z-Wave packet interception and injection tool that enabled us to perform vulnerability discovery on the Z-Wave door locks. Using this tool, we have demonstrated an implementation vulnerability in Z-Wave key exchange protocol that could be exploited to take full control of a target Z-Wave door lock by only knowing the Home and node IDs of the target device, both of which can be identified by observing the Z-Wave network traffic over a short period of time due to the frequent polling of devices in a Z-Wave network, for example to get status or battery level of the device. This vulnerability was not due to a flaw in the Z-Wave protocol specification, but because of an implementation error in disabling the use of temporary key after initial network key exchange during inclusion of a node to the network.

We have communicated the details of this vulnerability to the vendor who has conducted a security review of Z-Wave specification and SDK to ensure that they cover correct handling of the discovered vulnerability. Finally, Sigma Designs has taken action to prevent such implementation flaws to reach the market in the future by adding additional security test cases to the certification test suite.

We also recommend Z-Wave device manufacturers to examine their firmware code for this vulnerability. Due to the flexibility of Z-Force tool, it can also be used to identify other vulnerabilities for example memory corruptions in closed source Z-Wave firmware via fuzz testing.

Acknowledgment

We would like to thank Sigma Designs for their urgent response and verification of the reported vulnerability.

References

[1] Kennedy, D., & Simon, R. (2011). Pentesting over Power lines. *Defcon 2011.*

[2] Wright, J. (2011). Practical ZigBee Exploitation Framework. *toorcon 2011.*

[3] Sigma Designs. (n.d.). *Z-Wave development Kit.* Retrieved June 2013, from Sigma Designs public web site: http://www.sigmadesigns.com/uploads/documents/zwave_dev_kit_br.pdf

[4] OpenZwave. (n.d.). Retrieved June 2013, from open-zwave Google code site: https://code.google.com/p/open-zwave/

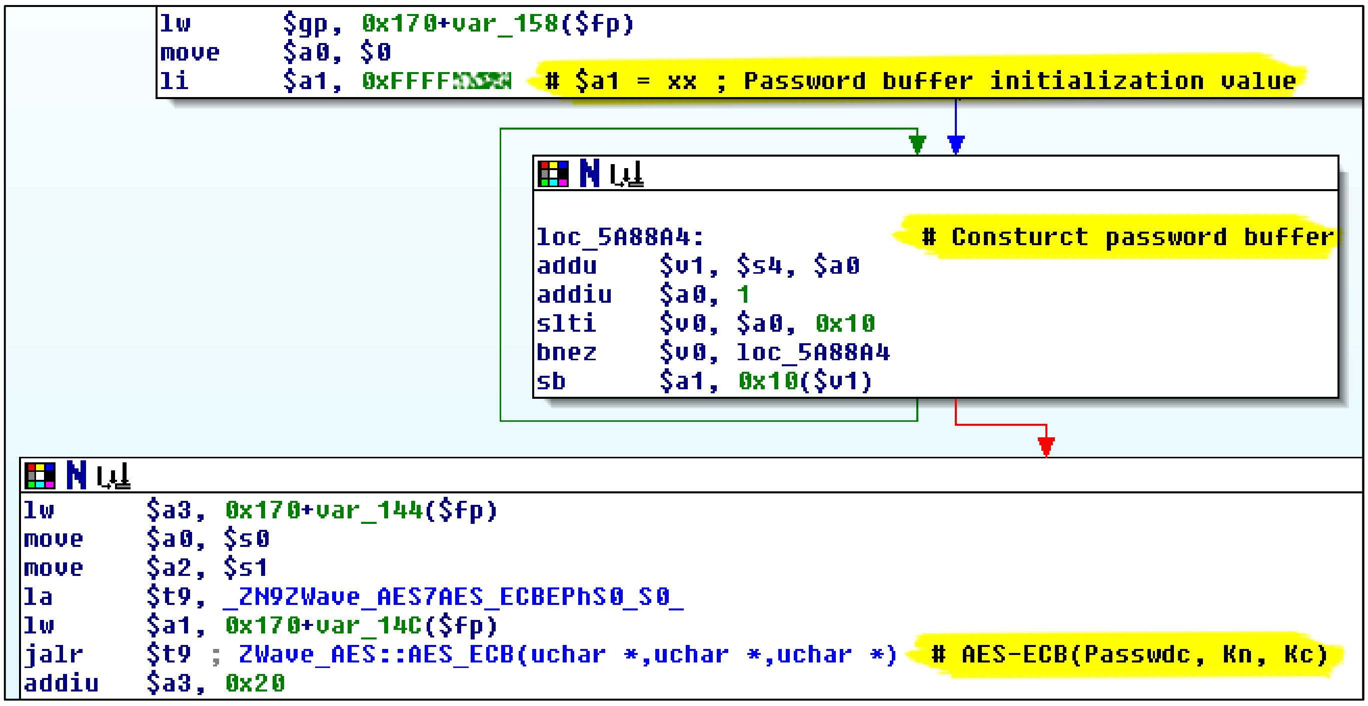
[5] International Telecommunication Union (ITU). (2012). G.9959 : Short range narrow-band digital radiocommunication transceivers - PHY and MAC layer specifications.

[6] Texas Instruments. (2009, December). *CC1110 Mini Development Kit.* Retrieved from http://www.ti.com/tool/cc1110dk-mini-868

[7] Texas Instruments. (2013, May). *SmartRF Studio.* Retrieved from http://www.ti.com/tool/smartrftm-studio

[8] Morais, A., & Cavalli, , A. (2011). Route Manipulation Attack in Wireless Mesh Networks. *Advanced Information Networking and Applications (AINA).*

APPENDIX A. KEY DERIVATION USING HARD-CODED PASSWORD IN A Z-WAVE FIRMWARE

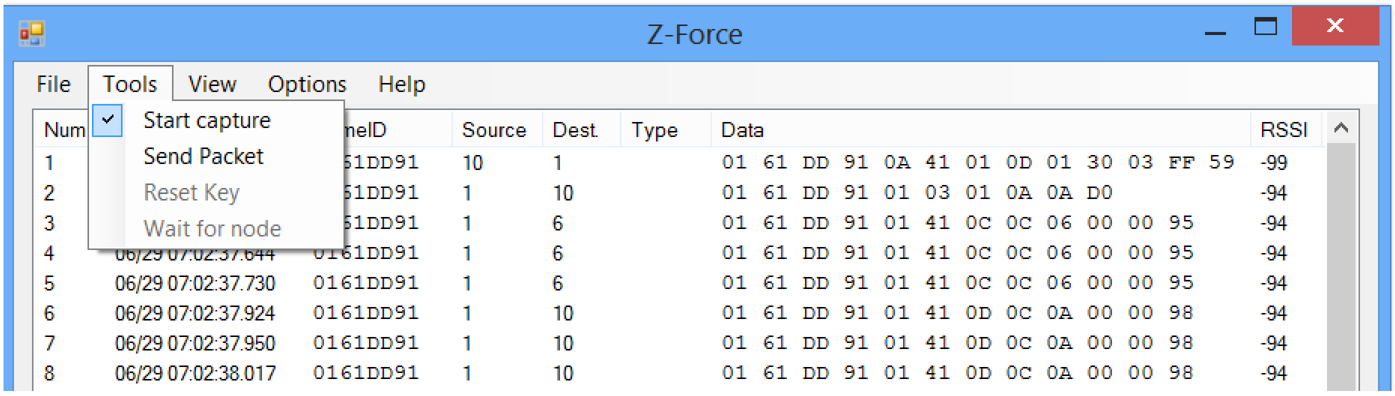


```
lw      $gp, 0x170+var_158($fp)
move    $a0, $0
li      $a1, 0xFFFF[redacted]  # $a1 = xx ; Password buffer initialization value

loc_5A88A4:                  # Consturct password buffer
addu    $v1, $s4, $a0
addiu   $a0, 1
slti    $v0, $a0, 0x10
bnez    $v0, loc_5A88A4
sb      $a1, 0x10($v1)

lw      $a3, 0x170+var_144($fp)
move    $a0, $s0
move    $a2, $s1
la      $t9, _ZN9ZWave_AES7AES_ECBEPhS0_S0_
lw      $a1, 0x170+var_14C($fp)
jalr    $t9 ; ZWave_AES::AES_ECB(uchar *,uchar *,uchar *)  # AES-ECB(Passwdc, Kn, Kc)
addiu   $a3, 0x20
```

APPENDIX B. Z-WAVE FRAMES INTERCEPTED BY Z-FORCE



Num		meID	Source	Dest.	Type	Data	RSSI
1		51DD91	10	1		01 61 DD 91 0A 41 01 0D 01 30 03 FF 59	-99
2		51DD91	1	10		01 61 DD 91 01 03 01 0A 0A D0	-94
3		51DD91	1	6		01 61 DD 91 01 41 0C 0C 06 00 00 95	-94
4	06/29 07:02:37.644	0161DD91	1	6		01 61 DD 91 01 41 0C 0C 06 00 00 95	-94
5	06/29 07:02:37.730	0161DD91	1	6		01 61 DD 91 01 41 0C 0C 06 00 00 95	-94
6	06/29 07:02:37.924	0161DD91	1	10		01 61 DD 91 01 41 0D 0C 0A 00 00 98	-94
7	06/29 07:02:37.950	0161DD91	1	10		01 61 DD 91 01 41 0D 0C 0A 00 00 98	-94
8	06/29 07:02:38.017	0161DD91	1	10		01 61 DD 91 01 41 0D 0C 0A 00 00 98	-94