Automatic Protocol Field Inference for Deeper Protocol Understanding

Ignacio Bermudez, Alok Tongaonkar
Symantec Corp.
{ignacio_bermudezcorr,
alok_tongaonkar}@symantec.com

Marios Iliofotou
Caspida Inc.
marios@caspida.com

Marco Mellia, Maurizio M. Munafò
Politecnico di Torino
{marco.mellia, maurizio.munafo}@polito.it

***Abstract*—Security tools have evolved dramatically in the recent years to combat the increasingly complex nature of attacks, but to be effective these tools need to be configured by experts that understand network protocols thoroughly. In this paper we present FieldHunter, which automatically extracts fields and infers their types; providing this much needed information to the security experts for keeping pace with the increasing rate of new network applications and their underlying protocols. FieldHunter relies on collecting application messages from multiple sessions and then applying statistical correlations is able to infer the types of the fields. These statistical correlations can be between different messages or other associations with meta-data such as message length, client or server IPs. Our system is designed to extract and infer fields from both binary and textual protocols. 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 nature 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) and from malware such as (Ramnit).**

I. Introduction

In recent years the attacks against networks have become more complicated. To defend against these complex attacks, network defense tools have also evolved to use more sophisticated mechanisms. For instance, firewalls have moved from using simple packet-filtering rules to application level rules [1] that need deeper understanding of the protocols being used by network applications. Similarly, intrusion detection systems are increasingly using vulnerability based signatures [2] that contain information specific to network protocols. Access control mechanisms are also evolving from IP address based policies to fine-grained policies which use the protocol objects such as users and message types.

It is clear that configuring all of the above applications require a deeper understanding of network protocols. However comprehending protocol specifications is a very tedious task. The traditional approach of manually reverse engineering a protocol cannot cope with the rate at which new network applications are made available to the market and brought into workplace. Moreover, many of the proprietary protocols' specifications are not publicly available. As a result, security administrators have to configure these security tools 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, either from binary code analysis [3]–[7] or from network traffic [8]–[14]. Most of the times network application binaries are not available to the network operators limiting the usefulness of code analysis based techniques. Hence, we focus on network traffic based analysis. The state-of-the-art techniques in this area try to infer message formats or underlying protocol state machines.

In this paper, we take a complimentary approach of identifying field boundaries and inferring the field types for protocols. Thus, we study well known protocols and identified a set of field types that can 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 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/acknowledgement numbers and (vi) Accumulators such as generic counters and timestamps. We note that a protocol may not have all the above types of fields. Also, different fields may be useful for different security applications. For instance, Host IDs can be used by access control systems while Message Length can be used by Intrusion Prevention Systems to prevent buffer overflow kind of attacks.

We built a system, called FieldHunter that uses a two step methodology: (i) Field extraction: here we extract fields from the protocol messages. Due to the different characteristics of fields in textual and binary protocols, we use techniques specific to each type to extract fields. (ii) Field type inference: this step is common for the fields extracted in both textual and binary protocols. 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 that we were able 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. In addition we fed FieldHunter with Ramnit command and

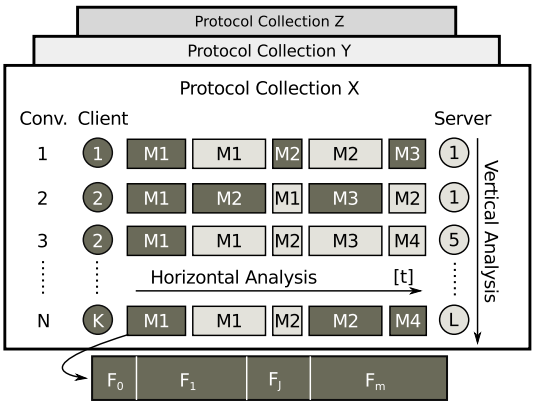

Fig. 1. Terminology diagram.

control traffic extracted from the same traces. Ramnit traffic is unencrypted first and then analyzed by FieldHunter, which represents a milestone for our system, that shows how useful it is on providing valuable information about specific security applications such as this malware.

The rest of the paper is organized as follows. §II defines the terminology used throughout the paper, §III provides details about the core algorithms used by FieldHunter to deal with binary and textual protocols. Performance evaluation and parameter tuning is presented in §IV. We discuss related works in §V and finally conclude the paper in §VI.

II. Terminology

Figure 1 gives a pictorial representation of the terminology we use throughout this paper. Our methodology takes as input a set of **conversations** (i.e., flows defined by the usual 5-tuple) 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[1](light-colored). Messages consist of different pieces of information enclosed in **fields**. As we show in the Fig. 1, conversations evolve horizontally over time (t) and messages can be compared vertically across multiple conversations.

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 (Note that (i) and (ii) are very similar to vertical and horizontal sub-collections as defined in [15]). (iii) Grouping together messages by direction, e.g., all C2S messages. Message grouping is instrumental for FieldHunter to find patterns in the collections. Then 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 testbed 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, or by relying on a Deep Packet Inspection (DPI) classifier [16].

[1]Client is the initiator of the conversation and the server is the other end. Hosts are identified by their IP address.

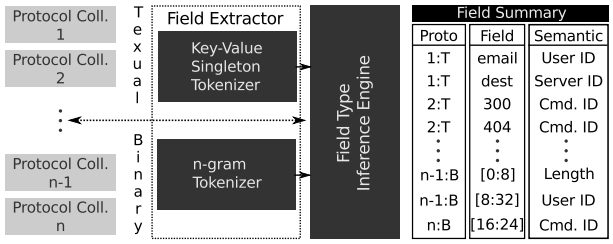

Fig. 2. FieldHunter system diagram.

Application conversations are transported by TCP/UDP segments and are extracted by FieldHunter using the following methodology: for messages transported over UDP it is assumed that each segment contains one application message; 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 message length has been identified by FieldHunter.

III. 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 Fig 2.

A. Field Extractor

Textual and binary protocols differ greatly in the way fields are used. Textual protocols typically use *delimiters* such as "`:`" or "`0x0D0A`" 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.

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 key and value in a field. We categorize delimiters in three 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. (iii) **Value-value delimiter** (D_{v-v}) separates different values for the same field, e.g., the comma ',' in the field `AllowedPorts: 4534, 80, 53`. FieldHunter identifies D_f and passes fields along with their D_{k-v} to the Field Type Inference Engine. Our implementation does not analyze D_{v-v}. Hence we do not discuss this type of delimiter in the paper.

Common choices of D_f and D_{k-v} delimiters are shown in Table I. The delimiters are derived using documentation

TABLE I
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	0x0D0A	':', ' '	FTP	0x0D0A	' '
SMTP	0x0D0A	':', ' ', '-'	TFTP	0x1D	0x1E
POP3	0x0D0A	':', ' '	CS	0x5C	0x5C
RTSP	0x0D0A	':'	GNU	0x0D0A	':', ' '
SIP	0x0D0A	':', ' '	RTP	0x0D0A	':', ' '
GAME	0x00	0x00	MSN	0x0D0A	' '
TEL	0x0D0A	':', ' '			

of the listed protocols, and are actually observed in our data sets. As we see, there are popular delimiters, such as `0x0D0A` (carriage-return and line-feed pair), 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.

a) **Field Delimiter Inference:** To infer the D_f, 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 check by comparing common prefixes and exact matches respectively.

b) **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 taken 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.

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. Unfortunately FieldHunter, does not know the message data structure. To overcome this FieldHunter splits messages into n-grams which are used by Field Type Inference Engine. We observe that for most of the field types, the n-grams in the field also show characteristics similar to the field. For instance, if a protocol 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-grams 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 thousand), 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 consider byte endianness for fields that contain numerical representations, heuristics are repeated trying both little-endian and big-endian. This is not the case for fields that can be interpreted as categorical representations.

B. 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. For clarity, in the rest of the paper we use the term "n-gram" to interchangeably mean "binary n-gram" or "textual field", e.g., when it is stated "*n-gram entropy is computed*" actually it means that either "binary n-gram entropy is computed" or "textual field entropy is computed". On the other hand, 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 correlates 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 acknowledgement 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

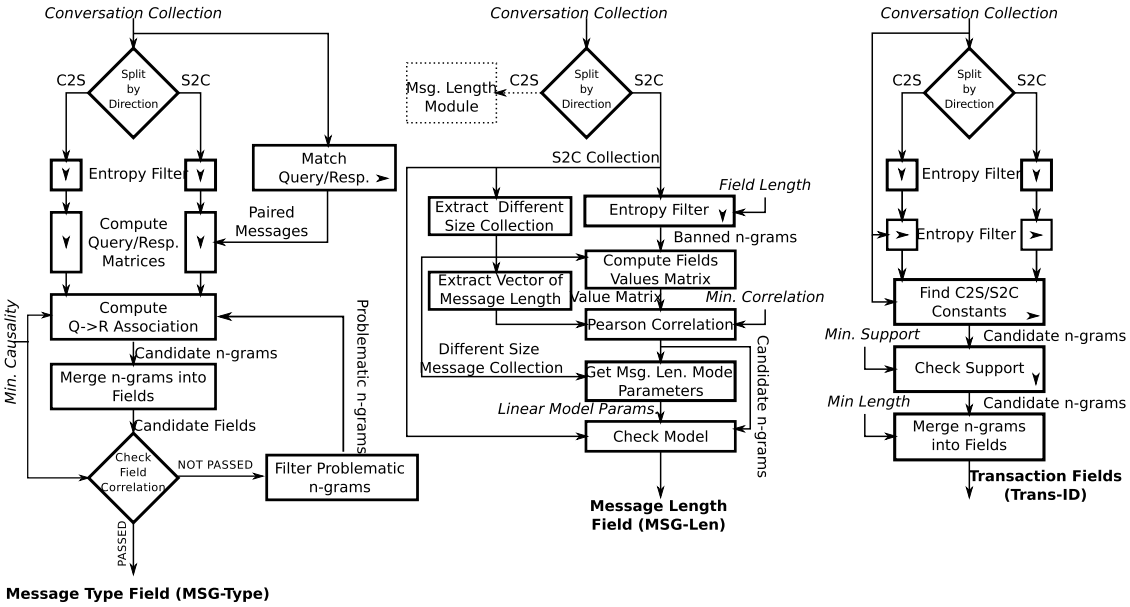

Fig. 3. MSG-Type (left), MSG-Len (center) and Trans-ID (right) modules.

highlighted in italic. More details on parameter selection are given is §IV-D.

1) **Message 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 receivers to understand what type of message is received, e.g., a request, a status update, an error message, etc.

Our methodology is based on two key observations: (i) MSG-Types takes values from a well defined static and small set; and (ii) represents transitions in an underlying protocol state machine. Hence, by pairing request/response messages, there is a high chance that their corresponding MSG-Type fields are related. The leftmost diagram in Fig. 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 taking 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 fields are discarded, according to observation (ii), the next step is to 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) [17].

FieldHunter takes n-grams for which causality is greater than a threshold of 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 so that $q1 \Rightarrow r1$, $q5 \Rightarrow r5$, $q6 \Rightarrow r6$. Then it checks whether the groups $(q1, q5, q6) \Rightarrow (r1, r5, r6)$ holds the causality. If this holds, the field containing n-grams at offsets $(1, 5, 6)$ are returned as the MSG-Type field.

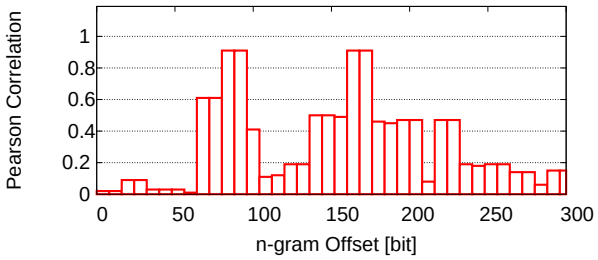

Fig. 4. n-gram correlation with MSG-Len for SopCast.

2) **Message Length (MSG-Len):** Our goal here is to find fields that report the length of the application message. As such, it is expected that MSG-Len field is linearly correlated with the actual message size. For higher confidence, two different tests are used for identifying linear correlations.

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 MSB and LSB are not correlated in this case. Hence, FieldHunter iteratively selects n-gram windows of size 32, 24, 16-bit that are shifted at a step of 8-bit. 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. E.g. 98% of messages are length 40 bytes. To avoid such biases, we stratify messages by length, creating in this way a size heterogeneous sub-collection not affected by the bias problem. We select all the fields such coefficient is above a certain threshold as MSG-Len candidates. Empirically we found 0.6 to be a good threshold.

Figure 4 show the results of applying the Pearson correlation to the SopCast protocol collection obtained from the one of our traces. In this example, we use 16-bit n-gram. Coefficient spans 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 (see § IV-B1). We cross-verified these results using information extracted by manual protocol reverse engineering attempts found on the Internet.

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

3) **Host Identifier (Host-ID):** Identifies entities beyond the network addresses. Its functionality permits network omnipresent identification of a particular host or persona. For instance, in peer-to-peer applications, a "Peer-ID" field can uniquely identify a specific peer/host in the whole overlay, even when the peer is behind NAT or is moving over multiple networks.

The heuristic assumes that all 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 *minimum length (4 Bytes)*. Notice that the adoption of statistical tests, such as correlation, makes algorithms robust to handle traffic where assumption might not always hold; such as when NAT is used.

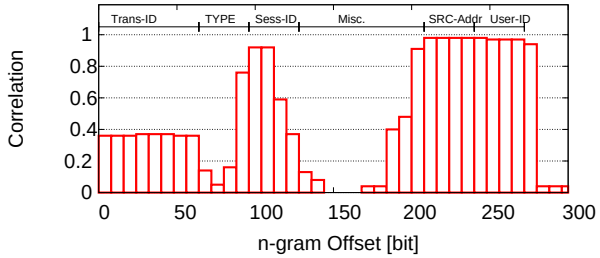

Fig. 5. n-gram correlation with Client IP (Vuze DHT).

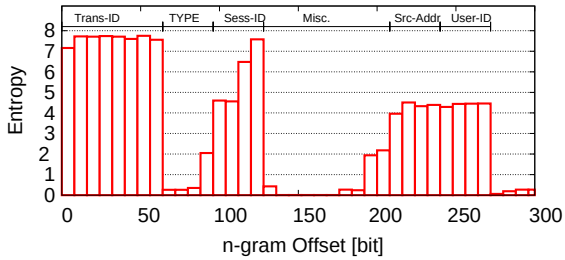

Fig. 6. The n-gram entropy for Vuze DHT over a C2S vertical sub-collection.

Figure 5 shows the categorical correlation between n-grams in a vertical collection and the corresponding source IP address for the Vuze DHT collection. Note how $R(x, y)$ is very close to one (high correlation) for n-grams that represents 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.

4) **Session Identifier (Session-ID):** 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.

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 measure using entropy as before.

Figure 6 shows the entropy of n-grams for the Vuze DHT protocol [18] taken from the same trace as above. 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/field 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 format can change and Trans-ID may appear at different offsets (e.g. in Vuze DHT). Therefore, the heuristic does not assume the protocol message formats are the same in for 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, e.g., caused by message reordering or retransmission in the collection. Finally, consecutive n-grams are merged to form a field of at least *minimum length (2 Bytes)*. For textual protocols such n-gram merging is not needed.

6) **Accumulators:** These fields typically represent message sequence numbers, acknowledgement numbers, timestamps, etc. Thus, we search for fields that have their values increasing over consecutive message within the same conversation. To identify such fields, we use 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 quite variable Δ.

We search for accumulators in C2S and S2C directions independently with each other. As with the MSG-Len field, here we do start with fixed size n-grams. We assume accumulators are encoded in fields of a given *field length*, e.g, 64, 32 and 16 bit. 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 (e.g., 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".

C. Field Summary

As final result FieldHunter provides information of the field type extracted automatically out of known/unknown protocols. 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.

IV. Experimental results

This section presents the result of running FieldHunter using ISP packet traces reported in § IV-A. A DPI tool 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, e.g., showing only conversation exchanged with a handful of servers, etc. 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 § IV-D for more details. All our parameter selection is made using TR1-2012, whereas we tested FieldHunter on all three traces.

TABLE II
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

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

A. Datasets

We evaluate FieldHunter using three different traces (Table II). Data were collected from different geographic regions (Asia, Europe, and South America), between years 2007-2012. All traces contain full payload from network connections. Given the large size of the TR1-2012 trace we limited the payload per connection to the first 1048 bytes[2].

B. Evaluation of Binary Protocols

Table III reports the results for nine binary protocols. Seven of these protocols have known specifications. The table reports the number of discovered and Ground-Truth (GT) bits, for both C2S and S2C collections. Note that in many cases in Table III, the number of discovered bits is larger than the GT bits. This is because many protocols such as ED2K carry other protocols. Since FieldHunter works on complete payload of the conversation, it identifies fields within the inner protocol as well. This is a limitation of how the DPI generates the collection. The average AoC (Accuracy over Coverage) 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 (e.g., a 32-bit long number), FieldHunter easily identifies the less significant bits, but tends to miss the most significant bits. This is because the latter appear as "constant", i.e., data do not variate and statistical inference cannot take place. Finally, the last two rows in Table III show two closed protocols, such as SopCast and PPLive. From the results depicted in the table, we show details for three interesting case studies.

1) **ED2K and KADEMLIA:** ED2K and KADEMLIA eMule messages are preceded by a common header which is used as GT. FieldHunter correctly identifies such common header. Moreover it discovers additional fields, that sum up to

[2]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.

TABLE III
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/?	?/?	?/?

TABLE IV
Summary of the results from running FieldHunter on the textual protocols

Protocol	#Fields	K-V	CMD	IDs	FP-IDs
	C2S/S2C	C2S/S2C	C2S/S2C	C2S/S2C	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	*/17	*/15	*/2	*/2	*/0
RSP	3/*	2/*	1/*	1/*	0/*

a total of 128 bits in the C2S EDK2 collection. After manual inspection, we observed those fields to correctly include key hash information, Session-ID, Host-ID, etc.

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.

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

3) **Domain Name Service (DNS):** For DNS in the TR1-2012 trace, FieldHunter successfully identifies the Trans-ID and a MSG-Type fields, each of 16 bits. We expect parsing DNS on 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 local DNS server). Despite this, FieldHunter correctly identifies some protocol fields.

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

C. Evaluation on Textual Protocol

Table IV reports overall results for textual protocols as 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), detailing those that proved to be False Positives (FP). Here, by IDs we refer Host-IDs and Session-IDs and Transaction-IDs. Overall, from the 26 fields labeled as IDs, 22 are manually verified as correct identifiers and only four are false positives. In general, we observe that the majority of the fields of textual protocols are successfully inferred by FieldHunter in both the C2S and S2C directions. As for the binary protocols, we pick two interesting textual protocols as case study.

1) **Microsoft Notification Protocol (MSNP):** The MSN protocol is present in all three traces. FieldHunter correctly finds that `USR` field carries a Host-ID and indeed, it carries the MSN's user name. Another interesting field is `CVR` 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.

2) **Real-Time Streaming Protocol (RTSP):** The S2C direction of this protocol gives many inferred ID fields. Out of these six IDs, 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 FP fields are `Last-Modified` and `Cache-Control`. For instance, `Last-Modified` is the timestamp of the last modification for a given content. Since one single object is requested using multiple RTSP conversations, its modification time appears constant across conversations on the same IP pair. Finally, the `Cache-Control` tends to take always 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.

D. Sensitivity Analysis & Parameter Tuning

We evaluate the sensitivity of FieldHunter to different parameters and to external factors, such as the number of conversations needed. As mentioned before, we perform parameter tuning using one trace, and then we evaluate FieldHunter on all three traces. We show next, the design proved to be robust to parameter tuning. This is partly confirmed by the results in Figs. 6, 4, 5, which show a clear field inference.

In our tuning, first we focus on one of the most challenging field to infer, the MSG-Type for binary protocols. To tune parameter, we take all collections for those protocols where we

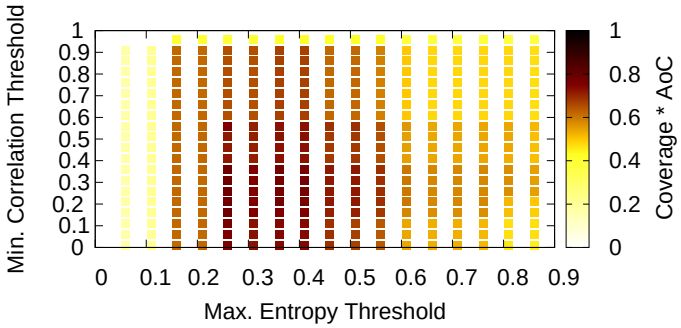

Fig. 7. Parameter sensitivity for the MSG-Type.

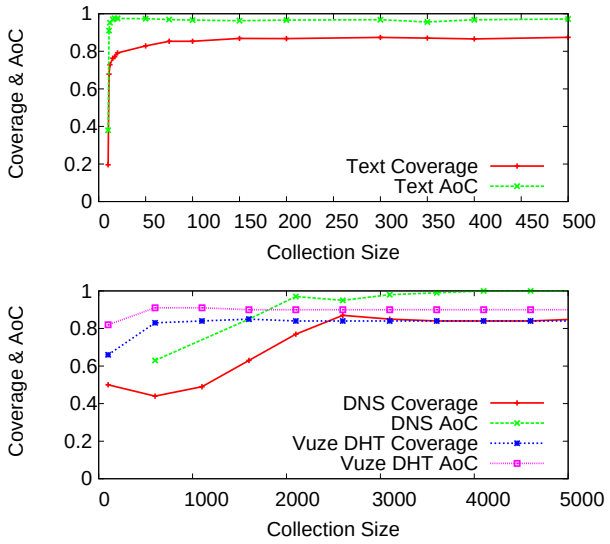

Fig. 8. Coverage and AoC versus the number of conversations. text-based protocol (top); DNS and Vuze DHT binary-based protocols (bottom).

have the 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 Fig 7. The darker the block, the better FieldHunter performs. As observed, there is a large range of good parameters that yield scores above 0.8, which means that in most cases FieldHunter is able to correctly pinpoint the MSG-Type field. For other field types, we observed qualitatively similar results and we do not show them here due to space constraints.

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 the fields present in our traces. Then, we randomly extract a reduced subset of conversations from the collections and run FieldHunter over the subset. Results are compared against our ground truth to compute the Coverage and AoC (Fig. 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 are always successful in identifying the D_f delimiter for all the protocols we test. Most of the mis-labeling happens due to challenges in inferring the D_{k-v} for some fields.

For binary, we repeat the same experimental setup as before, but focusing on two extreme protocol cases, DNS and Vuze DHT. The results are shown in the bottom plot of Figure 8. As we can see, the two protocols react differently and also differ from textual protocols (note the different scales in the x-axis). 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 have highest diversity, with many end-points exchanging messages. Conversely, DNS (from TR1-2012) represents a challenging scenario due to little diversity in the collections: typically only one MSG-Type (DNS requests) is found, and conversations are very short (a single request/response). As we see, eventually we achieve very good results for DNS, but it requires as many as 2,000 conversations.

V. Related Work

Automatic inference of protocol formats from passively network monitoring was first addressed by Beddoe [19]. Here authors applied the Needleman-Wunsch algorithm for alignment of byte sequences between network payloads. The same algorithm has been used in Scriptgen [20] and RolePlayer [10] for automating the learning of protocols in honey-nets. Their works aim to find variant and invariant segments in textual protocols, although 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 [9]. 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 [10], [19], [20], where fields of the protocols are expected to appear in predefined order. In [13] authors propose Prodecoder that uses semantic information for field extraction, by using the LDA model. Their approach looks promising on 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.

In [8], [11], [12], [14], [21] authors derive automatically protocol signatures purely from network traces. In PEXT [11] and ReveX [8] signatures are extracted for protocols using similar tokens to cluster flows. Differently [21] uses semantic information found in the protocol to group messages with similar formats. Authors in [14] propose a system that automatically can produce signature for botnets' command and control traffic. This is an interesting application since command and control traffic is obscured and undocumented by default. Besides our work does not aim to obtain signatures for network protocols, FieldHunter can help considerably

with such applications as mentioned in [14] future works. Automatically generated signatures can be good for classifying traffic, but understanding the mechanics and semantic of the protocol is valuable complimentary information for a system expert to verify the quality of the signatures. Indeed, preliminary experiments on decrypted Ranmit traffic shows that FieldHunter is able to identify some binary fields of its Command and Control protocol header.

Other authors have tackled the problem of protocol reverse engineering by using binary execution analysis. For instance Prospex [7] 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 [22] 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.

Our technique is complimentary to other systems that aim to extract message format and syntax at the time that our goal is to identify containers/fields of information, and as such it can potentially improve their results.

VI. Conclusions and Future Work

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.

As future work we want to extend FieldHunter to infer fields from not well documented protocols of mobile applications. Such knowledge can be used to create fine grained policy engines to block specific events like uploads or downloads, in enterprise networks where users bring their own devices and connect to Internet.

Acknowledgments

We acknowledge the help of Lorenzo De Carli, who provided us unencrypted samples of Ramnit command and control traffic extracted from one of our datasets. Without his help we could not prove how useful is FieldHunter on identifying automatically fields for this very specific kind of applications such as malware.

References

[1] Checkpoint Application Intelligence. http://www.checkpoint.com/products/technologies/ai.html.

[2] Z. Li, G. Xia, H. Gao, Y. Gao, Y. Chen, B. Chen, J. Jiang, and Y. Lv. NetShield: Massive Semantics-based Vulnerability Signature Matching for High-speed Networks. In *ACM SIGCOMM*, 2010.

[3] J. Caballero, H. Yin, Z. Liang, and D. Song. Polyglot: Automatic Extraction of Protocol Message Format using Dynamic Binary Analysis. In *ACM Conference on Computer and Communications Security*, 2007.

[4] P. Comparetti, G. Wondracek, C. Kruegel, and E. Kirda. Prospex: Protocol Specification Extraction. In *IEEE Security and Privacy*, 2009.

[5] W. Cui, M. Peinado, K. Chen, H. Wang, and L. Irun-Briz. Tupni: Automatic Reverese Engineering of Input Formats. In *ACM Conference on Computer and Communications Security*, 2008.

[6] Z. Lin, X. Jiang, D. Xu, and X. Zhang. Automatic Protocol Format Reverse Engineering through Context-Aware Monitored Execution. In *Symposium on Network and Distributed System Security*, 2008.

[7] Gilbert Wondracek, Paolo Milani Comparetti, Christopher Kruegel, and Engin Kirda. Automatic network protocol analysis. In *15th Symposium on Network and Distributed System Security (NDSS)*, NDSS '08, San Diego, CA, 2008. ISOC.

[8] Joao Antunes, Nuno Neves, and Paulo Verissimo. Reverse engineering of protocols from network traces. In *Reverse Engineering (WCRE), 2011 18th Working Conference on*, WCRE '11, pages 169–178, Limerick, IR, 2011. IEEE.

[9] Weidong Cui, Jayanthkumar Kannan, and Helen J Wang. Discoverer: Automatic protocol reverse engineering from network trace. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, USENIX Security '07, pages 1–14, Boston, MA, 2007. USENIX Association.

[10] Weidong Cui, Vern Paxson, Nicholas Weaver, and Randy H Katz. Protocol-independent adaptive replay of application dialog. In *Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS)*, NDSS '06, San Diego, CA, 2006. ISOC.

[11] Maxim Shevertalov and Spiros Mancoridis. A reverse engineering tool for extracting protocols of networked applications. In *Reverse Engineering, 2007. WCRE 2007. 14th Working Conference on*, WCRE '07, pages 229–238, Vancouver, BC, CA, 2007. IEEE.

[12] A. Tongaonkar, R. Keralapura, and A. Nucci. SANTaClass: A Self Adaptive Network Traffic Classification System. In *TC6/IFIP Networking*, 2013.

[13] Yipeng Wang, M. Zubair Shafiq, Liyan Wang, Alex X. Liu, Zhibin Zhang, Danfeng Yao, Yongzheng Zhang, and Li Guo. A semantics aware approach to automated reverse engineering unknown protocols. *2012 20th IEEE International Conference on Network Protocols (ICNP)*, pages 1–10, October 2012.

[14] Christian Rossow and Christian 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, pages 21–40, Berlin, Heidelberg, 2013. Springer-Verlag.

[15] Christian Kreibich and Jon Crowcroft. Honeycomb: creating intrusion detection signatures using honeypots. *ACM SIGCOMM Computer Communication*, 34(1), 2004.

[16] A. Tongaonkar, R. Torres, M. Iliofotou, R. Keralapura, and A. Nucci. Towards self adaptive network traffic classification. *Elsevier's Commuter Communications Journal*, April 2014.

[17] YY Yao. Information-theoretic measures for knowledge discovery and data mining. In *Entropy Measures, Maximum Entropy Principle and Emerging Applications*. Springer, 2003.

[18] Vuze Wiki. Distributed hash table, October 2012.

[19] Marshall A Beddoe. Network protocol analysis using bioinformatics algorithms. Technical report, Baseline research, 2005.

[20] C. Leita, K. Mermoud, and M Dacier. Scriptgen: an automated script generation tool for honeyd. In *Computer Security Applications Conference, 21st Annual*, CSAC '05, pages 214–226, Tucson, AZ, 2005. IEEE.

[21] Vinod Yegneswaran, Jonathon T Giffin, Paul Barford, and Somesh Jha. An architecture for generating semantics-aware signatures. In *USENIX Security*, pages 34–43, 2005.

[22] Juan Caballero, Pongsin Poosankam, Christian Kreibich, and Dawn 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, pages 621–634, Chicago, IL, 2009. ACM.