Towards Automatic Protocol Field Inference

Ignacio Bermudez[a], Alok Tongaonkar[a], Marios Iliofotou[b], Marco Mellia[c], Maurizio M. Munafò[c]

[a]*Symantec Corporation, 350 Ellis St., Mountain View, CA, 94086, USA*
[b]*Caspida, 2100 Geng Road#100, Palo Alto, CA, 94303, USA*
[c]*Politecnico di Torino, Corso Duca degli Abbruzzi 24, 10129, Turin, Italy*

Abstract

Security tools have evolved dramatically in the recent years to combat the increasingly complex nature of attacks. However, these tools need to be configured by experts that understand network protocols thoroughly to be effective. In this paper, we present a system called *FieldHunter*, which automatically extracts fields and infers their types. This information is invaluable for security experts to keep pace with the increasing rate of development of new network applications and their underlying protocols. FieldHunter relies on collecting application messages from multiple sessions. Then, it performs field extraction and inference of their types by taking into consideration statistical correlations between different messages or other associations with meta-data such as message length, client or server IP addresses. We evaluated FieldHunter on real network traffic collected in ISP networks from three different continents. FieldHunter was able to extract security relevant fields and infer their types for well documented network protocols (such as DNS and MSNP) as well as protocols for which the specifications are not publicly available (such as SopCast). Further, we developed a payload-based anomaly detection system for industrial control systems using FieldHunter. The proposed system is able to identify industrial devices behaving oddly, without any previous knowledge of the protocols being used.

1. Introduction

In recent years attacks against networks have become more complicated. To defend against these complex attacks, network security systems have also evolved to use more sophisticated mechanisms. For instance, firewalls have moved from using simple packet-filtering rules to using application level rules that need deeper understanding of the protocols being used by network applications. Similarly, intrusion detection systems are increasingly using vulnerability based signatures [1] that contain information specific to network protocols. Access control mechanisms are also evolving from IP address based policies to fine-grained policies which use protocol objects such as users and message types.

It is clear that configuring all of the above applications requires a deeper understanding of the network protocols, which is done through reading protocol specifications. However, comprehending protocol specifications is a very tedious task. Moreover, many of the proprietary protocols specifications are not publicly available. The traditional approach of manual reverse engineering a protocol cannot cope with the rate at which new benign or malicious applications are made available and brought into workplace. As a result, security administrators have to configure security applications with very limited visibility into the network protocol space; thus adversely affecting the efficacy of these tools in securing the network.

The above technology challenge has led to a growing interest in the research community in the development of techniques for automating the reverse-engineering process for extracting protocol specifications, which consists of inferring message formats and underlying protocol state machines. The state-of-the-art techniques can be classified in two categories: reverse-engineering through binary code analysis [2, 3, 4, 5, 6] and from network traffic [7, 8, 9, 10, 11, 12, 13]. In this work, we present an automatic reverse-engineering system of the second category, i.e. it infers protocol specifications from just network traffic data. Reverse-engineering using network traffic has an advantage over techniques using binary analysis, because application binaries are not always available to the security operators.

Our approach to the problem of protocol reverse engineering aims to extract field boundaries and field protocol types from network traces that belong to the protocol. As compared to previous works in this area, we are able to extract richer protocol information in terms of (i) extracting diverse field types, and (ii) handling binary and textual protocols in an uniform framework. We study well known protocols and identify a set of field types that can be used in a multitude of security applications. We focus on identifying: (i) Message Type (MSG-Type), such as flags in DNS protocol or GET/POST keywords

Email addresses: ignacio_bermudezcorr@symantec.com (Ignacio Bermudez), alok_tongaonkar@symantec.com (Alok Tongaonkar), marios@caspida.com (Marios Iliofotou), marco.mellia@polito.it (Marco Mellia), maurizio.munafo@polito.it (Maurizio M. Munafò)

in HTTP, (ii) Message Length (MSG-Len), usually found in TCP protocols to delimit application messages in a stream, (iii) Host Identifier (Host-ID) such as Client ID and Server ID, (iv) Session Identifier (Session-ID) such as cookies, (v) Transaction Identifier (Trans-ID) such as sequence/acknowledgment numbers, and (vi) Accumulators such as generic counters and timestamps. We note that a protocol may not have all the above types of fields.

We built a system called FieldHunter, that uses a two step methodology: (i) Field extraction: here we extract fields from the protocol messages. (ii) Field type inference: here we infer the type of the fields extracted in the previous step. The key contribution of our work is the development of various heuristics based on observed statistical properties for inferring the different field types. In our evaluation, we used real network traces from three different Internet Service Providers (ISPs) to validate the ability to extract various field types from well known protocols such as Real Time Protocol (RTP), as well as protocols without any publicly available specification such as SopCast's protocol.

Next, to illustrate the use of FieldHunter in building end-to-end security applications, we developed a payload-based anomaly intrusion detection system for industrial control systems. Industrial Control Systems (ICS) encompass several types of control systems used in industrial production, including Supervisory Control and Data Acquisition (SCADA) systems, Distributed Control Systems (DCS), and other smaller control system configurations such as Programmable Logic Controllers (PLC). Due to their use in the industrial sectors and critical infrastructures, they are a prime target for attackers and the cost of successful attacks is tremendous to the victims. Typically, the attackers targeting ICS are sophisticated, and have a lot of resources; sometimes even being state sponsored. They are able to avoid detection by traditional defense mechanisms [14]. To make matters worse, there has been a trend of increasing number of attacks on critical infrastructure as evidenced by the rapid increase in the number of reported attacks on ICS from 91K in 2012 to over 675k in 2014 [15]. To the best of our knowledge, our system is the first payload-based anomaly detection system that handles legacy proprietary protocols commonly used in ICS networks.

The rest of the paper is organized as follows. §2 defines the terminology used throughout the paper, §3 provides details about the core algorithms used by FieldHunter. Performance evaluation and parameter tuning are presented in §4. We describe the anomaly detection system for ICS in §5. We discuss about assumptions and limitations in §6, related works in §7 and finally conclude this work in §8.

2. Terminology

Figure 1 shows a pictorial representation of the terminology used throughout this work. Our system uses as input a set of **conversations** [1] of a particular application. We refer to such a set as **collection**. Conversations consist of exchanged **messages** between two hosts. Messages from client to server are denoted as C2S (dark-colored) and from server to client as S2C (light-colored). We consider the initiator of the conversation as the client, the other end as the server, and identify hosts by their IP address. Messages consist of different pieces of information enclosed in **fields**. As we show in figure 1, conversations evolve horizontally over time (t) and messages can be compared vertically across multiple conversations.

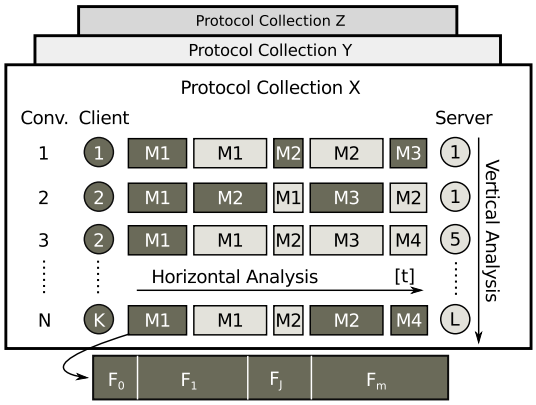

Figure 1: Terminology diagram.

To enable the analysis of a collection, the messages in the conversations can be grouped together in the following ways: (i) Grouping messages based on their position in conversations, e.g., all third messages in C2S direction. (ii) Grouping together all the messages of a conversation. This essentially captures session-like information. (iii) Grouping together messages by direction, e.g., all C2S messages. We note that (i) and (ii) are very similar to vertical and horizontal sub-collections as defined by Kreibich et al. [16]. Message grouping is instrumental for FieldHunter to find patterns in the collections. If these groups do not contain enough message diversity, FieldHunter cannot unveil the field types it is designed for.

It is worth mentioning that the formation of protocol collections used by FieldHunter is beyond the scope of this work. However, we suggest two alternatives for the same. One way is to use a test-bed in which the application is executed while the traffic exchanged is being captured. Alternatively, the collection can be extracted from passive observation of actual traffic by the means of network classifiers, i.e., by filtering all conversations involving a well-known port (see §5), or by relying on a behavioral traffic classifier classifier [17].

Application conversations are transported by TCP/UDP segments and are extracted by FieldHunter using the following methodology: (i) for messages transported over UDP it is assumed that each segment

[1] A conversation is formed of the two flows in opposite directions, where a flow is defined by the 5-tuple (Layer-4 Protocol, Source IP, Source Port, Destination IP, Destination IP)

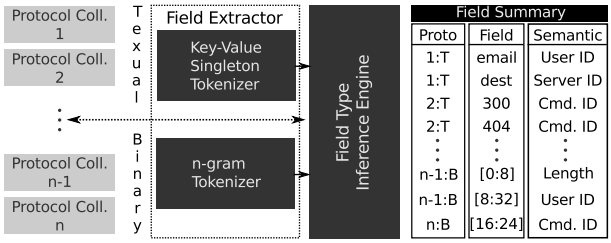

Figure 2: FieldHunter system diagram.

contains one application message, and (ii) for TCP it is assumed that TCP PUSH flags delimits the beginning of a new application message from the end of another one. An accurate message extraction can be done once the MSG-Len field has been identified by FieldHunter.

We make a distinction between textual and binary protocols as follows: Textual protocols use human readable words and symbols to structure data, and they look more like a text document. Exponents of textual protocols are HTTP and SMTP. On the other hand, binary protocols encode data on bits rather than symbols and the way that data is structured is quite rigid. Examples of binary protocols are DNS and DNP.

3. Design

In this section, we describe the system design and discuss the two components of FieldHunter (i) Field Extractor (ii) Field Type Inference Engine. These components are run in sequence to obtain a field summary report (which describes the identified fields and their types) as shown in figure 2.

3.1. Field Extractor

Textual and binary protocols differ greatly in the way fields are delimited. Textual protocols typically use *delimiters* such as "`:`" or "`\r\n`" (carriage-return and line-feed pair) to separate fields. On the other hand in binary protocols, fields either have fixed offset and size or offsets and lengths that are specified in some preceding fields. Hence we have developed different techniques for textual and binary protocols. Next, we explain each of these techniques separately.

3.1.1. Textual Protocols

Field extraction for textual protocols boils down to identifying field delimiters. However, this is a non-trivial task as many protocols use multiple delimiters for different purposes. For instance, consider a message such as `TIME-OUT: 60 # PORT: 54001`. In this message, "#" is used to separate out the fields, while ":" is used to separate out the key and the value in a field. Hence, we categorize delimiters into two types: (i) **Field delimiter (D_f)**: separates the different fields of a message, e.g., the "#" character in the above example. (ii) **Key-value delimiter (D_{k-v})**: separates the key from its corresponding value, e.g., the ":" in the same example.

Table 1: Common text-based protocols and their observed delimiters. GAME: Team Fortress (game), TEL: Telnet, CS: Counter Strike (game), GNU: Gnutella.

Prot.	D_f	D_{k-v}	**Prot.**	D_f	D_{k-v}
HTTP	\r\n	':', ' '	FTP	\r\n	' '
SMTP	\r\n	':', ' ', '-'	TFTP	0x1D	0x1E
POP3	\r\n	':', ' '	CS	0x5C	0x5C
RTSP	\r\n	':'	GNU	\r\n	':', ' '
SIP	\r\n	':', ' '	RTP	\r\n	':', ' '
GAME	0x00	0x00	MSN	\r\n	' '
TEL	\r\n	':', ' '			

Commonly used D_f and D_{k-v} delimiters are shown in Table 1. These delimiters are obtained from the documentation of the listed protocols, and are actually observed in our data sets. As we see, there are popular delimiters, such as `\r\n`, as well as non-standard delimiters, such as `0x00` (null), `0x1D`, and `0x5C`.

Generally speaking, FieldHunter identifies delimiters using three key observations: (i) Delimiters are non-alphanumeric sequences of 1 or 2 characters. (ii) Delimiters have a high horizontal and vertical frequency compared to other non-alphanumeric sequences in a textual protocol. (iii) There is only one D_f that splits up the messages into **key-value pairs** (`UID: 1234`, `Content-length: 872`) or **singleton keywords** fields (`HELO`, `LOGOUT`, `OK`, `FAILED`). FieldHunter first identifies D_f and then proceeds with the D_{k-v} if fields are key-value paired.

Field Delimiter Inference. FieldHunter finds frequent sequences of non-alphanumeric characters in the protocol which are considered to be delimiter candidates d. Then from among all the candidates it chooses only one ($D_f = d$), such that it splits up any protocol message into valid key-value pairs and singletons. Validity of key-value pairs and singletons is checked by comparing common prefixes and exact matches respectively.

Key-Value Pair Delimiter Inference. Once D_f has been detected, messages are split into fields from which we need to identify key-values along with D_{k-v}, and singletons. The identification of D_{k-v} is performed in three steps: (i) FieldHunter clusters fields of the same type by using the Longest Common Prefix (LCP); (ii) by re-clustering the clusters, FieldHunter cleans up possible outliers caused by two or more keywords sharing a common prefix. E.g., `Port:` and `Point:` have `Po` in common, and finally (iii) we choose the D_{k-v} as the non-alphanumeric suffix part of the LCP of each group. In the case that all the LCPs are identical for a group, then we say that the field contained by the group is a singleton and we do not search for a delimiter.

3.1.2. Binary Protocols

In binary protocols, fields represent serialization of variables as they are structured in memory. To parse these fields, message recipients need to know the structure of the data, i.e. the offset and length of the fields. The challenge for FieldHunter is that the message data structure is initially unknown. Therefore messages are split into n-grams which are used by Field Type Inference Engine. We observe that for most of the field types, the n-grams forming the field also show similar characteristics to the field. For instance, in a protocol that has a 32-bit Host ID field, the four 8-bit n-grams also exhibit similar statistical properties as Host ID. In such cases, we identify the field type for the single n-gram and then check whether consecutive n-grams can be merged into a larger field of the same type.

We note that this assumption does not always hold. For instance, a 32-bit Accumulator field may increment by one every time. But given the number of samples that we may consider in our collection (say order of thousands), the most significant bits may show up as constants and not accumulators. This issue is circumvented for fields such as Message Length and Accumulators (numerical representations) by considering n-grams of larger size first, say 32-bit n-gram, and then iteratively reducing n-gram size till the whole n-gram fits the field. Moreover, we handle byte-endianness for fields that contain numerical representations by repeating the heuristics separately for both little-endianness and big-endianness. This is not the case for fields that can be interpreted as categorical representations.

3.2. Field Type Inference Engine

Our approach is based on the following key observation: Fields with different types change differently over specific sub-collections. For instance, a field that consistently takes a distinct value for each IP address may represent a Host-ID. Similarly, fields that increment by one over sequential messages of a conversation may be part of a message counter.

FieldHunter assigns types to fields by using different statistical tests that are further explained. The techniques for FTI are similar for both textual and binary protocols. In the rest of the paper we use the term "n-gram" to interchangeably to mean "binary n-gram" or "textual field" for ease of exposition. For example, when we state "*n-gram entropy is computed*", we actually mean that either "binary n-gram entropy is computed" or "textual field entropy is computed". We use specific statistical tests based on different associations between observed variables to infer different field types. The association between two variables (a, b) can be of the following types: (i) "numerical correlation" ($a \Leftrightarrow b$), e.g., message length field is numerically correlated to the observed length of the message, (ii) "categorical correlation" ($a \in A \Leftrightarrow b \in B$), e.g., user IDs correlate categorically with IP addresses and (iii) "causality correlation" ($a \Rightarrow b$), e.g., certain type of message will result in a particular response from server.

The labeling process works by making a hypothesis that a given field is of a certain type. When the hypothesis holds, i.e., the field exhibits the statistical behavior of the field type, FieldHunter labels the field as such. We note here that a field may be labeled as multiple field types. For instance, an acknowledgment number field could be labeled as both Transaction ID as well as an Accumulator.

In figure 3 the more complex heuristics are illustrated using block-diagrams. Blocks in the diagrams represent different tests; horizontal/vertical arrow inside a block defines horizontal/vertical sub-collection analysis and thresholds are highlighted in italic. More details on parameter selection are given in §4.4.

3.2.1. **Message Type (MSG-Type)**

MSG-Type contains information about the underlying protocol state machine and its values represent the semantic of the whole message. Thus, the content of MSG-Type field is used by the receiver to understand what type of message is received, for example a request, a status update, or an error message.

The process of finding MSG-Types is based on two key observations: (i) MSG-Type takes values from a well defined small static set; and (ii) represents transitions in an underlying protocol state machine. Hence, by pairing request/response messages, there is a high probability that their corresponding MSG-Type fields are related. The leftmost diagram in figure 3 describes the MSG-Type labeling process.

Using observation (i) above, FieldHunter first looks for n-grams that vertically are neither random nor constant. Randomness of a n-gram x can be measured using the **entropy** $H(x)$ metric. Let p_i be the probability of having the n-gram take the value i. Then $H(x) = -\sum_i p_i \log_2 p_i$, where $0 \cdot \log(0) = 0$. By definition for 1-byte n-grams (8-bits) $H(x)$ takes values between 0 (constant) and 8 (perfectly random). Then n-grams that are unlikely to be part of a MSG-Type field are discarded. Once some candidate fields are identified, according to observation (ii), we check for n-grams that have a causal relationship with n-grams in the response messages. Here FieldHunter uses categorical correlation metric. Towards this end, FieldHunter measures causality using the information theoretic metric $I(q;r)/H(q)$, where $I(q;r) = H(q,r) - H(q|r) - H(r|q)$ is the mutual information, that measures the information shared by a request (Q) and a response (R) [18].

FieldHunter takes n-grams for which causality is greater than a threshold, say 0.8, as MSG-Type candidates. For the case of binary protocols, if multiple n-grams are candidate, these are grouped together and causality is checked again. Thus, if a group coincides with the actual MSG-Type field, then the whole candidate group should also satisfy the initial hypothesis of causality. For example, suppose n-grams at byte offset 1, 5, 6 show a large causality such that $q1 \Rightarrow r1$, $q5 \Rightarrow r5$, $q6 \Rightarrow r6$. Then we check whether the groups $(q1, q5, q6) \Rightarrow (r1, r5, r6)$ holds

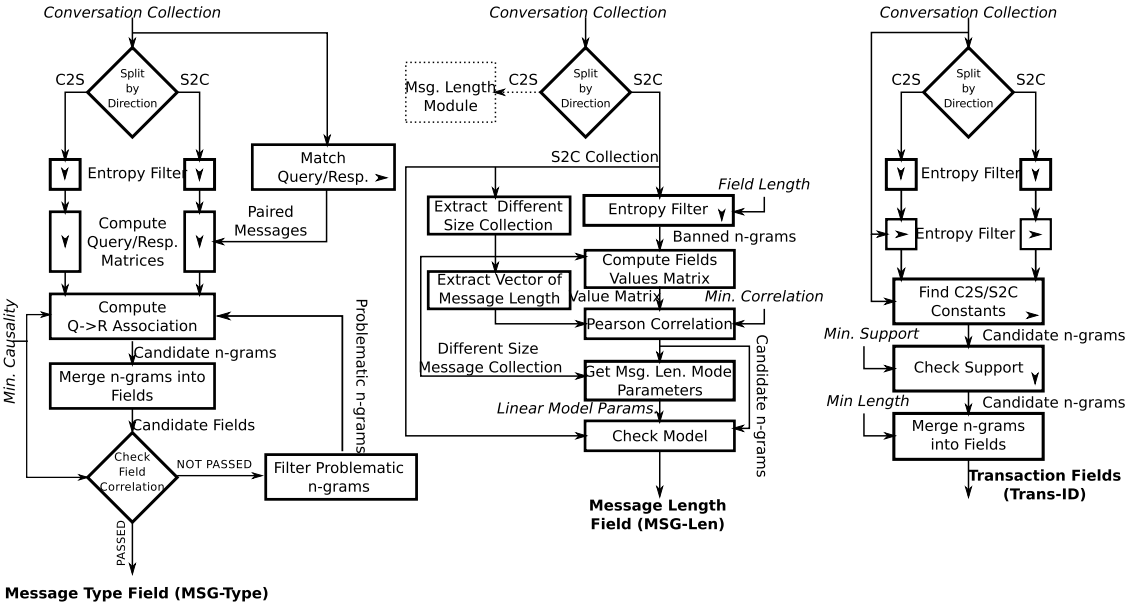

Figure 3: MSG-Type (left), MSG-Len (center) and Trans-ID (right) modules.

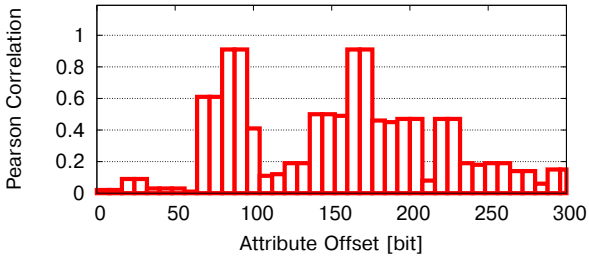

Figure 4: n-gram correlation with MSG-Len for SopCast.

the causality. If this holds, the field containing n-grams at offsets $(1, 5, 6)$ are returned as the MSG-Type field.

3.2.2. Message Length (MSG-Len)

Our goal here is to find fields that contain indication of the application message length. As such, we expect the MSG-Len field is linearly correlated with the actual physical message size. We use two different tests for identifying linear correlations in order to have higher confidence on our results.

The complete MSG-Len test algorithm is depicted in the central diagram in figure 3. This heuristic does not use the typical 1-byte n-gram and for textual protocols it decodes the content of the field as a number. The reason why 1-byte n-grams do not provide good results is that Most Significant Byte and Least Significant Byte are not correlated in this case. Hence, FieldHunter iteratively selects n-gram windows of size 32, 24, 16-bits that are shifted at a step of 8-bits. Such windows sizes are the standard sizes used to represent integers in computer memory. At each iteration Pearson correlation coefficient tells whether the numeric values of the fields are associated with the length of the messages. Notice that the computation of this correlation could be affected by biases due to some popular messages in the collection of the same size, for instance, if 950 out of 1,000 of collection messages are 40 bytes long. We avoid such biases by stratifying messages by length, creating in this way a size-heterogeneous collection not affected by the bias problem. We select all the fields for which the coefficient is above a certain threshold as MSG-Len candidates. We empirically found 0.6 to be a good threshold.

Figure 4 shows the results of applying the Pearson correlation to the SopCast protocol collection obtained from one of our traces. In this example, we use 16-bit n-gram. Pearson coefficient values span from zero to one, where zero indicates no correlation and one represents a strong correlation. In figure 4 there are two clear spikes, one at offset 88-bit and the other at 168-bit that suggests the presence of a MSG-Len field (see § 4.2). We cross-verified these results using DPI signature rules for UDP SopCast found in OpenDPI [19], an open source packet inspection engine [2].

Once the candidates are found, the next step is to conduct a test to verify that the candidates indeed are carrying information regarding the length of the message. The hypothesis is that the message length expresses the length of the message in an unit of measurement, such as bytes or words, and that it describes the length of data starting from a given byte offset. In other words, we state that the message length is ruled by the following linear equation: $MSG_{len} = a \cdot FIELD_{value} + b$, such that $MSG_{len} \in \mathbb{N}$ is the observable message length, $FIELD_{value}$ is the value taken by candidate field, $a > 0$ accounts for the unit of measurement and $b \in \mathbb{N}$ is the starting offset of the data described by the field. To verify the assumption, the linear equation is solved and (a, b) are obtained. This process is repeated taking all possible message pairs with different lengths. Finally a candidate is considered as a true MSG-

[2] OpenDPI project has since been discontinued.

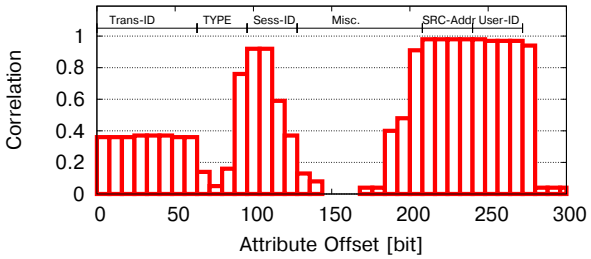

Figure 5: n-gram correlation with Client IP (Vuze DHT).

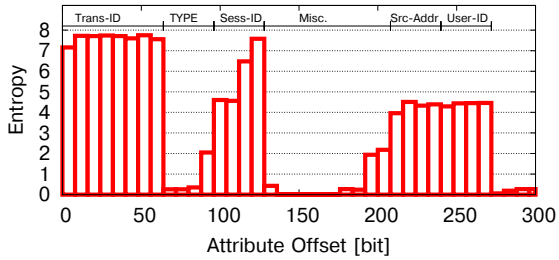

Figure 6: The n-gram entropy for Vuze DHT over a C2S vertical sub-collection.

Len field if for most of the pairs ($> 90\%$) the solution is acceptable ($a > 0 \wedge b \in \mathbb{N}$).

3.2.3. Host Identifier (Host-ID)

Host identifiers are used to identify a particular host or device beyond the boundaries of the local network. For instance, in peer-to-peer applications, the "Peer-ID" field uniquely identifies a specific peer/host in the whole overlay, even when the peer is behind a Network Address Translation (NAT) device or is moving over multiple networks.

The heuristic assumes that all the messages sent by the same host carry the same Host-ID, i.e., for a given source IP, messages are likely to have the same Host-ID. Then Host-ID should be strongly correlated with the IP address of the sender. Based on this assumption, FieldHunter computes the categorical correlation $R(x, y) = I(x; y)/H(x, y) \in [0, 1]$ of n-grams x with the sender IP address y, where $H(x, y)$ is the joint entropy (that measures the total amount of information that x and y jointly carry). That is, for each $x \in X$, there is a different $y \in Y$, and vice-versa. N-grams with correlation coefficient greater than certain threshold, say 0.9, are selected as candidates. Finally, consecutive candidate n-grams are merged into fields of at least a *length of 4 bytes*. Notice that the adoption of statistical tests, such as correlation, makes the algorithms robust to handle noise in the data, such as when NAT is used.

Figure 5 shows the categorical correlation between n-grams in a vertical collection and the corresponding source IP address for the Vuze DHT collection [20]. Note how $R(x, y)$ is very close to one (high correlation) for n-grams that represent the Client Address and the Client-ID. However, we also observe that the first n-grams of the Session-ID are also correlated with the sender IP address. The explanation for this protocol peculiarity is found in the Vuze's specification. Vuze's Session-ID is an application's global counter randomly initialized at the start-up and incremented by 1 for each new conversation. Hence, the most significant bits in the Session-ID are likely to be the same for all messages sent by the same sender. By imposing a *minimum length* constraint, FieldHunter can discard such fields.

3.2.4. Session Identifier (Session-ID)

Session Identifier keeps track of application-level sessions that span over multiple conversations. Semantically, it is similar to the use of `Cookies` in HTTP. Since the Session-ID remains constant between a pair of endpoints, FieldHunter correlates the n-grams to the pair of client and server IP addresses. Then we proceed using the same categorical correlation as we do for Host-ID.

3.2.5. Transaction Identifier (Trans-ID)

The algorithm we use to detect Trans-IDs is illustrated in the rightmost diagram in figure 3. It is assumed that Trans-ID are randomly picked by the transaction creator and then copied back in the replies. Therefore, we first search for n-grams that appear random across both vertical and horizontal collections. Randomness is measured using entropy as before.

Figure 6 shows the entropy of n-grams for the Vuze DHT protocol. The figure shows the entropy of the first 36 n-grams (reported on the x-axis at the corresponding offset) in the C2S vertical sub-collection. On the top, the protocol field names are reported as extracted from documentation. In this example, n-grams with high entropy are good candidates for the Trans-ID field.

Next, all consecutive request/response messages are paired and for each of them, it is checked whether the n-grams/fields take the same values. If the check passes, then the pair of n-grams are added to a set of Trans-ID candidates. Note that request/response message formats can change and Trans-ID may appear at different offsets (for instance in Vuze DHT). Therefore, the heuristic does not assume the protocol message formats are the same in both directions.

Finally, FieldHunter measures the consistency of these candidates over all the conversations, i.e., n-gram candidates with enough support, say > 0.8, are finally marked as such. Minimum support allows some degree of mismatch, for example, caused by message reordering or retransmission in the collection. Finally, consecutive n-grams are merged to form a field of at least 2 bytes. For textual protocols such n-gram merging is not needed.

3.2.6. Accumulators

Accumulators are fields that have increasing values over consecutive message within the same conversation. These

fields typically represent message sequence numbers, acknowledgment numbers or timestamps. To identify such fields, we calculate the difference, denoted as Δ, between values of n-grams in two subsequent messages. We expect Δ to be positive and fairly constant. Notice that differences are not required to be perfectly constant. For instance a byte-wise counter in a protocol of variable size messages would have variable Δ.

We search for accumulators in C2S and S2C directions independently of each other. As with the MSG-Len field, here we start with fixed size n-grams. We assume accumulators are encoded in fields of a given *field length*, for example, 64, 32 and 16 bits. For each field offset, we compute the vector of increments (Δ) considering each consecutive message pair in each conversation. In order to use one threshold that captures the variations among Δs of different scales (such as sequential counters vs millisecond timers), we compress Δ using a logarithmic function; $\hat{\Delta} = \ln \Delta$. Next, we analyze $\hat{\Delta}$ and select those that have relatively low entropy, i.e., $\hat{\Delta}$ looks "fairly constant".

3.3. Field Summary

FieldHunter provides information of the field type extracted automatically out of protocols as the final result. It provides two separate reports (corresponding to each direction of messages) for each protocol. The report contains the set of fields for which the types have been inferred. Note that we may not identify the type for some of the fields and will skip them in the report.

4. Evaluation of FieldHunter

In this section we evaluate the efficacy of FieldHunter in inferring protocol specification of known protocols. We use TSTAT [17], a DPI tool that classifies traffic and feeds FieldHunter with protocol collections. In general, each collection presents different characteristics. For instance, some may contain wrongly classified flows caused by DPI false positives. Other may present little diversity, for example, showing only conversation exchanged with a handful of servers. Different traces generate different collections that are separately analyzed (for cross verification purpose). We consider a protocol collection as valid only if it has at least 200 conversations for textual or 2,000 for binary protocols; see § 4.4 for more details.

The subset of protocols for which we present results are summarized in Tables 3 and 4 for binary and textual protocols, respectively. Both straightforward and challenging cases are considered in our evaluation.

4.1. Datasets

We evaluate FieldHunter using three different ISP traces (Table 2). Data was collected from different geographical regions (Asia, Europe, and South America), between the years 2007 to 2012. All traces contain full payload from network connections. Given the large size of the TR1-2012 trace we limit the payload per connection to the first 1048 bytes[3]. Although, all our parameter selection is made using TR1-2012, we tested FieldHunter on all three traces.

Table 2: Summary of the traces we use.

Name	Location	Network Location	Date	Duration
TR1-2012	Europe	Edge	04-2012	24 h
TR2-2009	S. America	Backbone	10-2009	4 h
TR3-2007	Asia	Backbone	01-2007	7 h

Table 3: Summary of the results from running FieldHunter on the binary-based protocols.

Protocol	Discovered/GT [bits]		Cov/AoC	
	C2S	S2C	C2S	S2C
Vuze DHT	288/240	200/208	0.87/1	0.85/0.87
DNS	48/32	56/32	0.75/1	1/1
uTP	88/96	200/96	0.75/1	0.67/0.87
RTP	80/88	80/88	0.82/1	0.82/1
ED2K	128/16	16/16	1/1	1/1
KADEMLIA	352/16	104/16	1/1	1/1
STUN	256/160	184/160	0.9/0.83	0.85/0.88
SOPCAST	128/?	152/?	?/?	?/?
PPLIVE	0/?	32/?	?/?	?/?

4.2. Evaluation of Binary Protocols

Table 3 reports the number of Discovered and Ground-Truth (GT) bits, for both C2S and S2C collections, the Coverage (Cov) and the Accuracy over Coverage (AoC). Cov is the ratio between Discovered bits of the GT and the GT bits; while AoC is the ratio between correctly discovered bits over the total number of discovered bits.

The first seven protocols in the table have known specifications, and the latter two do not have. Note that for some protocols, the number of discovered bits is larger than the GT bits. This happens because many protocols carry other protocols on top of them, such as ED2K. FieldHunter does not differentiate the header from the protocol's payload, resulting in identifying the fields of the transported (inner) protocol as well.

The average Accuracy over Coverage (AoC) is 0.83 in the worst case. We observe that typical inaccuracies are due to the Accumulator type. For counters that span over large fields (for instance, a 32-bit long number), FieldHunter easily identifies the less significant bits, but tends to miss the most significant ones, which are identified as "constant". From the results shown in the table, we discuss the details for three interesting case studies.

4.2.1. ED2K and KADEMLIA

ED2K and KADEMLIA eMule messages are preceded by a common header which is used as GT. FieldHunter

[3]We did not observe this to cause any notable problems. Only for some protocols with long payloads, such as HTTP, portions of the payload and rarely portions of the application-layer header were not fully captured.

correctly identifies such common header. Moreover, it discovers additional fields, that sum up to a total of 128 bits in the C2S EDK2 collection. After manual inspection, we observed those fields to correctly include Session-ID, and Host-ID.

4.2.2. SopCast

SopCast is a proprietary and closed protocol used for P2P-TV broadcasting. Unveiling information about the message format of such protocols is one of the motivations for developing FieldHunter.

Specifically, this protocol represents a large fraction in the TR3-2007 trace. FieldHunter identifies 128 bits corresponding to: MSG-Len, Trans-ID, Session-ID, Host-ID (we hypothesize it is used for NAT traversal since it uses 64 bits, 32 of which correspond typically to private IP addresses, and 32 are identical to the Host public IP address) and some accumulators of 16, 32 and 64 bits (possibly used to reorder video/audio chunks).

4.2.3. Domain Name Service (DNS)

For DNS in the TR1-2012 trace, FieldHunter successfully identifies the Trans-ID and a MSG-Type field, each of 16 bits. We expected parsing DNS in this trace to be challenging due to the bias in the collection. First, most of the C2S messages are "DNS Requests" messages. Second, requests are directed to the most popular DNS resolvers (in the TR1-2012 trace customers use the ISP DNS server). Despite this, FieldHunter is able to identify some protocol fields.

Interestingly, in the C2S messages, FieldHunter reports the presence of a 16 bit accumulator in the DNS Trans-ID field. We manually verified this, and discovered that some implementations of DNS clients generate a "random" Trans-ID by using a local counter. FieldHunter captured this peculiar but common behavior, exposing more details about the protocol.

4.3. Evaluation of Textual Protocol

Table 4 reports overall results for textual protocols. It reports the number of inferred fields, the number of key-value pairs (K-V) and singletons (most of them MSG-Type) for each direction (with the exception of the last two protocols for which the DPI provided just one direction of the conversation). In addition, we report those fields that we label as being identifiers (IDs), highlighting those that proved to be False Positives (FP). Here, by IDs we mean Host-IDs, Session-IDs, and Trans-IDs. Overall, from the 26 fields labeled as IDs, 22 are verified as correct and only three are false positives. In general, we observe that the majority of the fields of textual protocols are successfully inferred in both the C2S and S2C directions. Similar to the binary protocols, we pick two interesting textual protocols as case studies.

Table 4: Summary of the results from running FieldHunter on the textual protocols

Protocol	#Fields C2S/S2C	K-V C2S/S2C	CMD C2S/S2C	IDs C2S/S2C	FP-IDs C2S/S2C
STUN	3/3	2/2	1/1	1/1	0/0
FTP	19/18	12/17	7/1	2/1	0/0
HTTP	9/14	9/14	0/0	3/0	1/0
POP3	9/28	5/24	4/4	2/0	0/0
SMTP	19/9	15/9	5/0	1/1	0/0
MSNP	3/4	3/4	0/0	2/0	0/0
RTSP	9/25	9/18	0/7	3/6	0/2
GAME RSP	*/17 3/*	*/15 2/*	*/2 1/*	*/2 1/*	*/0 0/*

4.3.1. Microsoft Notification Protocol (MSNP)

The MSN protocol is present in all three traces. FieldHunter correctly finds that the field called `USR` field carries a Host-ID and which indeed is the MSN's user name. Similarly, the `CVR` field which is used to send specific information about the client and its OS to the server. This field is captured by FieldHunter since system settings are different for each MSN user, but consistent during the communication with the server. Although `CVR` is not an actual Host-ID, this is a right interpretation for the field type because the field behaves the same as Session-ID.

4.3.2. Real-Time Streaming Protocol (RTSP)

The S2C direction of this protocol returns 6 inferred ID fields. Out of these, four are correctly labeled and two are false positives. The latter occur when some fields that are supposed to take different values actually always take the same value for a given conversation, behaving similar to a Session-ID. The false positive fields are `Last-Modified` and `Cache-Control`. For instance, `Last-Modified` is the timestamp of the last modification for a given content. Since a single object is requested using multiple RTSP conversations, its modification time appears constant across conversations. Similarly, the `Cache-Control` field tends to always take the same value among conversations used to retrieve the same content as well. In general, we observe that the original collection may be biased toward some specific subset of protocol fields and values. This is challenging for FieldHunter, and, in general, any field inference algorithm that relies on traffic data.

4.4. Sensitivity Analysis & Parameter Tuning

We evaluate the sensitivity of FieldHunter to different parameters and to external factors, such as the collection size. As mentioned before, we perform parameter tuning using one trace, and then we evaluate FieldHunter on all three traces. Next we show how the design proved to be robust to parameter tuning.

First we focus on one of the most challenging fields to infer, the MSG-Type for binary protocols. We consider all collections for those protocols that have available

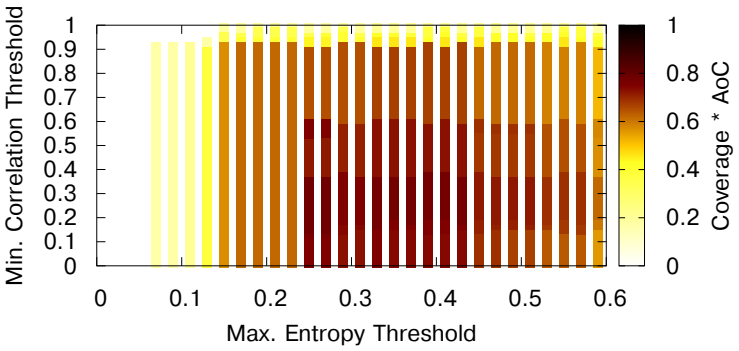

Figure 7: Parameter sensitivity for the MSG-Type.

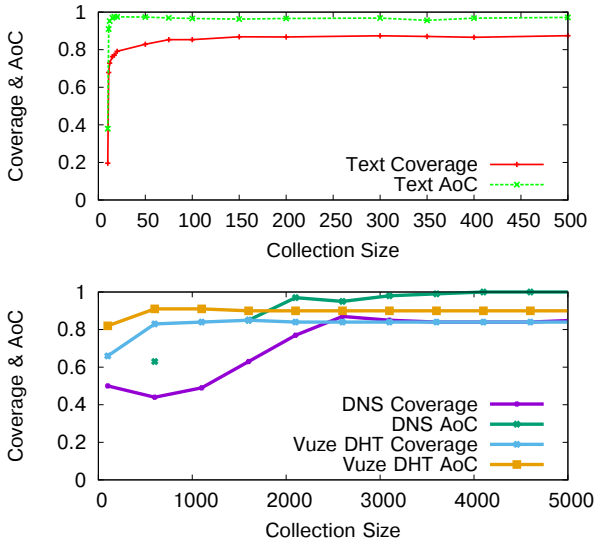

Figure 8: Coverage and AoC versus the number of conversations. text-based protocol (top); DNS and Vuze DHT binary-based protocols (bottom).

ground truth. Then for each collection, the MSG-Type algorithm is executed manifold by tweaking the thresholds (*Min. Correlation* and *Max. Entropy*). For each threshold pair, the product between Coverage and AoC is computed, providing a coefficient from 0 to 1, where values close to 1 are desired. The results are reported in figure 7. The darker the block, the better FieldHunter performs. As can be clearly seen, there is a large range of good parameters that yield scores above 0.8, which means that in most cases FieldHunter was able to correctly pinpoint the MSG-Type field. We repeat the experiment for other field types and we observed qualitatively similar results.

We now evaluate the effect of the collection size for both binary and textual protocols. For textual protocols, we first select nine protocols for which we know all fields present in our traces. Then, we randomly extract a subset of conversations from the collections and run FieldHunter over the subset. Results are compared against ground truth to compute the Coverage and AoC (figure 8). We see that FieldHunter performs well even with limited number of textual conversations. In fact, when 50 conversations are considered, we identify 85% of all the fields, with 97% AoC. Overall, using large enough collections, we always identify the D_f delimiter for all tested protocols. Most of the mis-labeling happens due to challenges in inferring the D_{k-v} for some fields.

For binary protocols, we perform similar evaluation, but focus on two challenging protocol cases, DNS and Vuze DHT. The results are shown in the bottom plot of figure 8. As we can see FieldHunter may require a bigger collection sizes to produce the best results. We believe that the heuristics apply differently on textual and binary protocols as textual protocols are less sensitive to diversity. Vuze DHT represents the protocol for which we had highest diversity in our dataset, with many end-points exchanging a variety of messages. Conversely, DNS (from TR1-2012) represented a challenging scenario due to little diversity in the collections: typically only one MSG-Type (DNS requests) was found, and conversations were very short (a single request/response). As we see, eventually we achieved very good results for DNS, but it required as many as 2,000 conversations.

5. Application

In this section, we present an end-to-end security application that uses FieldHunter. Specifically, we focus on intrusion detection systems (IDS) which are a common defense mechanism for many critical infrastructures. IDS can be categorized into anomaly detection systems or signature based systems based on the detection mechanisms used. Anomaly detection systems work by learning the *normal*, also called *baseline*, behavior of any system and flagging any deviation from this as *anomaly*, which can be considered an indicator of malicious behavior. This is in contrast with signature based systems which use signatures of known threats. The advantage of anomaly detection systems over signature based ones is that they can detect previously unknown attacks, i.e. zero-day attacks. However, anomaly detection systems can suffer from higher false positives.

In this work, we develop a payload-based anomaly detection system for Industrial Control System (ICS) networks. Most of the current solutions rely on statistical features such as volume of traffic for creating baseline [21]. Such systems are ineffective against stealthy attacks that work by modifying the protocol behavior without causing discernible change in statistical properties of the traffic. There are a few systems that offer payload based anomaly detection, but they rely on the protocol specifications which are not available for many of the legacy protocols that are used quite often [22].

FieldHunter forms the core of the anomaly detection system. It infers protocol fields from the ICS network being monitored. These protocol field summaries are con-

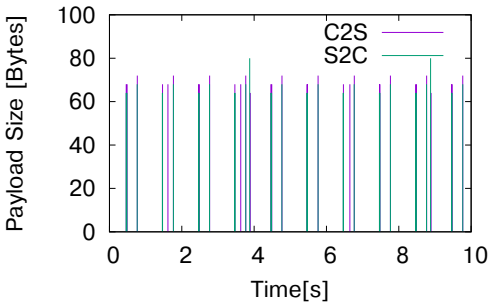

Figure 9: Temporal patterns shown in ICS communications

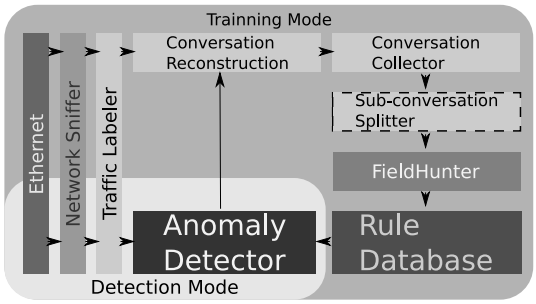

Figure 10: Payload Based Anomaly Detection System diagram.

verted to policy rules which form the baseline of the system. Any deviation in network protocol payload from these rules are flagged as anomalies. This allows us to detect attacks which modify the protocol messages or introduce new types of messages which are not part of the protocol. There is a direct mapping between the human readable field summaries and the policy rules generated. This is greatly beneficial to any analyst analyzing the root cause of alerts generated by the anomaly detection system.

5.1. ICS Networks

ICS networks typically consists of a set of devices such as sensors or controllers and a central monitoring/administrative unit. These networks differ from other networks such as enterprise or ISP networks in many respects. ICS network traffic contains a large portion of proprietary protocols which are unknown or not well documented along with a few well known protocols such DNS or SNMP. These proprietary protocols are predominantly binary based. The communication patterns in these networks are quite deterministic as only a handful of devices communicate with each other according to configurations that are set statically. Moreover, many of the connections are long-lived, often lasting for hours or days.

These characteristics present some unique challenges for FieldHunter. First challenge is the lack of diversity in the traffic, which renders some of the heuristics such as detection of Host-IDs ineffective. However, this is not a big limitation as our goal is to detect anomalous behavior and not protocol understanding. Hence, even if we incorrectly label a Host-ID field as a Constant, this field can be used to detect violations when the attacker intentionally changes the "Constant" value.

Second challenge is the presence of long-lived network flows. Many of the thresholds for various heuristics are set assuming that the input to FieldHunter contains multiple flows. We analyzed long-lived ICS connections and observed that often these connections show repeating patterns which indicate some conversations happen with a specific frequency. Hence, multiple conversations occur serially within a single flow. Therefore, we can break up a single flow into multiple conversation by computing the frequency of communication. Figure 9 depicts packet payload sizes in a window slice of 10 seconds for a very long conversation. We observe that the devices involved in the communication send burst of protocol messages at a frequency of approximately 1 second.

5.2. Design

The payload-based anomaly detection system (Figure 10) has two modes of operation: Training and Detection. In Training mode, the system generates network protocol profiles associated with specific TCP/UDP ports used by the physical devices for communication. During this phase, FieldHunter plays a critical role, since it learns the protocol field summaries used as baselines by the next step. In Detection mode, the anomaly detector uses rules to determine flow by flow if any baseline policy rule has been violated. For instance, the detection mode can verify whether a flow contains the particular value that a specific field is supposed to take as per the policy rule. Otherwise, the system raises an alarm of violation.

5.2.1. Training Mode

Network traffic is intercepted using a tap, that listens to network communications of devices connected to the ICS network. TCP/UDP payloads are extracted and grouped properly, forming the protocol collections required by FieldHunter to create functional protocol descriptions.

Traffic Labeler, accomplishes the fundamental job of creating the protocol collections. Since this IDS is profiling network protocols running in a particular mode on industrial devices, traffic is labeled by their associated triplets (Layer-4 protocol, destination port, destination IP address). For instance, packets going through a bi-directional TCP network connection, in which IP 192.168.1.101 is sending packets to 192.168.1.202, using source port 9988 and destination port 1822, has two triplets associated with it: (TCP, 9988, 192.168.1.101) and (TCP, 1822, 192.168.1.202).

The Conversation Reconstruction module reassembles conversations. The Conversation Collector groups them by label (triplet), until we get enough number of packets associated to a triplet i.e., greater than a pre-defined threshold. The dataset associated to a label is the protocol collection for FieldHunter (See §2) and the threshold tells how much data is enough to start field inference from the collection.

The Sub-conversation Splitter module is used in to handle very long-lived connections. This module splits the long conversations into multiple small conversations when strong temporal patterns in the traffic are observed (see figure 9). To do this, the module artificially increments the number of connections associated with a collection such that there is one for each direction of the conversation.

FieldHunter uses these conversations to generate protocol summaries. The protocol summaries are converted to policy rules in a straight-forward manner. The type of the field determines the nature of the rule. We have a predetermined way of converting each field type to a rule. The key idea behind this conversion is that each rule specifies how a field should behave, for instance, what values it can take or how the value correlates to something else such as message length. As an example, the rule for MSG-Type field specifies which opcodes to expect in the flows. These rules are put into the Rule Database.

The duration of the training mode depends on in the amount of available data rather than the clock time. This means if an application handles more traffic, the required clock time to generate summaries is lesser. Figure 8, gives a rough idea of the required amount of application messages to be collected before moving into the detection mode.

5.2.2. Detection Mode

In a nutshell, detection mode can be described by the interaction of three different modules: the Rule Database, the Traffic Labeler and the Anomaly Detector. Our system has two inputs in detection mode: on the one had it receives network traffic and on the other it gets the rules from the database. The output of the anomaly detection system is packets/connections which are flagged as anomalous as they violate the rules learned by the system.

The traffic labeler simply labels traffic with the respective pair of triplets and forwards it to the anomaly detector. Using the triplets, the detector can retrieve the rules from the database and apply them to the traffic. If rules are violated, system raises alarms indicating that the traffic contains anomalous packets or connections. When rules can not be found in the rules database, traffic is forwarded to the training mode path.

5.3. Evaluation of Anomaly Detection System

We tested our system using traffic collected from PowerCyber the test-bed. It simulates and emulates components of a smart grid including industrial SCADA [23]. The anomaly detection system is trained with clean network traffic. Finally, we evaluate its efficacy by running traffic collected during an attack.

5.3.1. Datasets

Network traffic is collected from one control center network of PowerCyber's test-bed. Traffic is collected during normal operation of the control center network and also during an attack. Such attack is performed intentionally in a controlled environment.

Table 5: Network captures used for evaluation of the payload-based anomaly detection system

Trace	**Duration**	**#Flows**	**#Messages**	**#Malicious Flows**	**#Endpoints**
TPC	2[h]	176	46,503	0	5
MPC	2[h]	180	31,274	2	5

Table 5 describe the datasets used for testing the detection system. Two datasets of two hour duration, each with about 180 flows, are used. Despite the few flows present in these datasets, the number of application messages is about thousands, which allows FieldHunter to provide meaningful results from it. The attack dataset contains both normal traffic and two flows belonging to an authentication attack. We use full packet traces and there is no packet filtering. Training Packet Capture (TPC) contains only normal traffic, while Mixed Packet Capture (MPC) contains normal and attack traffic mixed together.

5.3.2. Evaluation

We evaluated the system end to end, using TPC dataset to train the system and MPC for testing. The expectation was that the anomaly detection system should flag only malicious flows in the mixed dataset and ignore the benign traffic.

We found 6 different protocol summaries from MPC. One of them belongs to the DNP3 protocol [24], which is common in SCADA systems. The message structure of DNP3 protocol is defined by a header containing the constant value 0x0564 at the start, the message length, a control byte, destination and source ids, and a Cyclic Redundancy Check (CRC) field followed by the payload. Similarly payload is structured as data blocks which contain user data (1-16 byte) followed by a CRC.

From the protocol, FieldHunter is able to identify parts of the constant field, message length, and destination and source ids. In the given trace, the control byte appears as constant (0xC4 for client-to-server direction and 0x44 for server-to-client direction). Similarly, the destination field appear as a constant in the client-to-server messages as there is only one server in the trace, and the same for the source in the other direction. Therefore, the identity of clients can be associated to IP address.

MPC, contains a malicious DNP3 flow which abuses the lack of authentication in the protocol. Another machine in the same network spoofs valid DNP3 messages, but it has to provide a 2 byte entity in the source id field of the protocol. The attacker has two options: (i) provide a new source id, or (ii) use one already used by another client. In both cases we can raise an alarm, because in case (i) we observe a new source id not seen during training; for case (ii) we observe that the source id does not match the IP address associated to that identifier. In this particular attack the attacker spoofed the identifier of one of the

observed clients. The system detected the anomaly and can provide a detailed report to the administrator to make an informed decision.

We also evaluated our system on real industrial datasets. Results were similar to the ones presented for test-bed dataset. Unfortunately, due to privacy concerns about the data, we can not disclose further details in this paper.

6. Assumptions and Limitations

In this section, we discuss the assumptions of FieldHunter. Further, for each assumption we describe the limitations imposed on the system due to that assumption.

Training data contains patterns corresponding to the protocol fields:

FieldHunter assumes that the training data contains the patterns that allow it to identify protocol fields. Hence, the system cannot identify fields when traffic is compressed or encrypted. However, this limitation can be overcome by using a pre-processing stage that decompresses or decrypts traffic before passing it to FieldHunter. Another consequence of this assumption is that the FieldHunter is dependent on the quality of the dataset. Therefore, if the traces do not contain the required diversity that highlights data patterns, FieldHunter is not able to identify fields from traffic, or worst it may misclassify fields. For example, a Server-ID may be classified as a constant if traces only contain traffic directed to one particular server.

TCP PSH flag delimits the start of a new application message:

For TCP communication FieldHunter does not assume that a single TCP segment contained in packet carries an application message. It heavily relies on TCP PSH flag to delimit the end from the beginning of a new application message in the TCP stream. The assumption here is that the TCP PSH flags are set when sender does not have more data to transfer. On the receiver side ,TCP PSH flag is used to trigger transmission of the data to the application layer. When a TCP PSH flag is incorrectly set in the middle of an application message, it creates a message misalignment. This is similar to the noise due to having random data in the protocol collection. Note that we did not observe this odd behavior in any of our traces.

In addition, we assume that the data in-between two PSH flags belongs to a single application message. However, this is not true all the times, since multiple application messages can be contained in between two consecutive PSH flags due to TCP buffering. This situation can be mitigated if the protocol shows different types of messages with multiple lengths allowing FieldHunter to split the "single" application message into multiple correct ones.

Binary protocols have a header and a payload:

We assume that every binary protocol has similarly structured messages – each with a common header and a payload that may change. However, this assumption may not be true in all cases. For instance, some protocols can have messages with different formats for handshakes or preamble communication and subsequently, when connection is established all communication uses standard uniform messages. FieldHunter is not designed to discriminate among these two types of message formats. This can be overcome by using a preprocessing module that splits the conversations to ensure that the messages having different formats are in separate collections. Moreover, FieldHunter can still provide meaningful results if this situation happens. However, the quality of the final result depends on how popular is each type of message is in the whole collection. For instance, if flows are long lived, then the preamble message will be less popular than the rest of the messages, and the final result will contain fields of the protocol format that follows the preamble.

Another characteristic of FieldHunter is that it cannot differentiate between message header and payload. So even though, we target field inference from the header, sometimes FieldHunter ends up identifying fields contained in the payload part of the protocol.

Dataset contains pure protocols:

Our expectation is that the protocol collections are pure since FieldHunter is sensitive to noise in the form of different protocol format types. However, traffic classifiers such as a DPI can produce false positives, which means flows belonging to other protocols are classified as the protocol of our interest. FieldHunter is not able to detect the presence of this noise, and will underperform depending on how bad is the noise level.

Fields in textual protocols are Key-Value paired:

FieldHunter looks for key-value pairs found in textual protocols. It does not take advantage of other characters that provide richer structure to textual protocols such as parenthesis to enclose one object, or commas used in enumeration. When complex textual protocols are processed by FieldHunter it can still obtain valuable information. However, it can produce a less richer result set that fails to take into account the structure of the data.

ICS protocols are similar to "traditional" network protocols:

Our observation is that many of the ICS network protocols use traditional transport UDP/TCP network protocols. Hence, our assumption is that these protocols can be profiled using FieldHunter. However, there are other differences in the environments which impose further challenges that require special attention. For instance, the diversity in such networks is more difficult to reach given the applications or setups of those networks. This makes

it more difficult to correctly identify some fields since the patterns do not show up in the traffic collection, or in some cases, the final result can overfitted to the training data given the lack of samples that truly represent the protocol behavior. However, we note that for the anomaly detection application that we described, we do not need to infer all the fields of the protocol. Hence, FieldHunter is very effective for this application.

7. Related Work

Protocol Format Reverse Engineering using Network Traces: Automatic inference of protocol formats from passive network monitoring was first addressed by Beddoe [25]. The authors applied the Needleman-Wunsch algorithm for alignment of byte sequences between network payloads. The same algorithm has been used in Scriptgen [26] and RolePlayer [9] for automating the process of learning protocols in honey-nets. Their works aim to find variant and invariant segments in textual protocols. In contrast, our aim is to identify a broader selection of field types.

The problem of extracting message format specification for security applications was later addressed by Discoverer [8]. They first clustered messages with similar formats together using sequence alignments and then identified parts of the messages that change across flows. In contrast to FieldHunter, Discoverer has the same limitations as in [25, 26, 9], where fields of the protocols are expected to appear in a predefined order. In [12] authors propose Prodecoder that uses semantic information for field extraction, by using the LDA model. Their approach looks promising for identifying keys and the syntax of textual protocols, but it is not clear how LDA can properly merge n-grams into fields of binary protocols.

Protocol Format Reverse Engineering using Binary Analysis: Other authors have tackled the problem of protocol reverse engineering by using binary analysis. For instance Prospex [6] is a system that analyzes both binary execution traces combined with network traffic. Binary analysis requires an instrumented system with enough privileges to read protected memory of the application that uses the protocol. Similar in spirit, in Dispatcher [27] the authors focused on protocol reverse-engineering for botnet infiltration. All the above works rely on binary analysis and they are therefore very different from what we want to achieve with FieldHunter, where we only have passive access to network traffic.

Protocol Network Signature Generation from Network Traces: In [11, 28, 10, 7, 13] authors automatically derive protocol signatures purely from network traces. In PEXT [10] and ReveX [7] signatures are extracted for protocols using similar tokens to cluster flows. On the other hand [28] uses semantic information found in the protocol to group messages with similar formats. Authors in [13] propose a system that can automatically produce signature for botnets' command and control traffic. Obtaining automatically generated signatures for traffic classification has multiple positive implications. However, understanding the mechanics of the semantic of the protocols is a valuable complementary information for the system experts to verify the quality of automatically generated signatures.

Payload Based Anomaly Detection Systems: Systems able to detect anomalies from network traffic observing protocol payload, have been previously proposed in [29, 30, 31, 32]. These anomaly detectors define signatures on patterns found in network payloads, from which they create models and baselines. Our proposed system differs from all of them, because we use a generic abstraction of protocol format to create rules and baselines. This allows easier false positive back tracking, since the system can explain better the cause and context of a violation. For example, when a new value for an opcode has been observed. Moreover, as far as we are aware, none of these systems have been tested in industrial control networks.

ICS Anomaly Detection Systems: In the recent years there has been an interest on developing anomaly based IDS for ICS. Authors in [21] propose a system that uses clustering to detect anomalies. Their system does not necessary look only at network traffic, but also at other features that may come from other logs such as engine speeds and temperature. Another work that is more closely related to ours is [22]. Here, the authors use known protocol descriptions to extract field values from network traffic, which they use to create models to detect sequence attacks. The system that we have developed shares some similarities with their solution. However, a crucial difference is that they assume that they know the protocol specification. This is an unrealistic assumption as many of the protocols used by industrial devices are poorly documented.

FieldHunter is complementary to other systems for extraction of protocol message format, as our final goal is to identify containers/fields of information. Moreover, we present a specific application for FieldHunter in a critical security context, using data obtained from a realistic scenario.

8. Conclusions

In this paper, we presented FieldHunter, a system that automatically infers protocol field types from passive observation of network traffic. We showed that FieldHunter is able to provide a comprehensive set of fields and their types for both textual and binary protocols that may not have a publicly available specification. Therefore, we believe that a system such as FieldHunter can significantly improve the effectiveness of modern network security tools.

Finally, we extended FieldHunter and built a payload-based anomaly detection system on top of it. FieldHunter provides valuable information about network protocol specification, allowing it to detect realistic zero-day

attacks on ICS network. Our anomaly detection system can detect stealthy attacks in ICS systems with undocumented protocols that current statistical-based or traditional payload-based anomaly detection systems can not.

References

[1] Z. Li, G. Xia, H. Gao, Y. Tang, Y. Chen, B. Liu, J. Jiang, Y. Lv, Netshield: massive semantics-based vulnerability signature matching for high-speed networks, ACM SIGCOMM Computer Communication Review 41 (4) (2011) 279–290.

[2] J. Caballero, H. Yin, Z. Liang, 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, ACM, 2007, pp. 317–329.

[3] P. M. Comparetti, G. Wondracek, C. Kruegel, E. Kirda, Prospex: Protocol specification extraction, in: Security and Privacy, 2009 30th IEEE Symposium on, IEEE, 2009, pp. 110–125.

[4] W. Cui, M. Peinado, K. Chen, H. J. Wang, L. Irun-Briz, Tupni: Automatic reverse engineering of input formats, in: Proceedings of the 15th ACM conference on Computer and communications security, ACM, 2008, pp. 391–402.

[5] Z. Lin, X. Jiang, D. Xu, X. Zhang, Automatic protocol format reverse engineering through context-aware monitored execution., in: NDSS, Vol. 8, 2008, pp. 1–15.

[6] G. Wondracek, P. M. Comparetti, C. Kruegel, E. Kirda, S. S. S. Anna, Automatic network protocol analysis., in: NDSS, Vol. 8, 2008, pp. 1–14.

[7] J. Antunes, N. Neves, P. Verissimo, Reverse engineering of protocols from network traces, in: Reverse Engineering (WCRE), 2011 18th Working Conference on, WCRE '11, IEEE, Limerick, IR, 2011, pp. 169–178.

[8] W. Cui, J. Kannan, H. J. Wang, Discoverer: Automatic protocol reverse engineering from network trace, in: Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium, USENIX Security '07, USENIX Association, Boston, MA, 2007, pp. 1–14.

[9] W. Cui, V. Paxson, N. Weaver, R. H. Katz, Protocol-independent adaptive replay of application dialog., in: NDSS, 2006.

[10] M. Shevertalov, S. Mancoridis, A reverse engineering tool for extracting protocols of networked applications, in: Reverse Engineering, 2007. WCRE 2007. 14th Working Conference on, WCRE '07, IEEE, Vancouver, BC, CA, 2007, pp. 229–238.

[11] A. Tongaonkar, R. Keralapura, A. Nucci, Santaclass: A self adaptive network traffic classification system, in: IFIP Networking Conference, 2013, IEEE, 2013, pp. 1–9.

[12] Y. Wang, M. Z. Shafiq, L. Wang, A. X. Liu, Z. Zhang, D. Yao, Y. Zhang, L. Guo, A semantics aware approach to automated reverse engineering unknown protocols, 2012 20th IEEE International Conference on Network Protocols (ICNP) (2012) 1–10doi:10.1109/ICNP.2012.6459963.

[13] C. Rossow, C. J. Dietrich, Provex: Detecting botnets with encrypted command and control channels, in: Proceedings of the 10th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, DIMVA'13, Springer-Verlag, Berlin, Heidelberg, 2013, pp. 21–40.

[14] N. Falliere, L. O. Murchu, E. Chien, W32. stuxnet dossier, White paper, Symantec Corp., Security Response 5.

[15] 2015 dell security annual threat report, Tech. rep., Dell Inc. (2015).
URL `https://software.dell.com/whitepaper/dell-network-security-threat-report-2014874708`

[16] C. Kreibich, J. Crowcroft, Honeycomb: creating intrusion detection signatures using honeypots, ACM SIGCOMM Computer Communication 34 (1).

[17] Tcp statistic and analysis tool.
URL `http://tstat.polito.it`

[18] Y. Yao, Information-theoretic measures for knowledge discovery and data mining, in: Entropy Measures, Maximum Entropy Principle and Emerging Applications, Springer, 2003, pp. 115–136.

[19] Opendpi.
URL `https://code.google.com/archive/p/opendpi/`

[20] Distributed hash table (Oct. 2012).
URL `http://wiki.vuze.com/w/Distributed_hash_table`

[21] I. Kiss, B. Genge, P. Haller, G. Sebestyen, Data clustering-based anomaly detection in industrial control systems, in: Intelligent Computer Communication and Processing (ICCP), 2014 IEEE International Conference on, 2014, pp. 275–281.

[22] M. Caselli, E. Zambon, F. Kargl, Sequence-aware intrusion detection in industrial control systems, in: Proceedings of the 1st ACM Workshop on Cyber-Physical System Security, CPSS '15, ACM, New York, NY, USA, 2015, pp. 13–24.

[23] A. Hahn, A. Ashok, S. Sridhar, M. Govindarasu, Cyber-physical security testbeds: Architecture, application, and evaluation for smart grid, Smart Grid, IEEE Transactions on 4 (2) (2013) 847–855.

[24] Overview of the dnp3 protocol.
URL `http://www.dnp.org/pages/aboutdefault.aspx`

[25] M. A. Beddoe, Network protocol analysis using bioinformatics algorithms., Technical report, Baseline research (2005).

[26] C. Leita, K. Mermoud, M. Dacier, Scriptgen: an automated script generation tool for honeyd., in: Computer Security Applications Conference, 21st Annual, CSAC '05, IEEE, Tucson, AZ, 2005, pp. 214–226. doi:10.1109/CSAC.2005.49.

[27] J. Caballero, P. Poosankam, C. Kreibich, 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 '09, ACM, Chicago, IL, 2009, pp. 621–634.

[28] V. Yegneswaran, J. T. Giffin, P. Barford, S. Jha, An architecture for generating semantics-aware signatures, in: USENIX Security, 2005, pp. 34–43.

[29] K. Wang, S. Stolfo, Anomalous payload-based network intrusion detection, in: E. Jonsson, A. Valdes, M. Almgren (Eds.), Recent Advances in Intrusion Detection, Vol. 3224 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2004, pp. 203–222.

[30] D. Bolzoni, S. Etalle, P. Hartel, E. Zambon, Poseidon: a 2-tier anomaly-based network intrusion detection system, 2010 International Workshop on Innovative Architecture for Future Generation High Performance 0.

[31] K. Wang, J. Parekh, S. Stolfo, Anagram: A content anomaly detector resistant to mimicry attack, in: D. Zamboni, C. Kruegel (Eds.), Recent Advances in Intrusion Detection, Vol. 4219 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2006, pp. 226–248.

[32] R. Perdisci, D. Ariu, P. Fogla, G. Giacinto, W. Lee, Mcpad: A multiple classifier system for accurate payload-based anomaly detection, Computer Networks 53 (6) (2009) 864 – 881, traffic Classification and Its Applications to Modern Networks.