Dissecting Customized Protocols: Automatic Analysis for Customized Protocols based on IEEE 802.15.4

Kibum Choi
Korean Advanced Institute of Science and Technology
kibumchoi@kaist.ac.kr

Yunmok Son
Korean Advanced Institute of Science and Technology
yunmok00@kaist.ac.kr

Juhwan Noh
Korean Advanced Institute of Science and Technology
juwhan@kaist.ac.kr

Hocheol Shin
Korean Advanced Institute of Science and Technology
h.c.shin@kaist.ac.kr

Jaeyeong Choi
Korean Advanced Institute of Science and Technology
go1736@kaist.ac.kr

Yongdae Kim
Korean Advanced Institute of Science and Technology
yongdaek@kaist.ac.kr

ABSTRACT

IEEE 802.15.4 is widely used as lower layers for not only well-known wireless communication standards such as ZigBee, 6LoWPAN, and WirelessHART, but also customized protocols developed by manufacturers, particularly for various Internet of Things (IoT) devices. Customized protocols are not usually publicly disclosed nor standardized. Moreover, unlike textual protocols (e.g., HTTP, SMTP, POP3.), customized protocols for IoT devices provide no clues such as strings or keywords that are useful for analysis. Instead, they use bits or bytes to represent header and body information in order to save power and bandwidth. On the other hand, they often do not employ encryption, fragmentation, or authentication to save cost and effort in implementations. In other words, their security relies only on the confidentiality of the protocol itself.

In this paper, we introduce a novel methodology to analyze and reconstruct unknown wireless customized protocols over IEEE 802.15.4. Based on this methodology, we develop an automatic analysis and spoofing tool called WPAN automatic spoofer (WASp) that can be used to understand and reconstruct customized protocols to byte-level accuracy, and to generate packets that can be used for verification of analysis results or spoofing attacks. The methodology consists of four phases: packet collection, packet grouping, protocol analysis, and packet generation. Except for the packet collection step, all steps are fully automated.

Although the use of customized protocols is also unknown before the collecting phase, we choose two real-world target systems for evaluation: the smart plug system and platform screen door (PSD) to evaluate our methodology and WASp. In the evaluation, 7,299 and 217 packets are used as datasets for both target systems, respectively. As a result, on average, WASp is found to reduce entropy of legitimate message space by 93.77 % and 88.11 % for customized protocols used in smart plug and PSD systems, respectively. In addition, on average, 48.19 % of automatically generated packets are successfully spoofed for the first target systems.

WiSec'16, July 18 - 20, 2016, Darmstadt, Germany.
 ISBN 978-1-4503-4270-4/16/07...$15.00
DOI: http://dx.doi.org/10.1145/2939918.2939921.

Keywords

Wireless spoofing attacks; customized PAN protocol; automatic protocol reversing

1. INTRODUCTION

In an Internet of Things (IoT) environment, many devices communicate with one another using various wireless communication protocols. Because most applications operate in a personal area such as a home, office, or hospital, wireless personal area network (WPAN) technologies, such as ZigBee, Bluetooth, and Z-Wave, are essential for implementing IoT systems.

In particular, among various WPAN protocols, IEEE 802.15.4 has been widely used for lower layer protocol for several standards such as ZigBee, 6LoWPAN, WirelessHART, and MiWi. IEEE 802.15.4 is a standard that specifies the physical (PHY) layer and medium access control (MAC) layer for low-rate wireless personal area networks (LR-WPAN) [15]. It operates on one of three frequency bands: 868–868.6 MHz, 902–928 MHz, or 2,400–2,483.5 MHz. It can provide up to 250 kbps at 10 m distance with low power and low cost. Because of the requirements of power and cost efficiency, IEEE 802.15.4 has a simple structure.

Notably, for many IoT systems and some safety-critical control systems, proprietary protocols (i.e., customized protocols) are used on top of IEEE 802.15.4. These include smart metering systems and platform screen door (PSD) systems. Although details of these customized protocols are unknown, they can be vulnerable to sniffing and spoofing attacks, because of the low power characteristics (i.e., simplicity and short message length) of IEEE 802.15.4 and the shared medium in wireless communication without message encryption and authentication. Through these attacks, private information such as electricity usage, life patterns, or medical records can be stolen, and overcharge of electricity bills, failure of power supply-and-demand control, wrong medical treatment or undesirable control of systems can occur. Furthermore, they can control target devices for their own purpose: for example, switching on and off, injecting malicious data into, and incapacitating the devices.

IEEE 802.15.4's MAC layer supports encryption that is one of big challenges for analyzing protocols. However, in this study, customized protocols using encryption are not considered because customized protocols do not usually use it for the low power characteristics. Even when no messages are encrypted, some challenges remain in analyzing and reconstructing customized protocols. First, obtaining and reverse engineering binary files that implement customized protocols are difficult. Unless the firmware of a target device is

opened on the Internet, to obtain firmware, physical access to the device, and even in such a case, desoldering a memory or processor chip may be required. Even after obtaining the firmware, manual protocol reverse engineering is inefficient and time-consuming. While automatic reverse engineering has garnered considerable attention these days, it only supports popular architectures such as x86 or ARM. Second, wireless custom protocols use binary formats because of low-power consumption requirements, instead of using text data such as keywords. Because packet length is directly associated with power consumption, these protocols use fields that are as short as possible to represent various fields and commands. Finally, even after reverse engineering the protocol, evaluating analysis results is difficult due to the absence of ground truth.

The goal of this study is to analyze and reconstruct packet formats for customized protocols built on top of IEEE 802.15.4 standard. In addition, we aim to generate spoofing packets for the protocol. For these goals, we designed a novel methodology and developed an automatic analysis tool named WPAN automatic spoofer (WASp) based on the methodology. Our analysis and reconstruction methodology consists of four phases: packet collection, packet grouping, protocol analysis, and packet generation. Packet collection refers to the manual wireless channel sniffing process that must be conducted in carefully controlled conditions. This directly affects the effectiveness of analysis because critical factors such as the number of collected packets, the number of transceivers, and expected operations are bounded in this phase. The other three phases are fully automated. Initially, WASp groups packets according to crucial information from packet headers. For each packet group, the tool analyzes MAC layer data reuse (e.g., address field), byte-level entropy, the range of each byte column, and the existence of a cyclic redundancy check (CRC). In the second step, the tool combines results of all tests and generates scored analysis reports for every packet group using our scoring algorithm. Finally, for reports with a high score, it generates feasible packet lists for spoofing.

Our system is closely related to that of Netzob [4], which aims to analyze and cluster the protocol by considering semantic information embedded in TCP/IP layer. While both Netzob and WASp target customized protocols, the main difference between them is the layer they are targeting. Netzob targets protocols on top of the TCP/IP layer, whereas WASp targets protocols built on top of the MAC layer of IEEE 802.15.4 which contains much less information then the TCP/IP layer. In other words, Netzob mainly relies on semantic information existing in TCP/IP. Therefore, we believe critical changes for Netzob are required to achieve the same goal as that of WASp.

To evaluate WASp, we used two commercial target systems, including a smart plug system for home use and a PSD system in a subway system. WASp automatically analyzes collected packets based on characteristics of customized protocols by deriving various parameters including n-gram and entropy. The results of our evaluation show that, for a smart plug system, the tool reduces an average of 93.77 % of the entropy for a customized protocol and provides possible rules to construct spoofing payload at a 48.19 % average success rate. In other words, we can control the power supply of devices connected to a smart plug using automatically generated spoofing packets. We could not perform our attack on a PSD system, because of legal and safety concerns. However, an average of 88.11 % of the entropy is reduced for PSD's customized protocol. Note that all vulnerabilities are responsibly disclosed to the agencies in advance. Our study contributes the following:

- We derive a general analysis methodology for customized protocols on top of IEEE 802.15.4, which is far different from that of typical network packet analysis.
- We build an automatic protocol analyzer and prove the possibility of automated protocol reverse engineering against wireless customized protocols.
- We show that taking control of applications using automatically generated spoofing packets by our tool is possible.

The remainder of the paper is organized as follows. Section 2 summarizes existing research related to automatic protocol analysis and compares these studies to our own. Section 3 provides information about customized protocols on top of IEEE 802.15.4. Section 4 explains the concept of our design to analyze target protocols and the basic concept of our automatic analysis. The detailed implementation and algorithms of our methodology are described in Section 5. In Section 6, we present the results of automatic analysis and spoofing on two commercial systems. We discuss limitations of this study in Section 7. We conclude the study in Section 8.

2. RELATED WORK

Because of the massive increase in the number of network protocols, including that of malware and botnets, security researchers have tried to automate protocol reverse engineering. Automated protocol reverse engineering has been used for deep packet inspection, intrusion detection or prevention, and automatic packet generation for fuzzing and replay attacks. We can classify it into two major categories: program-based and trace-based analyses. In addition to these two, protocol fuzzing is somewhat related to our automatic spoofing packet generation.

Studies on program-based analysis have attempted to analyze target protocols on a binary program and some of its input traces. For example, studies have analyzed processes that occur as a result of input messages to a server or client programs. By contrast, studies on trace-based analysis reverse engineer only with network traces of target protocols. Finally, studies on protocol fuzzing tests a target system by injecting many packets that resemble legitimate ones.

2.1 Program-Based Analysis

First, program-based research analyzes customized protocols based on program implementation. RolePlayer [9] and AutoFormat [17] attempt protocol parsing using a context-aware monitored execution. Polyglot [6], Tupni [10], Reformat [23], Dispatcher [5], and Prospex [7] can be classified as a taint analysis using semantic program information. Because these programs use keywords or separators to identify fields, analysis is much simpler. In addition, they can monitor program flow according to specific message inputs. These cause the relation between parsed fields and their real meaning to be clearer than when employing network trace only approaches. This paper focuses on trace-based analysis, and program-based analysis is out-of-scope for this paper. Nonetheless, as we discussed in the introduction, program-based analysis is difficult and time-consuming.

2.2 Trace-based Analysis

Trace-based research analyzes network packet traces without any prior knowledge about the program that generates the packets, which can be divided into two types. The first type is reverse engineering against textual protocols such as HTTP or SMTP. Discoverer [8], Reverx [1], ProDecorder [22], SANTaClass [20], and ProWord [25] fall into this type. These programs try to extract fields by parsing keywords. For example, keywords such as "GET" and "POST" in HTTP have specific delimiters or a similar interval. However, wireless customized protocols mostly use binary data for energy efficiency. Therefore, keyword parsing methods from these kind of

Table 1: Comparison studies closely related to WASp

	Target	Purpose	Characteristics
PROVEX [19]	Botnet C&C traffic	Botnet detection	• Byte value distribution • Signature creation
ProGraph [14]	General traffic	Analysis and classification	• Graph signature extraction
Netzob [4]	General traffic	Analysis and classification	• Need semantic information
FieldHunter [2]	General traffic	Analysis and classification	• Partial field extraction

textual protocol analysis are difficult to be used for reverse engineering wireless customized protocol.

The second type studies against binary protocols. PROVEX [19] extracts botnet C&C traffic signature based on byte distribution, and ProGraph [14] tries to infer protocol message formats by exploiting intra-packet dependency derived from constructing a graphical model. However, both of these programs passively analyze given network traces, and limit their application to either classification of network traces or generation of network signature for protocol detection. FieldHunter [2] and Netzob [4], can be used for both binary and textual formatted protocols by utilizing semantic information. However, FieldHunter extracts only partial fields such as message type and length, which can be distinguished by means of statistical characteristics, and skips unidentified fields.

As previously discussed, Netzob is closely related to our system, WASp. First, it uses not only syntactic information (e.g., encoding, checksums, and IP addresses), but also semantic information (e.g., file read/write and listing directories) to extract a message structure with high accuracy. Second, during analysis, it filters out samples considered to be noisy in order to improve clustering output. Finally, it clusters acquired target samples with available syntactic and semantic information, and infers important characteristics of sub-message fields such as value, offset, and size. However, Netzob is considerably different from WASp regarding the base information it uses and its output. First, Netzob depends on higher layer information (i.e., network and transport layers) compared to WASp. Specifically, it refers to well-known field values (e.g., IPv4 addresses, TCP port numbers) of specific implementations of such layers (i.e., IP and TCP) to infer the structure of the target protocol. Therefore, Netzob would require an overhaul to analyze IEEE 802.15.4 customized protocols that WASp targets. Second, Netzob infers only the structure of the target protocol and does not generate packets that can be directly spoofed to targets. It lacks the packet generation engine present in WASp, which automatically assembles spoofing packets based on analyzed field information. Table 1 summarizes comparison between these four systems.

2.3 Protocol Fuzzing

Protocol fuzzing is a type of fuzz testing that generates invalid or unexpected packets and injects them into a target system. An effective fuzzer for security testing must be designed with deep understanding of the target protocol as well as require automatic packet generation for security attacks.

For example, by collecting transmitted packets from a target system, SECFUZZ [21] tests known security protocols such as internet key exchange (IKE). It generates packets based on the assumption that the fuzzer has necessary information related to decryption such as encryption keys and nonces. In particular, it finds vulnerabilities automatically by attaching a dynamic memory analytical tool. AspFuzz [16] and regression finite state machine (RFSM) fuzzer [26] test state-aware complex protocols, and they establish their respective state machine models to represent the states and transitions of a target protocol. Thus, they can generate packets using anomalous data as well as the order of those packets. However, these fuzzers are not adequate to analyze customized protocols based on IEEE 802.15.4. They cannot analyze customized protocols because they require prior knowledge of target protocols. AutoFuzz [13] constructs a generic message sequence (GMS), an array list of static and variable data fields. It then generates packets by fixing data in static data fields and changing data in variable data fields. Although AutoFuzz does not require prior knowledge, its targets are textual protocols. Therefore, applying it to wireless binary-based customized protocols is difficult.

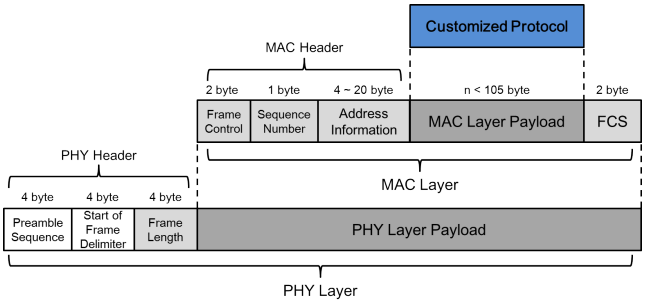

Figure 1: IEEE 802.15.4 data frame format (PHY and MAC layers) with customized protocol as an upper layer

3. BACKGROUND

In this section, we explain the basic information of the IEEE 802.15.4 architecture and data frame format. We also define a customized protocol based on IEEE 802.15.4 as our target protocol.

3.1 IEEE 802.15.4

IEEE 802.15.4 is a standard that defines lower layer protocols for WPAN communications that use frequency bands of 868–868.6, 902–928, or 2,400–2,483.5 MHz. It mainly focuses on low-speed and low-power communication between wireless devices. Although WiFi offers high-speed solutions, IEEE 802.15.4 requires much less power and focuses on home-range devices. Essentially, it supports a communication range of 10 m, and a transfer rate of 250 kbps. Even if low power consumption is one of its main features, extremely low manufacturing and operation costs are also the chief features of IEEE 802.15.4. Furthermore, technological simplicity and flexibility make it adaptable to various wireless communication protocols. For example, well-known protocols, such as ZigBee, 6LoWPAN, WirelessHART, have been developed based on the IEEE 802.15.4 standard.

3.1.1 Architecture

The IEEE 802.15.4 architecture consists of only two layers: PHY and MAC. The PHY layer defines the physical specifications to transmit and receive radio frequency (RF) signals through a physical transmission medium. The MAC layer provides a reliable link between two nodes and is responsible for the following: encoding digital bits into packet frames for transmission, decoding them in order to receive frames, and controlling access to data in a network. This architecture is used not only for the lower layer of well-known wireless communication standards, but also for publicly unknown customized protocols designed by device manufacturers.

3.1.2 Data Frame Format

The data frame format of PHY and MAC layers of IEEE 802.15.4 is depicted in Figure 1. A PHY layer frame includes a PHY payload,

which is a MAC layer frame that contains a MAC header, MAC payload, and frame check sequence (FCS). The MAC contains a frame control field (FCF), data sequence number, and address information. The MAC payload is the actual payload to be transmitted, and FCS is usually implemented as a CRC to detect common errors caused by noise in wireless communication channels.

As depicted in Figure 1, MAC payload is the real target field that contains a protocol customized by various manufacturers for their systems. To analyze this customized protocol, information in a MAC header and FCS is used. First, FCF in a MAC header contains important bitmap information such as packet type, security enabled status, acknowledgment (ACK) request status, and addressing mode. If the security enabled bit is not set, no encryption is applied to the upper layer, which means the protocol can be analyzed by an attacker. Address fields and FCS are utilized for address reuse detection and noise filtering, respectively.

3.2 Customized Protocol

The main target in this study is customized protocols designed by manufacturers themselves. Customized protocols can be identified by simply using deep packet inspection (DPI) tools such as Wireshark [12]. For example, if Wireshark indicates that an unknown protocol is a member of the IEEE 802.15.4 family of protocols (e.g., ZigBee and 6LowPAN) and payloads of their packets are abnormally parsed, then this unknown protocol can be considered as a customized protocol.

These customized protocols are widely used for various IoT devices. In this study, a customized protocol is defined as an upper layer protocol over well-defined PHY and MAC layers (i.e., IEEE 802.15.4). Because many wireless communication applications for IoT have several functionalities, complex protocols are not required. Thus, some manufacturers often use customized protocols instead of following existing standard protocols such as ZigBee and WirelessHART. In some cases, certain IoT devices seem to use ZigBee or some other standard WPAN protocols, but we found that they actually use customized protocols on top of IEEE 802.15.4. However, before capturing packets, recognizing whether a customized protocol is used in a wireless communication system is impossible. Therefore, finding proper target system for evaluation of WASp in Section 6 was difficult.

Customized protocols are not known publicly and vary according to devices or manufacturers. Therefore, analyzing customized protocols and generalizing them for automation are challenging. However, in many cases, customized protocols have several characteristics that enable unknown protocol analysis and spoofing attacks. These characteristics are as follows:

- MAC layer data such as source and destination addresses can be reused, because producing their own address system is burdensome.
- Common patterns such as ASCII bytes, periodically increasing or decreasing bytes, and sequential bytes are embedded.
- No authentication and poor packet integrity check are implemented. (e.g., using well-known CRC algorithms, checking only the increment of sequence numbers)
- Some bytes can be used without any alteration for spoofing attacks, because they are filled with fixed data regardless of the operation a target system.

Based on these characteristics, manually analyzing unknown customized protocols is possible. However, manual protocol reverse engineering is time-consuming and requiring new analysis for new targets.

Assumptions. To clarify the scope of our analysis, we assumed the following. First, the main purpose of protocol customization is neither flexibility nor convenience, but rather efficiency and low-power consumption. For example, a command field having textual data such as a keyword usually requires more than one byte, but a binary data field that corresponds to predefined commands can be represented in one byte. Second, packets are not encrypted. While some of the standards support encryption, customized protocols we have seen have not been using encryption. Third, packets are not fragmented, as fragmentation increases cost and the amount of power consumption. Finally, an attacker has no access privileges to the target system. From packet sniffing to spoofing attack, all processes are performed remotely.

4. ANALYSIS DESIGN

The final goal of this study is to analyze and reconstruct unknown wireless customized protocols over IEEE 802.15.4 for spoofing attacks at a high success rate. As a building block, we develop an analytical procedure for wireless spoofing attacks consisting of three phases inspired by the inductive approach. In this section, we provide an overview of our analytical procedure in three phases.

4.1 Overview

Figure 2 represents an overview of our analytical procedure to spoof wireless communication systems using the customized protocols mentioned in Section 3. Target devices employ IEEE 802.15.4 as PHY and MAC layers and an unknown customized protocol as an upper layer. Physical signals containing packets of a customized protocol are transmitted through a wireless channel (i.e., the air) that is publicly open. Because the target protocol is unknown, the starting point of analysis involves sniffing (i.e., collecting packets) by listening to RF signals in the proper wireless channel. For sniffing, we require an appropriate tool with an RF signal receiving and digital processing capability that can handle PHY and MAC layers such as KillerBee [24] and universal software radio peripheral (USRP) [11] with "gr-ieee-802-15-4" GNU radio module [3] that fully support the IEEE 802.15.4 standard.

After collecting enough packets, attackers can reverse-engineer the customized protocol. Protocol reverse engineering is the most important and challenging part. An attacker needs to distinguish each field and determine their meanings in raw byte sequences with logical evidence or intuition. This process is extremely tedious, time-consuming, and even prone to errors. The more that fields are revealed, the more efficient the spoofing attack is, because employing brute-force at every field is practically impossible.

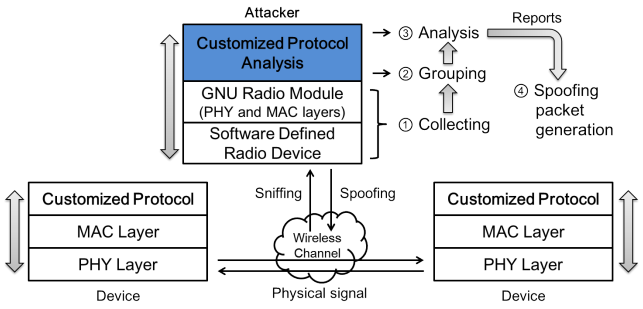

Figure 2: Overview of our analytical procedure for wireless communication sniffing and spoofing attacks against devices using customized protocols

In this study, we mainly focus on this protocol reverse engineering to generate possible spoofing packets efficiently. Therefore, our analytical procedure is built upon PHY and MAC layers as an lower layer. Generated spoofing packets can then be transmitted by the signal processing tool such as KillerBee and USRP.

4.2 Analytical Procedure

An inductive approach consists of observation, pattern, and theory. Similarly, protocol reverse engineering can be divided into three phases to make a spoofing attack possible for an IEEE 802.15.4-based customized protocol: collecting, grouping, and analysis as shown in Figure 2. To understand the meaning of each field in a customized protocol, controlling the variance of packets in predictable conditions in the first two phases is necessary. We strongly believe that this three-step procedure is a generic approach for IEEE 802.15.4-based customized protocol reverse engineering.

4.2.1 Collecting Phase

The first step in the collecting phase is to identify a communication channel. Because IEEE 802.15.4 has only 26 channels, by brute-forcing, we can identify the active channel that the target system uses. The collecting phase is usually considered as a simple process, as hardware supports this operation effectively. However, challenges in the subsequent phases totally depend on this packet collecting phase because any kind of contextual noises in this phase are hardly removed in the subsequent phases. Some variable factors or environments affect the variance of the packet data such as functionality, date, timing, and location. Note that all customized protocols utilize some of these data and their implementations can be somewhat different.

The functionality of a device is the most important factor to reverse engineer command related fields, and date and time are related to timestamps. The collecting phase has to run for sufficiently long time to clearly distinguish temporal information such as packet sequence number and time. The location is also highly related to the number of communication nodes. The number of nodes may complicate protocol reverse engineering. In other words, when the number of nodes is relatively small, the analysis become easier. By limiting the number of communication nodes, the entropy of address fields can be decreased. Therefore, if we can control these conditions, analysis becomes easier.

4.2.2 Grouping Phase

The second phase is a grouping process for the collected packets. Many types of packets exist in a typical digital communication protocol, such as request, response, command, and acknowledgment (ACK). The formats of these different types usually have different fields, and this difference causes the protocol analysis to be more complicated. By classifying the collected packets according to the source or destination addresses, we can obtain information about the request-response and command-ACK relationships. The length of the packet can provide useful information as well because different fields tend to have different packet lengths. For example, ACK packets are usually shorter than any other packets. Categorizing the collected packets according to the factors mentioned in the previous phase is also helpful.

4.2.3 Analysis Phase

The last step is the actual analysis of hex or ASCII-valued byte data in the grouped packets. The important data in the sequence of grouped packets are those that are repeated, periodic, monotonically increasing or decreasing, and otherwise meaningful. Repeated data can be related to the control features of the two previous phases including source or destination addresses, functional commands, and others. Periodic data can also be interpreted as having various meanings based on their timing characteristics such as time interval. Sequence numbers and timestamps of packets usually appear as monotonically increasing data, and they can be decoded according to their cycles. For example, the cycles of date or time data can be 12, 24, or 60 in decimal or hex, whereas those of a sequence number can be a full byte size. In particular, repeated data in every packet without any change are important because an attacker can use them as fixed values for a spoofing attack without knowing detailed information. Table 2 summarizes inferable information based on attributes of each byte previously mentioned.

Table 2: Inferable information according to attributes of each byte

Data Attributes	Deducible Information
Repeatability	Source/destination addresses
Periodicity	Time stamp (data or time), Functional commands
Monotonic increment or decrement	Sequence number
Constant Field	Use the values of corresponding byte positions as fixed for spoofing attacks

Because one byte of data can have several of these attributes, multiple analysis results are possible. With some criteria or algorithms used for scoring different analysis results, we can choose one or some results to generate possible spoofing packets. According to the analysis results, the uncertainty of each field can be reduced. Therefore, we can generate possible spoofing packets more efficiently.

5. IMPLEMENTATION

Based on the design in Section 4, we implemented an automatic wireless customized protocol reverse engineering tool named WASp. As its final output, this tool generates scored analysis reports and possible spoofing packets by automatically analyzing input files containing captured packets.

In this section, we present details of the implementation of each module and byte-level analysis in WASp. The entire program flow of WASp is shown in Figure 3. WASp consists of four modules: parsing and grouping, byte-level analysis, report construction, and packet generation, which are marked Ⓐ, Ⓑ, Ⓒ, and Ⓓ in Figure 3, respectively. Essentially, packet capture (pcap) files containing captured packets in specific conditions (mentioned in Section 4.2.1) and context files describing those conditions for each pcap file are used as inputs for our tool. Captured packets in pcap files are parsed and then divided into several groups (Ⓐ). After the grouping, the analysis module performs byte-level tests in each byte position for every group. We designed five byte-level tests: CRC, n-gram, entropy, feature, and range (Ⓑ). The results of each test are merged and scored as a report that includes basic protocol format information (Ⓒ). Finally, based on scored results in the reports, our tool automatically generates possible spoofing packets (Ⓓ). Because a report is generated for each group, a user can select a report to generate spoofing packets or refer all reports according to one's purpose.

5.1 Parsing and Grouping

Our tool requires two kinds of inputs: pcap files and context files for each pcap file. Pcap files contain raw data of captured packets, and context files contain information concerning capturing conditions in (keyword, value) tuples designated by a user or attacker. Because those conditions can be combined with a byte or field in

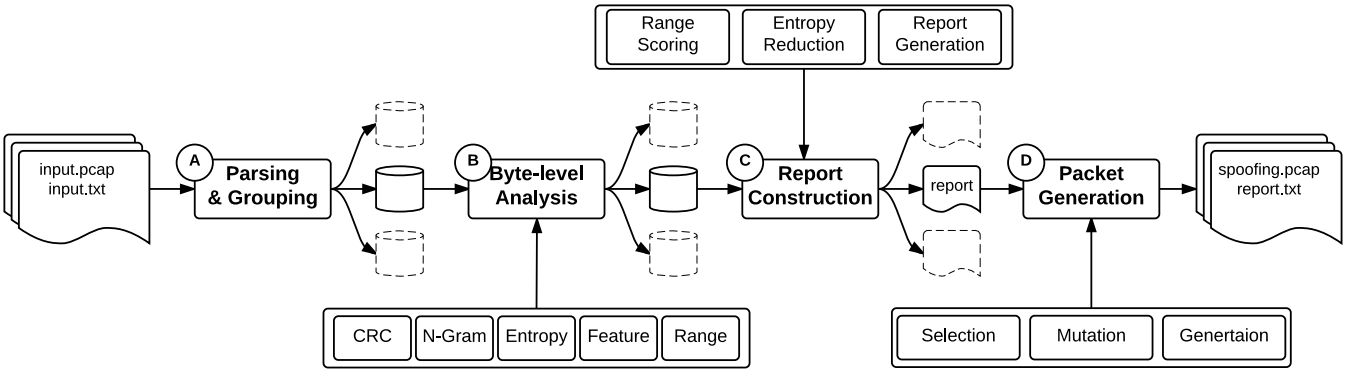

Figure 3: WASp: Automatic protocol reverse-engineering tool design

captured packets, context files can add information necessary for subsequent analyses.

For analyses, parsing and extracting MAC layer information from MAC headers of packets in pcap files are necessary. In this process, packets with wrong MAC layer CRCs are filtered out because they can be considered as broken packets in wireless transmission. After parsing, packets in pcap files are grouped into several groups according to packet length, and each group can be divided again into several subgroups according to (source address, destination address) tuples derived from MAC layer information. Because WPAN packets need to have minimal length to ensure low transmission power, customized protocols use a packet length that is as short as possible considering the purpose of each type of packet. In other words, the packet length is the easiest and most effective criterion to cluster packets according to type. By grouping and sub-grouping, we can reduce entropy related to address fields and even identify possible address byte positions and the values of given network nodes in a customized protocol.

Groups and subgroups classified in this module can also be used to produce a network diagram of a target system. Figure 4, which was automatically generated by WASp, shows an example of a network diagram for captured packets. Circles denote network nodes and their 2-byte hex values refer to MAC layer information in each node (i.e. addresses). Arrows refer to the direction of packet transmission, and next to each arrow, the byte length and number of corresponding packets are given. In addition, in Figure 4, packets contained in each arrow represent a subgroup and packets with the same byte length represent a group. This is helpful in understanding network structures or sender-receiver relationships of customized protocols, which have many communication nodes.

5.2 Byte-Level Analysis

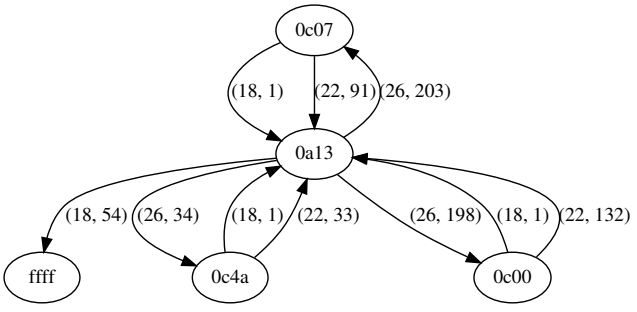

Figure 4: Example of network diagram for part of captured packets from PSD system as explained in Section 6

As previously mentioned, we essentially assumed that most data in the target protocols use a single-byte data format to reduce power consumption for each packet transmission. Because we do not know the target protocols at all, this is our best assumption at this point.

In the byte-level analysis module (Ⓑ in Figure 3), which is the main analysis process, we implemented five byte-level investigations. These investigations are based on characteristics of customized protocols explained in Section 3.2 for use in analyzing the formats of customized protocols.

CRC Test. CRC is one of the simplest methods for checking packet integrity. Thus, in customized protocols, well-known CRC algorithms are usually adopted. This test determines CRC existence using an n-gram method in customized payloads for each group or subgroup. For the CRC calculation, we adopted PyCRC module [18], which supports nine common CRC methods including CRC-16, CRC-32, and CRC-CCITT. Because we consider packets that have the same length as a group, this test detects the CRC method for each group and identifies the most frequent CRC method and its byte positions. We considered captured packets with unmatched CRCs not only in MAC layer but also in customized protocol layer as noise data, and we filtered them out for subsequent analysis.

N-gram Test. The purpose of the n-gram test is to locate reused MAC header data such as source addresses, destination addresses, and network IDs in customized layer data. We built databases for every possible contiguous n-byte sequence (n-gram data) from both MAC header and customized layer data. By comparing both n-gram databases with the same length for all packets in a group or subgroup, this test detects byte positions and lengths of reused MAC header data fully or partially.

In cases in which a group contains multiple subgroups, MAC header data of packets in the subgroups may be more consistent than those in the group. Therefore, results of an n-gram test for each subgroup are merged to conduct an n-gram test for the group.

Entropy Test. To measure the variance of each byte, we applied entropy (i.e., Shannon entropy), which means the average of information contained in each set of data in our analysis. In this test, we calculated each byte position's entropy in packets of a group or subgroup and the entropy of each context in context files related to that group or subgroup.

If the entropy value of a byte position is zero, it means that the byte position is fixed, and thus, we can simply use the fixed value as it is for spoofing packet generation. Non-zero entropy values are used in the subsequent steps.

Feature Test. For each pcap file, (context, condition) paired context information is manually written in a context file. The context infor-

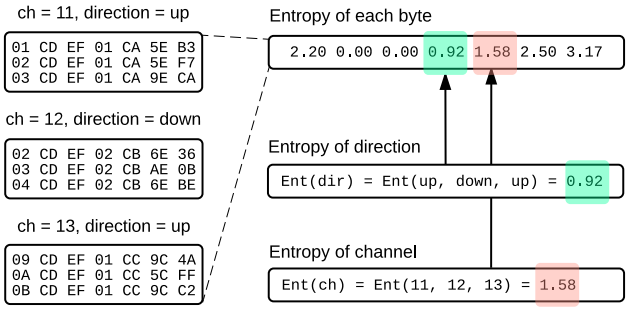

Figure 5: Example of feature test in byte-level analysis module

mation describes conditions such as communication channel numbers among 26 physical communication channels in IEEE 802.15.4, specific operations that can be observed by an attacker (e.g., turning on/off or moving directions), capturing locations (for multiple sender-receiver pairs) when packets are captured in a pcap file. In the feature test, entropies for each byte position and each context calculated in the previous test are compared to identify context-related byte positions that have the same entropy value as that of context entropy. If the entropy of the n-th byte position is equal to that of the channel information, then we consider the n-th byte position related to the channel.

An example is given in Figure 5. Three pcap files are recorded in different conditions, and each pcap file contains three packets with a byte length of seven shown in the left side of the figure. In this case, the entities of corresponding context files are (channel, 11) and (direction, up) for the first pcap file, (channel, 12) and (direction, down) for the second, and (channel, 13) and (direction, up) for the last. Entropies of each byte position for nine captured packets and context information (i.e., channel and direction) are shown in the right side of Figure 5. Because the values of entropies are the same as those of channel and direction (0.92 and 1.58 in the fifth and fourth byte positions, respectively), our feature test contemplates that the fifth and fourth byte positions are meaningfully related to channel and direction, regardless of their byte values.

Range Test. To identify another useful piece of information in a group of unknown byte sequences, we classify each byte position by the range of changes of byte values into raw decimal, raw hexadecimal, ASCII decimal, ASCII hexadecimal, sexagesimal, printable, upper-case and lower-case-letter ranges. For example, if the changes in value of one byte position are bounded between `0x30` (character "0") and `0x39` (character "9") in a group or subgroup, then this byte position is classified as both ASCII decimal and printable ranges. In general, sequence number bytes use only decimal or hexadecimal values (both raw and ASCII formats), time stamp bytes are bounded in sexagesimal range, and the value of command bytes is in letters range. In this test, byte positions identified by CRC and n-gram tests as well as zero-entropy byte positions are excluded. As a result, a bitmap that contains ones at the same bit positions as the corresponding byte positions as well as zeros at other bit positions is generated for each range.

Because one range can be either a subset of or overlapped with other ranges, this test can generate multiple results. Therefore, to generate possible spoofing packets efficiently, scoring the multiple results with reasonable criteria is necessary. This scoring process is included in the next module.

5.3 Report Construction

The purpose of the report construction module (Ⓒ in Figure 3) is to combine results of five tests in the byte-level analysis module to

Data: Results of range and entropy tests
Result: Score list for multiple results of range test
$L \leftarrow$ the byte length of packets
$B_k[n] \leftarrow$ k-th result (bitmap) of range test ($0 \leq n < L$)
$H[n] \leftarrow$ a result of entropy test ($0 \leq n < L$)
$scoreList \leftarrow$ an empty list
foreach B_k **do**
  $score \leftarrow 0$
  $entropy_{range} \leftarrow$ the entropy of corresponding range
  for $i \leftarrow 0$ **to** L **do**
    $score \leftarrow score + \dfrac{B_k[i] \times H[i]}{entropy_{range}}$
  end
  $scoreList$.append ($score$)
end
return $scoreList$

Algorithm 1: Scoring algorithm for results of range test

reduce entropy, and score overlapped results for some byte positions. In the last stage of this module, reports summarizing all results of automatic analyses are generated for every group and subgroup.

Range Scoring. As explained in the previous subsection, the range test derives multiple results. To determine the results to be used for generating possible spoofing packets, we developed a scoring algorithm that couples the results of range and entropy tests. This algorithm is briefly summarized in Algorithm 1.

In the algorithm, entropies of each byte position bounded in a specific range are reflected in the score of a range test result, with the exception of byte positions identified by CRC and n-gram tests, as well as zero-entropy byte positions. In addition, scores are normalized with the maximum entropy of the specific range. Calculated scores are used to generate possible spoofing packets in the packet generation module.

Entropy Reduction. First, we mark fixed-value byte positions through whole packets in a group or subgroup (i.e., zero entropy). Second, even though actual values of byte positions are not fixed, we can regard them as fixed or limited-valued byte positions based on results of CRC, n-gram, and entropy tests. Therefore we can reduce entropies for some dynamic-value byte positions.

Our rules to reduce entropy are the following. Note that all threshold values in the rules are heuristically chosen.

- In a group or subgroup, if the number of packets that use the same CRC algorithm at the same byte positions in the CRC test is over 90 % of total number of packets, those byte positions are fixed as CRC bytes.
- In a group or subgroup, if the number of packets that reuse the same MAC layer information at the same byte positions in the n-gram test is over 80 % of total number of packets, those byte positions are fixed as the reused values.
- If the range of a byte position is specified by the range test, the values of the byte position are bounded in the specific range.
- If the actual entropy of one byte position is less than half the full entropy of a byte, the values of those byte positions are limited by their existing values in a group and subgroup.

Report Generation. Before spoofing packet generation, every result of previous analyses for each group and subgroup are integrated

into a report. Because all tests except the entropy test can have multiple results, we merged results of those tests in order of priority based on the certainty and importance of each test. Byte positions identified in the CRC test are first locked, and then the results of n-gram and feature tests are applied in order. Finally, zero-entropy byte positions are fixed. The results of the range test are not merged exclusively. Instead, they are listed as scores in a report, and the results of the range test with the highest score is used for packet generation. In summary, for each byte position of customized protocol payloads in a group or subgroup, a single merged analysis result and the listed results of the range test are included in a report.

5.4 Packet Generation

Based on reports for each group and subgroup, the packet generation module (Ⓓ in Figure 3) automatically generates pcap files containing possible spoofing packets. Because reports show results based on byte position, we have to assign appropriate values to each byte position for spoofing attacks. Byte positions can be separated into two types: fixed and variable byte positions. From an attacker's point of view, he or she can simply duplicate fixed byte positions for spoofing attacks. By contrast, variable byte positions should be handled analytically based on analysis results rather than brute-forcing for efficient spoofing attacks. WASp generates spoofing packets based on the following three steps.

Selection. According to the number of groups and subgroups, dozens of reports are generated in the report construction module. WASp can generate possible spoofing packets for all reports, but it is considerably time-consuming and inefficient to verify the success or failure of spoofing attacks for every generated packet from all reports. However, because packets in subgroups are also contained in groups, generating possible spoofing packets for groups only is sufficient. In addition, a user can select specific reports for spoofing packet generation by considering contexts or spoofing conditions such as communication direction and target address.

Mutation. Our analysis approach is optimized to reduce entropy by finding meanings and bounding possible values of each byte or field. Naturally, the more we reduce entropy, the fewer the number of possible spoofing packets that are generated. This means excessive entropy reduction can reduce the chances of success in spoofing attacks. Therefore, we can trade entropy reduction for the chance of success. To increase the success rate, we can also mutate a report. For example, if adjacent byte positions with different entropies are in the same range, the maximum value among different entropies can be used for all of them, although one of them can be applied to entropy reduction rules mentioned in Section 5.3. This mutation can be enforced even for fixed byte positions.

Generation. After every process, all possible packets are generated according to the remaining entropies of each byte. These packets are used for spoofing attacks and in our evaluation.

6. EVALUATION

In this section, the performance of WASp is evaluated against two real-world targets: smart plug and PSD systems. Because of the unknown nature of customized protocols, we have no ground truth for our analysis results. Therefore, we use the amount of entropy reduction in our analyses and spoofing success rate in our experiments as evaluation metrics.

6.1 Target Systems

A smart plug system is an intelligent power metering system that basically consists of plug and controller units. The plug unit is controlled wirelessly by the controller unit and can turn the power of connected electrical devices on or off. In addition, it transmits the power usage information of the devices to the controller unit. For these functions, a customized protocol is used as wireless communication between the plug and controller units. We captured 217 packets in two contexts according to the functions of turning on/off and transmitting power usage information, to generate spoofing packets. The other target, a PSD system, prevents tripping and falling accidents at a platform, and it also reduces wind and dust caused by a train. Because the doors of both PSD and a train must be synchronized, PSD communicates with incoming trains wirelessly using a customized protocol. In our experiments, we captured 7,299 packets in 30 contexts which are combinations of five platforms, three trains, and both up and down directions.

We note that choosing target systems to evaluate WASp is limited because identifying whether a WPAN system uses a customized protocol before capturing packets is impossible.

6.2 Evaluation Metrics

For our evaluation, we used two evaluation metrics: the amount of entropy reduction and spoofing success rate. First, the amount of entropy reduction was used to demonstrate the efficiency of the automatic protocol analysis. Entropy is directly related to the number of generated spoofing packets, and too many spoofing packets cannot be transmitted in a reasonable time because of the limited speed of transceivers. For example, KillerBee, which in a SDR transceiver supporting IEEE 802.15.4, can ideally transmit as many as 500 packets per second, which means that nearly nine hours are required to transmit 3-byte brute-forced packets at least. Therefore, entropy must be reduced to a level sufficient to send in a limited time, especially for real-time spoofing attacks. In addition, the successful spoofing rate, which is the second metric, represents the accuracy of WASp. The high success rate proves that the formats and their values of spoofing packets generated by WASp (i.e., the report in Figure 3) are correctly constructed to approximate those of the actual customized protocol.

Therefore, both the amount of entropy reduction and the spoofing success rate must be considered to evaluate the performance of spoofing attacks. We note that although the main goal of this study is not to reveal the target protocol completely, WASp provides meaningful analysis results against the protocol. In addition, the results of WASp can be used to identify the most proper format of target protocols by analyzing successful and failed spoofing packets.

6.3 Evaluation Results

For our evaluation, we placed attacks on smart plug and PSD systems using automatically generated packets by means of WASp. Before actual analyses for generating spoofing packets, improper packets that contain wrong CRCs in MAC and custom layers were filtered out as noise packets through the process described in Section 5.1 and Section 5.2. The change in the number of packets by filtering is shown in Table 3. The numbers of packets that are used for actual analyses are given in the last row of Table 3.

6.3.1 Smart Plug System

For our evaluation, we performed spoofing experiments against a smart plug system because it is a commercial product and under our control. Therefore, for this target system, both metrics were measured by experiments.

Grouping. From the analysis results of packets collected from the smart plug system, we identified three packet formats (i.e., three groups) whose byte lengths were 23, 24, and 35, respectively, and the number of corresponding packets were 22, 135, and 43, respectively.

Table 3: Results of noise packet filtering with CRC in MAC layer and custom layer for both target systems (No custom layer CRC is implemented for the smart plug system.)

	# of Packets (Smart Plug)	# of Packets (PSD)
Originally Captured Packets without Filtering	217	7,299
Remaining Packets after Filtering Wrong MAC Layer CRC	200	6,986
Remaining Packets after Filtering Wrong Custom Layer CRC	200	6,963

These groups had no subgroups. This means that packets in each group are transmitted in only one direction.

Results. The plug unit is targeted for spoofing attack experiments because it is an actuator controlled by the controller unit. Accordingly, possible spoofing packets are generated based on the report of the group with 135 packets and a 24 byte length. From spoofing experiments, we determined that spoofing was successful if any response packet was detected for each spoofing packet.

By WASp (without mutation as described in Section 5.4), a series of possible spoofing packets (Type 1) that has maximally reduced entropy were generated. To enlarge the coverage of spoofing packets, Type 1 was mutated as Type 2 and 3. The last entropy reduction rule in Section 5.3 was not applied for Type 2, and the entropy was increased for Type 3 as the mutation example in Section 5.4. In other words, Type 1 is a subset of Type 2 and 3 for exclusively different byte positions. For each type, entropy reduction rates, the numbers of generated packets, and time duration to transmit all generated packets with a 0.2 seconds delay are shown in Table 4.

Type 1, which is the main type, showed 95.50 % reduced entropy compared to the original entropy. When each packet is transmitted every 0.2 seconds, 1.3 minutes are required to transmit all generated packets theoretically. By mutating Type 1, which increases entropy, the greater the number of spoofing packets that are generated and the more time is required proportionally. To test all packets for Type 3, over two hours are required.

In spoofing experiments, sending packets continuously for approximately two hours is nearly impossible, especially for a real-time spoofing attack. To avoid this practical limitation and for a fair comparison of the three types, we transmitted 100 spoofing packets that were randomly chosen for each type. In addition, we tested for different speeds to transmit packets from 0.002 to 0.9 seconds per packet. All experiments were repeated 100 times. The spoofing success rates were averaged for 100 times of identical experiments as shown in Figure 6.

The results show that the spoofing success rate increased from 0.002 to 0.1 seconds of the time interval according to the increased time interval per packet transmission and then stabilized after 0.2 seconds of the time interval. After stabilization, spoofing success rates for Type 1, 2, and 3 averaged 48.19 %, 5.52 %, and 47.09 %, respectively. Note that spoofing success rate of Type 1 and Type 3 are impressive results given the circumstances. Spoofing is performed without any internal system information such as current sequence number. Moreover, input datasets only contain limited number of packets during limited time span, making it impossible to guess the exact packet format.

Table 4: Automatically generated spoofing packet types against a smart plug system (success rates are averaged over 10 identical experiments.)

	Entropy Reduction Rate	# of packets	Time to transmit all packets (0.2 seconds delay)
Type 1	95.50 %	400	1.3 minutes
Type 2	93.57 %	5,200	17.3 minutes
Type 3	92.04 %	40,000	133.3 minutes

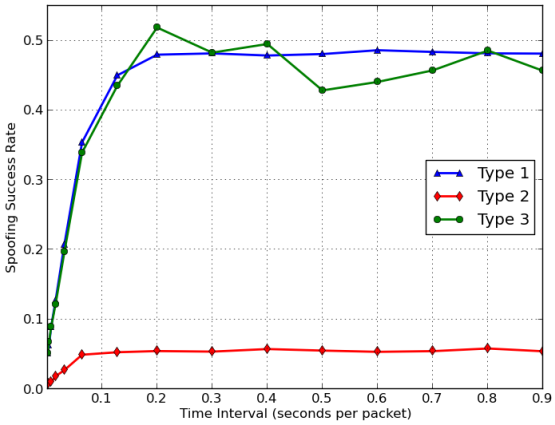

Figure 6: Spoofing success rate according to spoofing time interval from 0.002 to 0.9 seconds per packet (for 100 packets randomly chosen in possible spoofing packets generated by WASp)

When the time interval is too short, success rate is quite small, maybe because the device cannot handle packets in such a short time interval. Therefore, the graph shows the sharp increments of the spoofing success rate as the interval increases from 0 to 0.1 or 0.2 seconds. This means that for the spoofing test using spoofing packets generated by WASp, we cannot transmit spoofing packets and receive responses too quickly. Moreover, the total time for the spoofing test can be limited in some conditions, especially when a target system is not in our control. Hence, those packets should be generated efficiently. In other words, we have to lower the entropy of customized protocols during the analysis stage.

In the case of Type 2, the success rate significantly dropped compared to that of Type 1. Because the ratio of success rate drop between Type 1 and 2 (48.19 % and 5.52 %, respectively) is inversely proportional to the ratio of their packet numbers (400 and 5,200 in Table 4, respectively), this drop means that most of those additionally generated packets failed to spoof. Based on this comparison, we can remove useless values for a particular byte, which is also helpful in further protocol analysis. While it utilizes two orders of magnitude more spoofing packets, average success rates for Type 3 is lower than Type 1. This is due to the sampling effect. For each type, we choose 100 samples randomly as discussed above.

6.3.2 PSD System

In contrast to our evaluation of the smart plug system, we could use only the amount of entropy reduction as an evaluation metric in that of the PSD system. Because of legal and safety concerns, we could not perform spoofing attacks and were unable to obtain a spoofing success rate in this evaluation. However, the entropy reduction results suggest that our analysis is effective and that a spoofing attack in real-world situation is possible. Moreover, we strongly believe that packets generated by WASp can spoof a PSD system in a real-world situation because the entropy of PSD's customized protocol is reduced sufficiently.

Grouping. We clustered 7,299 packets into 15 groups according to packet length before noise packet filtering. After filtering, we

excluded 336 packets, which represent only 4.6 % of the originally captured packets. However, among 15 groups, nine groups and two subgroups were filtered out. The removed groups consisted of a few packets and the number of these packets was not enough to automatically analyze. Therefore, those removed packets could be considered as noise for analysis. In addition, this means that the number of reports to which an attacker should refer was effectively reduced by filtering.

Results. Table 5 represents the results of entropy reduction by means of byte-level analysis (Ⓑ in Figure 3) and report analysis (Ⓒ in Figure 3). However, three of them contained less than 0.2 % of the total packets, and two groups only contained a single packet. After filtering, three of the six groups contained more than 99.8 % of captured packets. Their entropy reduction results are listed in Table 5.

Using WASp, the entropies of byte positions are decreased by anchoring those byte positions as CRC, reused MAC data, and context information, and then applying our entropy reduction rules (Section 5.3) to collected packets of the customized protocol. The amount of entropy reduction for each group is summarized in the last row of Table 5. The average entropy reduction rate is 88.11 % for three dominant groups.

7. DISCUSSION

Ground Truth. Customized protocols are proprietary protocols that have no accessible documentation related to protocol specification. We cannot know the exact protocol format. In other words, no ground truth exists to check whether packets are generated in the correct format. In addition, the target systems are a black box when an attacker has generally no privilege against program source code or firmware. Therefore, we evaluated our tool by performing spoofing attacks against the target system and measuring the attack success rate. As we mentioned in Section 1, the initial purpose of this research is to improve the security of wireless customized systems by automating spoofing attacks. Therefore, we infer the correctness of our analysis results through the success rate of spoofing attacks. In other words, if the spoofing attack with generated packets is effective, then we can explicitly evaluate the accuracy of our automatic analysis without the original packet format.

Selection Bias. Our tool requires sufficiently many packets to derive exact analytical results for a high success rate of spoofing attacks. For example, if we assume that our tool collects packets for only a single day and one field in the collected packets indicates the date, this date field will be considered as a fixed field. Therefore, packets that are generated based on this analysis result can counter a spoofing attack on another day if the target checks the packet generation date. Not only as in this case, but also entropy, range, and feature tests may be incorrect because of the selection bias of collected packets, and this will adversely affect the quality of reports and packets. Therefore, users must consider possible field cases and collect packets that can cover all cases with sufficient randomness to remove selection bias.

Table 5: Results of entropy reduction for the customized protocol used in a PSD system

	Group 1	Group 2	Group 3
Number of Packets	418	2,615	3,916
Custom Layer Packet Size	7 bytes	11 bytes	15 bytes
Enrtopy Reduction Rate	96.26 %	90.04 %	78.01 %

Incorrect Implementation of IEEE 802.15.4 CRC. Some vendors have implemented IEEE 802.15.4 CRC inaccurately on their IoT devices. For example, we collected IEEE 802.15.4 packets from a wireless parking space detection (WPSD) system in a hypermarket, and we found that they had wrong CRC values. Systems that possessed an extremely simple function and short wireless packet structure such as a WPSD system may work with CRC error. However, we could not analyze the protocol because we cannot guarantee that there is no error in the packets, as such errors will impair accurate analysis. Thus, this kind of system is out of scope for this study, and our tool filters packets that have a wrong MAC layer CRC or application layer CRC.

IEEE 802.15.4 Family Protocols. IEEE 802.15.4 defines only the PHY and MAC layers, and IEEE 802.15.4 family protocols such as ZigBee, Z-Wave, and WirelessHART have their own network layer on the MAC layer of IEEE 802.15.4. Although these protocols are based on IEEE 802.15.4, they are not within the scope of our tool. We assume that reusing the source and destination addresses and poor integrity checks are the characteristics of customized protocols. However, the family protocols do not have these characteristics. In particular, they support strong integrity checks such as keyed-hash message authentication code (HMAC), so generating spoofing packets is impossible.

8. CONCLUSION

The main purpose of this study is to analyze and reconstruct unknown wireless customized protocols over IEEE 802.15.4 automatically. For this purpose, we classify the characteristics of these protocols and develop a novel methodology to analyze and reconstruct these protocols based on those characteristics. In addition, applying the methodology, we implement an automated wireless customized protocol spoofer called WASp for analysis and spoofing packet generation. Compared to manual analysis, WASp is proved to be much faster and to generate efficient analysis reports with five kinds of byte-level investigation. In particular, results of byte-level statistical analyses including entropy and range tests revealed that WASp identifies meaningful byte positions, bounds their values, and narrows the coverage down for spoofing packet generation efficiently. For evaluation, possible spoofing packets are generated based on the analyses by WASp, and we apply those packets to real-world commercial applications such as smart plug and PSD systems. Results reveal that, on average, our tool could reduce approximately 90 % of entropies for both target systems, and approximately 48 % of generated packets could spoof one of them.

Although several previous works have studied automatic reverse engineering of network protocols for intrusion prevention and detection, they focused on application-level protocols as their targets. Thus, they are not fully applicable to our target protocols, which possess little context information because of low-power requirements. Consequently, to the best of our knowledge, this study is the first to implement an automated protocol reverse-engineering tool specialized for unknown customized protocols over IEEE 802.15.4. We believe that WASp is a useful tool to understand customized protocols with unknown structures that do not include textual data. In addition, WASp can be used to secure those protocols and prevent potential spoofing attacks for IoT networks that use WPAN.

9. ACKNOWLEDGMENT

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (No. NRF-2014M3C4A7030648)

10. REFERENCES

[1] J. Antunes, N. Neves, and P. Verissimo. Reverse Engineering of Protocols from Network Traces. In *Proceedings of the 18th Working Conference on Reverse Engineering (WCRE)*, Limerick, Ireland, Oct. 2011.

[2] I. Bermudez, A. Tongaonkar, M. Iliofotou, M. Mellia, and M. M. Munafo. Automatic Protocol Field Inference for Deeper Protocol Understanding. In *Proceedings of the 14th IFIP Networking Conference (NETWORKING)*, Toulouse, France, May 2015.

[3] B. Bloessl, C. Leitner, F. Dressler, and C. Sommer. A GNU Radio-based IEEE 802.15. 4 Testbed. *12. GI/ITG FACHGESPRÄCH SENSORNETZE*, 2013.

[4] G. Bossert, F. Guihéry, and G. Hiet. Towards Automated Protocol Reverse Engineering Using Semantic Information. In *Proceedings of the 9th ACM Symposium on InformAtion, Computer and Communications Security (ASIACCS)*, Kyoto, Japan, June 2014.

[5] J. Caballero, P. Poosankam, C. Kreibich, and D. Song. Dispatcher: Enabling Active Botnet Infiltration using Automatic Protocol Reverse-Engineering. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*, Chicago, Illinois, Nov. 2009.

[6] J. Caballero, H. Yin, Z. Liang, and D. Song. Polyglot : Automatic Extraction of Protocol Message Format using Dynamic Binary Analysis. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS)*, Alexandria, VA, Oct.–Nov. 2007.

[7] P. M. Comparetti, G. Wondracek, C. Kruegel, and E. Kirda. Prospex: Protocol Specification Extraction. In *Proceedings of the 30th IEEE Symposium on Security and Privacy (Oakland)*, Oakland, CA, May 2009.

[8] W. Cui, J. Kannan, and H. J. Wang. Discoverer: Automatic Protocol Reverse Engineering from Network Traces. In *Proceedings of the 16th Usenix Security Symposium (Security)*, Boston, MA, Aug. 2007.

[9] W. Cui, V. Paxson, N. C. Weaver, and R. H. Katz. Protocol-Independent Adaptive Replay of Application Dialog. In *Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2006.

[10] W. Cui, M. Peinado, K. Chen, H. J. Wang, and L. Irun-Briz. Tupni: Automatic Reverse Engineering of Input Formats. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS)*, Alexandria, VA, Oct.–Nov. 2008.

[11] Ettus Research. USRP N2x0 Series Device Manual. http://files.ettus.com/manual/page_usrp2.html. [Online; accessed 12-March-2016].

[12] Gerald Combs and others. Wireshark. https://www.wireshark.org/. [Online; accessed 12-March-2016].

[13] S. Gorbunov and A. Rosenbloom. AutoFuzz: Automated Network Protocol Fuzzing Framework. *IJCSNS*, 10(8):239, 2010.

[14] Q. Huang, P. P. C. Lee, and Z. Zhang. Exploiting Intra-Packet Dependency for Fine-Grained Protocol Format Inference. In *Proceedings of the 14th IFIP Networking Conference (NETWORKING)*, Toulouse, France, May 2015.

[15] IEEE Computer Society. IEEE Standard for Local and metropolitan area networks - Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs), 2011.

[16] T. Kitagawa, M. Hanaoka, and K. Kono. AspFuzz: A State-aware Protocol Fuzzer based on Application-layer Protocols. In *Proceedings of the 15th IEEE symposium on Computers and Communications (ISCC)*, Riccione, Italy, June 2010.

[17] Z. Lin, X. Jiang, D. Xu, and X. Zhang. Automatic Protocol Format Reverse Engineering through Context-Aware Monitored Execution. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2008.

[18] C. Năvălici. PyCRC 1.21-Python CRC Calculations Modules. https://pypi.python.org/pypi/PyCRC, 2015. [Online; accessed 12-March-2016].

[19] C. Rossow and C. J. Dietrich. ProVex: Detecting Botnets with Encrypted Command and Control Channels. In *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2013.

[20] A. Tongaonkar, R. Keralapura, and A. Nucci. SANTaClass: A Self Adaptive Network Traffic Classification System. In *Proceedings of the 12th IFIP Networking Conference (NETWORKING)*, Brooklyn, NY, May 2013.

[21] P. Tsankov, M. T. Dashti, and D. Basin. SecFuzz: Fuzz-testing Security Protocols. In *Proceedings of the 7th International Workshop on Automation of Software Test (AST)*, Zurich, Switzerland, June 2012.

[22] Y. Wang, X. Yun, M. Z. Shafiq, L. Wang, A. X. Liu, Z. Zhang, D. Yao, Y. Zhang, and L. Guo. A Semantics Aware Approach to Automated Reverse Engineering Unknown Protocols. In *Proceedings of the 20th IEEE International Conference on Network Protocols (ICNP)*, Austin, TX, Oct.–Nov. 2012.

[23] Z. Wang, X. Jiang, W. Cui, X. Wang, and M. Grace. ReFormat: Automatic Reverse Engineering of Encrypted Messages. In *Proceedings of the 14th European Symposium on Research in Computer Security*, Saint Malo, France, Sept. 2009.

[24] J. Wright. KillerBee. https://code.google.com/p/killerbee/. [Online; accessed 12-March-2016].

[25] Z. Zhang, Z. Zhang, P. P. C. Lee, Y. Liu, and G. Xie. ProWord: An Unsupervised Approach to Protocol FeatureWord Extraction. In *Proceedings of the 33rd IEEE International Conference on Computer Communications (INFOCOM)*, Toronto, Canada, Apr. 2014.

[26] J. Zhao, S. Chen, S. Liang, B. Cui, and X. Song. RFSM: A Smart Fuzzing Algorithm Based on Regression FSM. In *Proceedings of the 8th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, Compiegne, France, Oct. 2013.