Traffic to Protocol Reverse Engineering

Antonio Trifilò, Stefan Burschka, Ernst Biersack

Abstract— **Network Protocol Reverse Engineering (NPRE) has played an increasing role in honeypot operations. It allows to automatically generate Statemodels and scripts being able to act as realistic counterpart for capturing unknown malware. This work proposes a novel approach in the field of NPRE. By passively listening to network traces, our system automatically derives the protocol state machines of the peers involved allowing the analyst to understand its intrinsic logic. We present a new methodology to extract the relevant fields from arbitrary binary protocols to construct a statemodel. We prove our methodology by deriving the statemachine of documented protocols ARP, DHCP and TCP. We then apply it to Kademlia, the results show the usefulness to support binary reverse engineering processes and detect a new undocumented feature.**

I. Introduction

The vision to reverse engineer the logic of unknown programs or the intent of users interacting on the Internet solemnly from the network traces is not new. Several approaches in the area of honey pot Malware analysis have been proposed mostly interested in interacting with the infected host to capture the Malware ([2], [3], [5]). The goal of their research ranges from capturing zero-day attacks to the long term gathering of data allowing to retrieve all available information about the malware's spread and activity in the network. Scripts are automatically generated being able to answer and retrieve the malware in an automatic way without a priori knowledge about the protocol. Given that their main goal is the interaction, they mainly focused on the derivation of a precise protocol message format. It allows to construct a state machine being able to answer and retrieve the malware.

In this work we go a step beyond. By passively eavesdropping an IP switch Span port we intend to reverse engineer the complete logic of the programs involved in a conversation. Therefore the traffic will not only be reproduced but its intrinsic logic, enabling the human analyst to actually understand the protocol or the human interaction in form of a comprehensible protocol state machine.

The idea for our work originates from the well established design of binary protocols in HW. A binary protocol is defined as a binary coded protocol intended to be read by a machine rather than a human being. Plain text protocols instead are human understandable and generally related to the application layer of the OSI model. In the design of binary protocols in HW binary time dependent sequences are coded directly into a complete state machine. It is straightforward to assume that it should be also possible for binary IP protocol traces, provided these bits or bytes coding into a state machine are known. Thus, for new protocols these groups of bits have to be detected in IP traces first. At least binary protocols are position dependent, meaning a Bit does not change its position over time. Modern text based protocols are more erratic; they have key value pairs which can be in any given position. Thus, the idea was to consider first a binary representation where the problem of Network Protocol Reverse Engineering (NPRE) seems to be solvable. The basic workflow solving the principal problem is outlined in Figure 1:

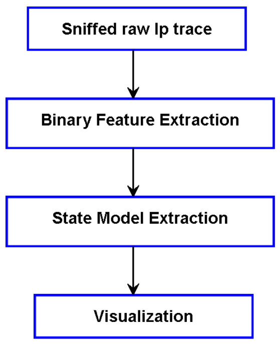

Fig. 1. Workflow for NPRE

- We record in a tcpdump file all network traffic generated by the protocol to be reverse engineered by just eavesdropping an IP switch Span port.
- We extract with a statistical analysis all relevant features in a protocol message format for understanding the logic of the protocol.
- From the selected features of the previous step we reconstruct the state machine of the protocol using our algorithms.
- A graphical representation of the state machine is provided to the human analyst.

The result of the entire workflow will be the automatic generation of the state machine of an unknown protocol from its network traces. Thus reverse engineering the logic of the protocol from its output represented by network traffic.

In order to present our method, the paper is structured as follows. Section II presents the theoretical background and algorithms for feature selection and state machine derivation. Section III provides the experimental results concerning the extraction of relevant fields and the protocol state machine derivation. Finally Section IV concludes the paper denoting the main contribution and future research.

Antonio Trifilò and Stefan Burschka are working at Swisscom Schweiz AG, Bern, Switzerland. (email: {antonio.trifilo, stefan.burschka}@swisscom.com). Ernst Biersack is in the Internet Research Group at Eurecom, Sophia Antipolis, France. (email: erbi@eurecom.fr)

We thank Vincent Lenders and Stefan Engel from armasuisse for cofinancing.

II. Methodology

A. Overview

Our approach can be described by three functional modules. For a detailed description of each module and the algorithms refer to the work of [1].

*a) **IP Sniffer**:* An IP sniffer is used to capture passively the raw traffic passing through an IP span port. The traffic is filtered appropriately selecting the protocol to be reverse engineered and then stored in a standard tcpdump file.

*b) **Binary Features Selector (BFS)**:* The basic idea is that only certain fields of a protocol message capture the logic of the protocol, while the others specify only additional parameters. For instance in the case of an HTTP request "*GET /pub/WWW/TheProject.html HTTP/1.1*" only the field "GET" is necessary to understand the basic logic of the protocol because it defines the type of action requested.

The reduction to a subset of interesting features is achieved by a statistical analysis of several flows based on the "Variance of the Distribution of Variances" (VDV) of each byte in the binary protocol message. The result indicates the relevant fields for state machine construction. For instance in the case of the Transmission Control Protocol (TCP) the Flags are selected representing the connection management logic.

*c) **State Machine Builder (SMB)**:* The SMB module is deputed to the construction of the protocol state machine starting from the features selected by the BFS. This module defines the notion of state and the proper transitions which interconnects every state. Both of these two main steps are performed by the "State Splitting Algorithm" which is the core of the SMB. The final result will be a graphical presentation of the protocol state machine to the human analyst.

In the following section we will describe in details the BFS and the SMB being the novel contribution of the paper. Before describing these two modules, a theoretical background is presented serving as a prior information being required in all traffic mining activities.

B. Background: "Father of all Protocols"

The definition of basic principles at the base of protocols is a fundamental step in the process of reverse engineering of a network protocol. It states that even a yet unknown protocol will always contain certain common concepts required for proper functionality in network environments. Consequently it allows to make assumptions and guide any mining activity and in particular the NPRE task. The following principles, termed as "Father of all Protocols", are identified concerning either the format of a message or the principle of communication:

- **Stability in time and space**: The protocol of a specific version and its message format does not change even when the experiment is repeated at different location and times. Thus several flows generated by a certain protocol are similar in behavior.
- **Grammar**: Defines the format of a message in a protocol. This assumption states that the format of a message is not randomly chosen and that a structure, i.e. grammar, is defined in order to allow a communication between two entities.
- **Positional information**: The grammar of a message follows a positional rule where left is more important than right. This assumption is true for text protocols where the left part of a message contains the command while the following part contains the parameters.
- **Error control**: During an exchange of data between two different entities via a specific protocol, an error mechanism is always present in order to deal with unknown events or situations.
- **Indication of peer state change**: This assumption states that an entity has to be able to communicate to the other peers whether a state change happened.
- **Request-Response Paradigm**: There are two different communication paradigms enabling a communication between two entities via a protocol: "Idle Repeat Request" (IRR) and "Continuous Repeat Request" (CRR). IRR allows that the sender only sends one data packet and waits for the answer of the receiver before sending another packet. Whereas CRR allows to send a collection of packets before receiving the first correct answer from the receiver.

In the following section, we will refer to these principles as background for the development of our methodologies.

C. Binary Features Selector

The BFS module identifies relevant fields in a protocol message format in contrast to the attributes which specify additional parameters. The methodology for feature selection presented here is based on a statistical analysis of the raw packets based on the "Variance of the Distribution of the Variances" (VDV) of each byte field in the protocol message.

In our IP related case the following feature extraction rules apply, representing an essence of the said "Father of all Protocol":

- Different flows of the same protocol have common underlying logic allowing the communication.
- Common logic is represented by relevant fields.
- Relevant fields always show similar behavior between different flows.

The theoretical motivation and the assumption that the traces supplied hold all truth leads now directly to our methodology:

- We defined the Byte as the basic feature unit. It reflects the assumption that most of binary protocols are organized in byte order. Each byte field in a flow of a certain protocol shows a certain change in value, represented by the Byte statistic in Figure 2.
- The distributions of the same field *i* over different flows are compared in Figure 3. For the comparison, the variance of the distributions $\sigma^2_{i,j}$ is considered denoting the degree of dispersion of byte field *i* in flow *j*. The assumption behind this choice is related to the low degree of variability we are interested in.

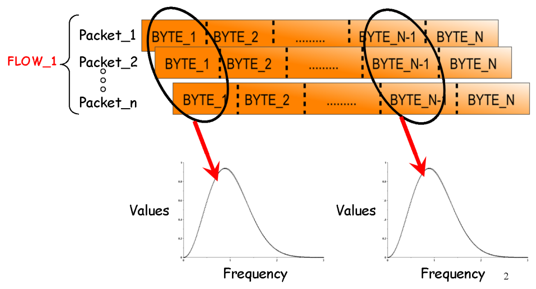

Fig. 2. Relevant Byte fields in different packets show characteristic Byte statistics

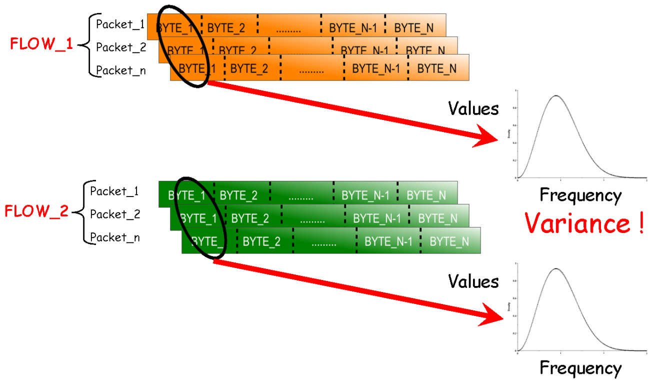

Fig. 3. Relevant Byte fields of different flows of same protocol show characteristic Variance of their Flow Statistic, Variance of Variance of the Byte field statistics

- For each byte field *i*, we compute for the entire set of flows *1..n* the distribution of the Variance. The shape of the distribution gives an indication of the variability of the field between several flows.
- We consider the variance of the previously computed distribution $\sigma^2\{\sigma^2_{i,1}, \sigma^2_{i,2} ... \sigma^2_{i,n}\}$ for each byte field *i* considering all flows *1..n*. Therefore the methodology is defined as "Variance of the Distribution of the Variances" (VDV).
- The most relevant field has the lowest value of the Variance. This conclusion denotes that relevant fields always show similar behavior between different flows, and therefore lowest variability. If more fields show a value of variance near to 0, they are both considered as relevant fields in the protocol.

At the end of this step the BFS has identified the most relevant field in a protocol message discarding all irrelevant attributes. Now the final protocol state machine has to be constructed by the SMB considering the most relevant field.

D. State Machine Builder

This module defines the notion of state and the proper transitions which interconnect every state. Both of these steps are performed by the "State Splitting algorithm" which is the core of the SMB and the novelty in our work. The algorithm II.1 can be divided into three different main steps which will be described below:

Algorithm II.1: STATE SPLITTING($InputTrace$)

```
comment: Reconstruct Protocol State Machine
CONSTRUCT_INITIAL_STATE_MACHINE($i$)
  for each $i \in InputTrace$
    do CREATE_INITIAL_STATES($i$)
       CONNECT_STATES($StateMachine, i$)

SPLITTING($InputTrace, StateMachine$)
  COMPUTE_SEQUENCE($InputTrace, Window$)
  COMPUTE_SEQUENCE($StateMachine, Window$)
  SPLIT_STATE($StateMachine, InvalidSequence$)

PRUNING($StateMachine, InputTrace$)
  for each $i \in InputTrace$
    do PARSE($StateMachine, i$)
       ELIMINATE_TRANSITION($StateMachine, i$)
```

1) Initial Construction of the State Machine: The first step of the algorithm constructs an initial protocol state machine where the states are identified as follows:

- START and END states are introduced defining a reference for the state model.
- Binary numbers are assigned for each state with a different value of the selected field. This step is motivated by the fact that different values of the relevant field represent different logic parts of the protocol.
- Different states are created considering the opposite side of a conversation, i.e if a message is sent or received. Thus the value of the relevant field in a message sent leads to a different state than the state generated by a value in a response message.

In the case of the TCP handshake, three different states are created, SENT SYN, RECEIVED SYN ACK, SENT ACK. Then they are interconnected according to the order appearing in the input traces. Thus an edge is created from the first state to the second and from the second to the third and so on.

2) Splitting: The splitting process aims at the reduction to minimum amount of possible states allowing all correct transitions. The minimum amount of states consists of only connected states, not equal and not being a subset of other states already present. The splitting step distinguishes between unequal states represented by the same feature value by additional inspection of the always present previous and following states using a time window. Assuming a request/response behavior in the protocol, and therefore considering two messages, to reveal a new state we only need an additional message. For instance in the case of TCP protocol we may distinguish between an ACK sent in the handshake phase after a SYN from an ACK sent in the closing connection after a FIN by looking one step ahead in time. In the first case we expect the starting of the connection, therefore a push or an ack message, in the latter the ending of the conversation and no more messages. This observation motivates the choice of a sliding time window of three messages for the derivation of the correct states. The necessary steps for the splitting outlined in Figure 4 are the following:

1) From the initial input traces, every sequence of states with a window of three is computed. For instance

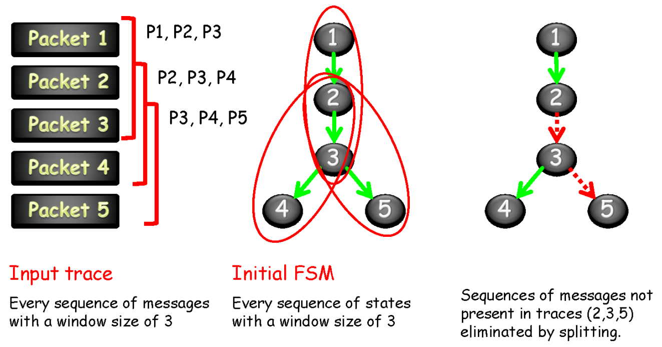
Packet 1
Packet 2
Packet 3
Packet 4
Packet 5
P1, P2, P3
P2, P3, P4
P3, P4, P5
1
2
3
4
5
Input trace
Every sequence of messages with a window size of 3
Initial FSM
Every sequence of states with a window size of 3
Sequences of messages not present in traces (2,3,5) eliminated by splitting.

Fig. 4. State Machine transition with time window=3; Comparison with input trace to identify invalid transitions

SENT SYN, RECEIVED SYN ACK, SENT ACK is an example of sequence with window three.

2) From the state machine created in the Initial Construction step every sequence of states with a window of three is computed.
3) Both sequences are compared and states with sequences not present in the input trace are split. From the original state with invalid transitions, several states with only correct ones are created, as sketched in Figure 5.

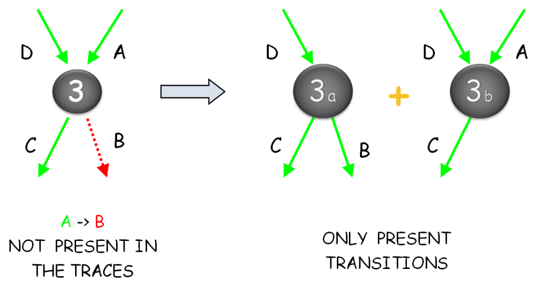

Fig. 5. State Splitting: Elimination of invalid transitions

3) Pruning: After the State Splitting Phase the state machine often comprises unnecessary edges. The reason lies in the nature of the splitting process itself. For each split state all the edges with no conflict are duplicated even if they may be originally related to only one particular state. In order to obtain the irreducible representation of the final protocol again the input traces have to be used in the same way as a regular expression is parsed by a finite state automaton. All non existing transitions in the parsing process are deleted because being redundant. The principal steps of the algorithm are the following:

- Parse traces with state machine.
- Keep transition visited at least once.

III. Results

A. Identification of relevant fields in protocols

The feature selection methodology based on the statistical analysis previously presented has been tested with several protocols: the Address Resolution Protocol (ARP), the Dynamic Host Configuration Protocol (DHCP), the Transmission Control Protocol (TCP) and the KAD protocol. We considered protocols ranging in different levels of the OSI model, from the second up to application level, with the final goal of proving the generality of our approach.

a) ***ARP****:* According to the message format specification of the ARP protocol, the "Operation Code" specifies the operation the sender is performing: "1" for request, "2" for reply. Therefore it expresses the logic of the protocol and it is eligible as relevant field. All the other fields of the ARP protocols are attribute parameters depending on the specific request and response. The result obtained with our BFS is shown in Figure 6.

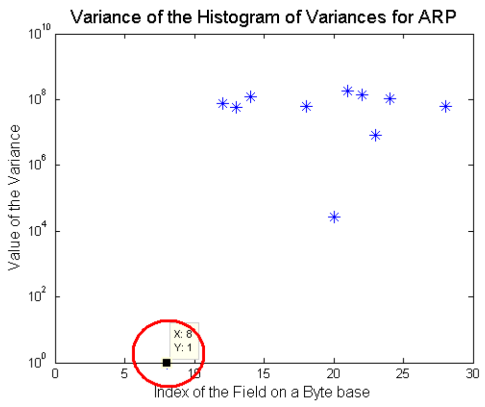

Fig. 6. "Operation Code" Field (red circle) in ARP protocol selected according to the Binary Feature Selector

The x-axis value represents the index of each byte in the protocol message, while the values on the y-axis represent the Variance with a logarithmic scale shifted by one in order to avoid negative values. Fields with constant values during the entire flow are automatically discarded because state transition is defined by a change in value. The lowest value of variance is in correspondence with the byte eight, i.e. the last byte of the "Operation Code" field. Thus our feature selection approach is able to identify the field carrying the logic of the ARP protocol.

b) ***DHCP****:* The application of our BFS to the DHCP protocol outlines two different fields with the lowest value of Variance as shown in Figure 7. The first one is the "Operation Code" (OPC) which specifies if a message is a request or a response. The second field is the "DHCP Message Type option" specifying the actual type of DHCP message. This field distinguishes between different types of requests, such as "DHCP REQUEST" or "DHCP DISCOVER", and different types of responses such as "DHCP ACK" or "DHCP NACK". Both fields express the logic of the DHCP protocol. The first captures the request/response behavior of the protocol, while the second specifies the type of request and response.

c) ***TCP****:* Same approach for the TCP protocol. The Variance for each field of the TCP header is computed and compared to the other fields in the TCP header. The Flags (byte 14) have the lowest value of Variance, as outlined in

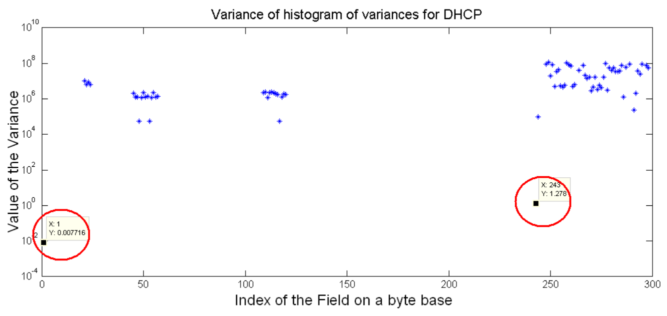

Fig. 7. “Operation Code” and “Message Type Option” are selected as relevant features in the DHCP protocol. (s. red circles)

Figure 8. This result is expected because TCP flags allow the establishment and management of the TCP connection thus representing the logic of the protocol.

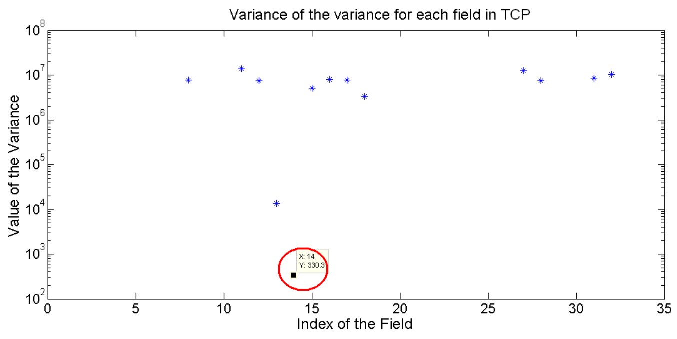

Fig. 8. TCP Flags selected by the Binary Features Selector

*d) **KAD**:* The application level protocol selected as a test case is the Kademlia protocol implemented in the KAD network, a peer to peer (P2P) overlay network based on a Distributed Hash Table algorithm (DHT). The choice of this protocol allows us to test our approach on a complex P2P protocol with a binary message format. Because of the unavailability of the RFC, we use as reference the work of [9]. The packet format is defined as follows:

- **Protocol Type** (one Byte): It specifies the type of protocol message. It states that a packet is part of the KAD protocol or of other protocols such as Edonkey [10].
- **Operation Code** (one Byte): It specifies the type of operation performed by the KAD protocol such as HELLO REQ or HELLO RES being used for the discovery of new peers in the network. It is representative for the logic of the protocol because it specifies its different actions.
- **Payload** (variable size): It contains the payload according to the command being specified in the “Operation Code” field.

Our Binary Features Selector is applied to diverse traces of the KAD protocol with around 1000 UDP packets. Because of the variable size of the payload in the various types of messages, packets are truncated to the size of the minimum one. As outlined in Figure 9, the field with the lowest

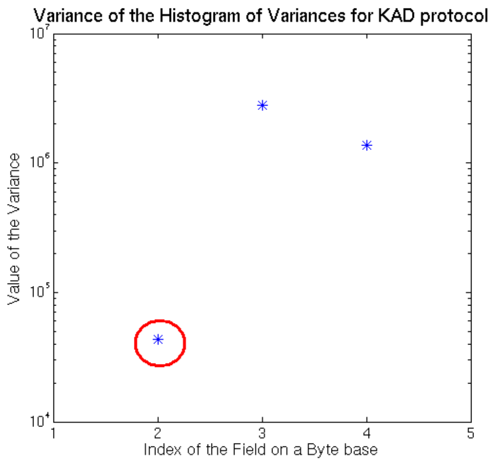

Fig. 9. “Operation Code” (red circle) is selected as the relevant feature in the KAD protocol message

variance is the “Operation Code”, the second byte position, which represents the actions performed by the protocol and therefore its logic.

B. Protocol State Machine Derivation

The derived protocol state machines presented in this section refer to the protocols previously examined with the BFS and in particular to TCP and KAD. The TCP protocol is analyzed due to its high relevance, various implementations and very good documentation. Nevertheless, we also focused on the KAD protocol because it is commonly used in P2P networks, rather complex, with binary format and poorly documented, thus being an interesting challenge.

***TCP protocol state machine**:* After identifying the TCP flags as the most relevant field, the SMB module is applied to tcpdumps representing live sessions with real users in Swisscom’s operational Testnetwork. We captured around 100 different TCP flows carrying different application level protocols such as FTP, HTTP/HTTPS and SSH. Due to the size of the state models the graph is divided into a main (Figure 10) and a sub graph s (Figure 11). Figure 10 is divided into different groups labeled with letters for a better visualization of the results.

We refer to the TCP connection state diagram as presented in the RFC 793 for a comparison with the results obtained. Group A in the proposed state machine represents the TCP handshake observed from the point of view of the initiator of the connection. The training traces also include a case of “Connection Reset” by the other peer.

Group B outlines the “Responder Sequence” in the TCP handshake, i.e. from the point of view of the peer receiving a request of a new connection. In our training traces an example of simultaneous opening of the connection is not available and thus not present in the state machine.

Group D represents the close of the connection initiated by the sender with the states SENT FIN ACK, RECEIVED FIN ACK, SENT ACK. After the SENT ACK, the connection ends reaching directly the end state.

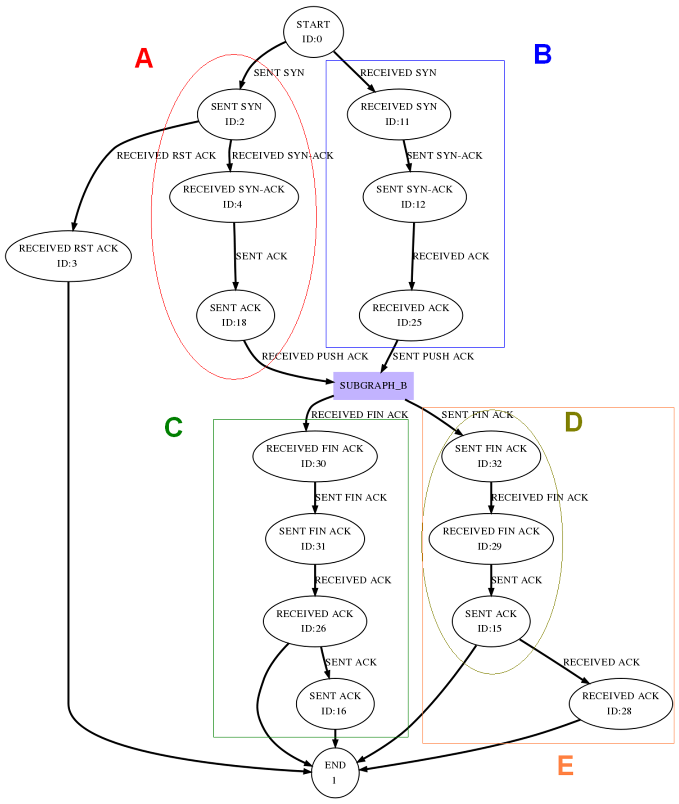

Fig. 10. Derived TCP state machine

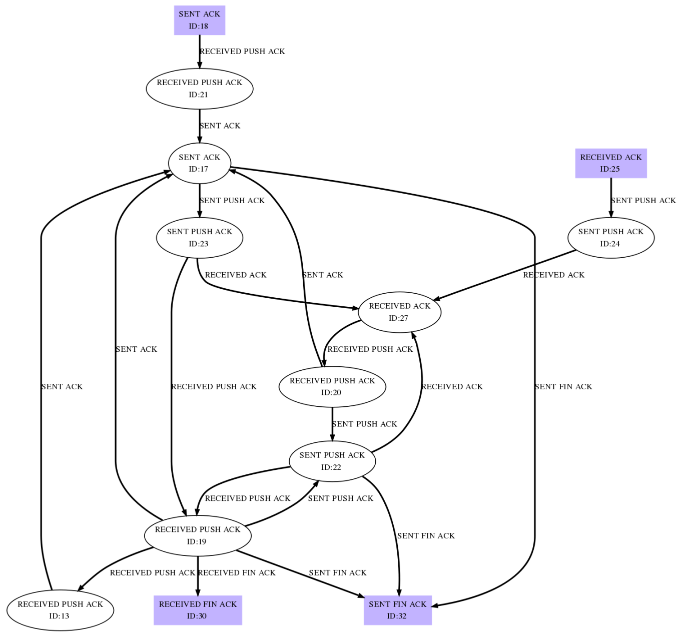

Fig. 11. Derived TCP state machine: subgraph B

Group E represents instead a simultaneous close where the ACK in the RECEIVED FIN ACK is not acknowledging the previous FIN. Thus the sequence of states SENT FIN ACK and RECEIVED FIN ACK generate a simultaneous close. This simultaneous close generates an additional ACK message necessary for acknowledging the first FIN. The state machine inferred realizes the simultaneous close by adding an additional RECEIVED ACK state before the END state. Thus both simultaneous close and a close initiated by only one side of the connection are correctly handled.

The group C is the "Responder Sequence" of a closing connection, as seen from the other side of the connection. It is the specular case of the previous two mentioned groups.

Sub Graph B in Figure 11 represents the established state of the TCP connection in which both error control and application specific logic is interconnected. These two mechanisms are respectively represented by the ACK Flag for error control and by the PUSH Flag controlled by the application layer. The interconnection of these two mechanisms generates several different sequences of messages where error control logic and the logic of different applications are combined.

Comparing the derived state machine with the one presented in the RFC proves the correctness of the result. Due to missing training data only the simultaneous open of the connection is missing, then the state model would be complete. Moreover only a "Connection Reset" state after a SYN in the TCP handshake is present. Correct appropriate transitions to the connection reset state from other states are part of the TCP state machine. These missing transitions depend only on missing training data.

KAD protocol state machine: We will present the KAD protocol state machine derived by the SMB module comparing it to the diagrams presented in the work of [9]. In particular we decided to focus on the "Initialization Process" which is the initial step of the entire set of operations performed in the lifetime of a KAD client. These Initialization processes seem to be very important for traffic and protocol identification and the complex behavior with many different operations makes it an interesting and challenging candidate to test our approach.

The initialization process is divided in the following steps:

- **Bootstrap Process**: In order to join the KAD network a client sends a BOOTSTRAP REQ to a known client asking for new peers in the network. The response is a BOOTSTRAP RES containing a list of several nodes in the network.
- **Initial Handshake** with a New Peer: In order to verify whether a peer is online, a REQ is sent to the new contact which will reply, if still online, with a RES message.
- **Firewall Check**: In order to verify whether it is behind a firewall or a NAT, the client sends a FIREWALLED REQ to a peer in the network. The peer will answer with a FIREWALLED RES trying to connect to the checking client. If the connection is successful it will send a FIREWALLED ACK.
- **Finding a Buddy**: A client in a firewalled state tries to connect to a non firewalled client, a buddy, for receiving incoming messages. The client sends a FINDBUDDY REQ to a peer in the network which will answer with

a FINDBUDDY RES if it is not firewalled.

Keeping in mind these main steps, we present in Figure 12 the derived KAD protocol state machine. In order to observe the initialization process a Tcpdump capture of the initial 10 minutes of the KAD activity with around 1000 UDP packets was captured.

The state machine in Figure 12 is graphically divided into different parts for a precise comparison to the previous presented logical steps. Group R outlines the initial handshake performed when contacting a new peer. Our client sends a REQ which is followed by a RES if the client is online. Starting from this initial handshake several phases follow such as searching for a buddy or starting a firewall check.

Group A represents the firewall check performed by the client. A FIREWALLED REQ is sent by our client while a FIREWALLED RES is returned by the responding client. In this case a FIREWALLED ACK RES is not present because our client is behind a NAT. During a firewall check normal communication between peers is performed. Thus in the group A a HELLO REQ and HELLO RESP messages are exchanged to verify the online status of a peer. After the firewall check the client may start a new firewall check (group A), find a buddy peer (group C), receive a firewall check (group B) or end the conversation (END state).

Group B represents a request for a firewall check performed by another client in the network. It is the specular case of a request performed by our client as explained before for group A. In this case however there is a successful FIREWALLED ACK RES sent to the opposite client meaning that the demanding client is not firewalled.

Group C represents the search for a buddy peer being capable of forwarding incoming connections. The message sent by our client is a FINDBUDDY REQ. The responding client may answer positively with a FINDBUDDY RES if its state is not firewalled. It does not answer ending the conversation if its state is firewalled.

Groups D and D' model the exchange of packets between peers for checking whether they are still online and available for communication. In group D our client sends a HELLO REQ to a new peer which in turn answers with a HELLO RES message if it is online. Other HELLO REQ messages are sent if the peer is not answering. A firewall check procedure may be started if the peer is online.

Compared to the theoretical description previously presented and the work of [9], our state machine models perfectly the exchange of packets between our client and the other peers in the network. Only the messages for the bootstrapping phase are not present because our particular client uses an external file for bootstrapping. Moreover the protocol state machine outlines additional correct transitions and sequences of states being not present in the diagrams of [9]. For instance, in the state machine it is clearly visible that a FIREWALL REQ is always followed by a HELLO REQ. Instead in the diagrams of [9] such a state is postponed after the FIREWALL RES. Additionally our state machine outlines the link between a firewall check and the search for a buddy connecting the two subgroups. Therefore our approach derives the correct state machine and enriches the model of the protocol as presented in the previous work.

IV. Conclusions

Our work demonstrated a valid approach for reconstructing the state machine of an unknown protocol only from its network traces. From raw network traces first we derive the protocol's features being relevant for capturing the logic of the protocol. Therefore we presented a Binary Features Selector module based on a statistical analysis of each field among all flows. We obtained that the most relevant field in a protocol has the lowest value of variance of the distribution of the variance computed for each flow. The approach has been evaluated with several protocols ranging in different layers of the OSI model: ARP, DHCP, TCP and KAD. For each of these protocols the relevant fields are automatically derived.

As a second step we started from the identified relevant features to derive the state machine of the protocol. We presented our State Machine Builder module based on the "State Splitting" algorithm for the correct reconstruction of the state machine. Focusing on the evolution in time of the sequence of states, the algorithm splits each state creating all correct transitions in the state machine. The result of the algorithm is a graphical representation of the state machine of the protocol immediately outlining its logic.

The approach is empirically evaluated with two different protocols: the Transmission Control Protocol (TCP) and the Kademlia peer to peer (P2P) protocol. Compared to the RFCs and our knowledge about the protocol, both the resulting state machines have proven the correctness of the methodology. In particular the reverse engineering of the Kademlia protocol outlines that our approach is successfully applied to complex problems. Kademlia is a binary protocol with an emerging complexity and poor documentation. Our approach without previous knowledge of the protocol is able to derive the state machine of the initialization process. Moreover it identifies correct transitions not documented in previous works, denoting the future potential of our work.

The most promising future work is related to the protocol state machine derivation from encrypted traffic. In this context Deep Packet Inspection is not feasible due to the encryption of the channel. However previous research in the context of encrypted traffic mining [11] has demonstrated a way to derive different types of commands in encrypted sessions. Starting with features selected by encrypted mining techniques, our State Machine Builder module may be applied to derive the state machine of even encrypted protocols. If the protocol transfers information of human behavior onto the IP layer, such as Telnet, SSH, FTP detailed robust state models, or signatures of normal or thus abnormal behavior could be created. While in state-full protocols first the human part has to be extracted, in non state-full ones such as HTTP/HTTPS every message directly codes into a human state. Hence, HTTP/HTTPS being generally used might be

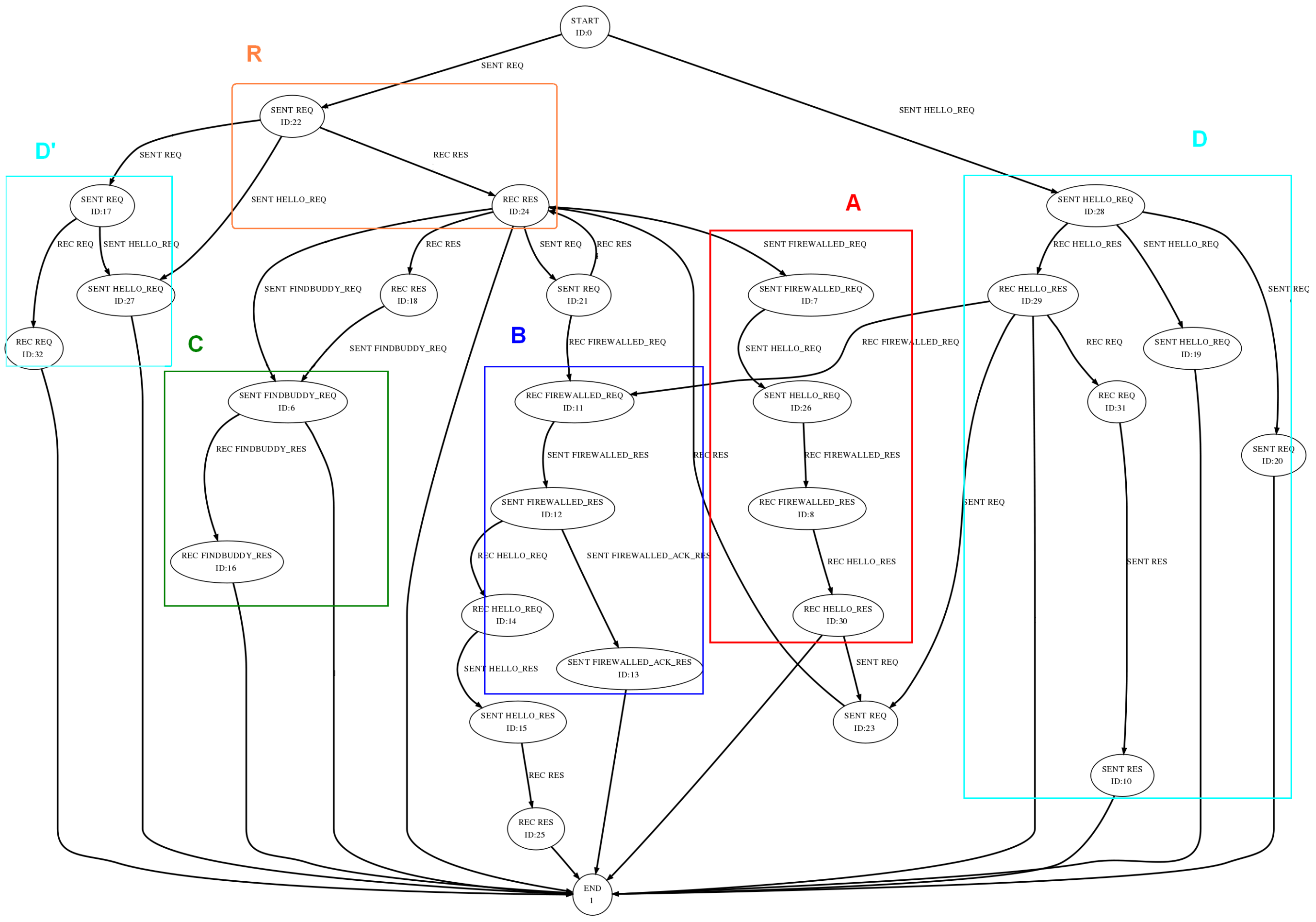

Fig. 12. State machine of the initialization process in KAD

a good protocol to produce human behavior models in order to detect abnormal behavior.

References

[1] A. Trifilò. *Traffic to Protocol Reverse Engineering*, Master Thesis Report, Eurecom (France), Swisscom Innovation (Switzerland), Politecnico di Torino (Italy), September 2008.

[2] C. Leita, K. Mermoud, and M. Dacier. *ScriptGen: An Automated Script Generation Tool for Honeyd*. In 21st Annual Computer Security Applications Conference (ACSAC), 2005

[3] C. Leita, M. Dacier, and F. Massicotte. *Automatic Handling of Protocol Dependencies and Reaction to 0-Day Attacks with ScriptGen-based Honeypots*. In Symposium on Recent Advances in Intrusion Detection (RAID), 2006.

[4] W. Cui, J. Kannan, and H. Wang. *Discoverer: Automatic Protocol Reverse Engineering from Network Traces*. In 16th Usenix Security Symposium, 2007.

[5] W. Cui, V. Paxson, N. Weaver, and R. Katz. *Protocol-Independent Adaptive Replay of Application Dialog*. In 13th Symposium on Network and Distributed System Security (NDSS), 2006.

[6] G. Wondracek, P. M. Comparetti, C. Kruegel, and E. Kirda. *Automatic Network Protocol Analysis*. Network and Distributed System Security Symposium (NDSS), Internet Society. USA, February 2008.

[7] J. Newsome, D. Brumley, J. Franklin, and D. Song. *Replayer: Automatic Protocol Replay by Binary Analysis*. In 13th ACM Conference on Computer and Communications Security (CCS), 2006

[8] X. Haijun, P. Fang, W. Ling, L. Hongwei. *Ad hoc-based feature selection and support vector machine classifier for intrusion detection*. In Grey Systems and Intelligent Services (GSIS), 2007.

[9] R. Brunner. *A performance evaluation of the kad-protocol* , Masters thesis. Eurecom (France), Mannheim University (Germany). November 2006.

[10] Wikipedia. *eDonkey Network*. http://en.wikipedia.org/wiki/EDonkey_network. August, 2008.

[11] J. R. Borque. *Encrypted Traffic Mining: SSH Command Guessing*, Master Thesis Report, Eurecom (France), Swisscom Innovation (Switzerland), January 2007.