Towards model-based anomaly detection in network communication protocols

Jędrzej Bieniasz, Piotr Sapiecha, Miłosz Smolarczyk, Krzysztof Szczypiorski
Cryptomage S.A.
Wrocław, Polska
e-mail: {jedrzej.bieniasz, piotr.sapiecha, milosz.smolarczyk, krzysztof.szczypiorski}@cryptomage.io

***Abstract*—Over the last few years many techniques have been applied to find and mitigate vulnerabilities, misuses, cyber-attacks and other cyber-security flaws. One of the approaches, which we consider in this paper, is a model-based technique applied to network communication protocols. This idea is not brand new, and model-based techniques have been successfully used to verify and validate the standard models of communication protocols. However, the implementation of network protocols varies from one system to another, and in many cases they miss standards or recommendations. Attackers know these flaws very often and try to use them before everybody else finds them, what can be called "zero-day exploit of communication protocol." To address this issue, a combination of the best features of model-based and anomaly detection techniques could be applied. Treating discovered anomalies as a signature of a cyber-attack or any other malicious activity and focusing on the investigation of them could significantly increase the success rate of the defense against them. In this paper we considered some significant inputs from the research community to model-based anomaly detection in network communication protocols. Then we prepared a synthetic brief of the theories and methods for modelling network protocols as state-machines. Next we examined the application of it in a cyber-security area. Finally we proposed some key directions that actual research should follow to bring some breakthrough results as soon as possible.**

Keywords: Network Protocols, Protocol Model Inference, Model-based Verification and Validation, Model-based Testing, Anomaly Detection, Protocol Reverse Engineering

I. INTRODUCTION

There are many proposals for identifying malicious network activities, such as pattern recognition, data mining, statistics, machine learning and many other techniques, the application of Model-driven Engineering (MDE) takes the majority of the attention of researchers. MDE is widely used in the software development process, where representing computation systems using models is a way of controlling their complexity and offers an efficient verification of behavior. At its core MDE is based on two methods:

1) *Model checking* – verifying the state space of a given state transition model.

2) *Model inference* – generating a model from the interaction with the system's components by applying automata learning algorithms.

This approach has a built-in and commonly known disadvantage – the difficulty of controlling state-space size. This difficulty is reflected in the advance of applied research for both methods. Model checking is widely used for industrial purposes, whereas model inference research is not as advanced, but the progress is constantly increasing.

One of the areas where model checking and model inference techniques find an increasing interest is cyber-security. From the cyber-security view, MDE could be an ideal tool to observe the behavior of IT systems and to detect anomalies – all events beyond accepted ones. Two main factors determine its usefulness:

1) Protocol implementations could have other flaws, which are accidentally added during development or because the developers do not have the required skills.

2) There is no way to prepare for all types of implementations, so learning algorithms, preferably automatic ones, need to be applied in a new network environment to infer a correct template of behavior.

The combination of MDE and anomaly detection fully addresses these concerns by:

1) Applying automatic algorithms for network protocol model inference in a particular IT system that consists of network appliances and hosts.

2) Observing the behavior of such a system and examining it against an established model of the correct behavior (anomaly detection).

All mismatched communication events could determine a real exploit of this particular flaw of the protocol. Many times the attackers know them before the developers do ("zero-day"), but novel techniques like the presented one here could not only detect them, but also actively combat them. The latest research shows some significant results, which brings us closer to widely applicable solutions. In this paper we focused on the current knowledge and results for automatic model inference. We also identified some key propositions for research directions to follow in the next few years.

The remainder of this paper is organized as follows: Section II briefly presents the state of current knowledge on model inference, automata learning and modelling protocols. In Section III, two possible applications of model inferring and modelling protocols are presented: an infiltrating botnet command and control (C&C) protocol and an Intrusion Detection System (IDS) for VoIP (*Voice over IP*). The key directions and conclusions for actual research in model-based anomaly detection are provided in Section IV.

II. MODELLING PROTOCOLS

A. Introduction

Network protocols are modelled in many ways depending on the need of the modeler: time diagrams, diagrams of messages and diagrams of entities, etc. The standard is a model of a state-machine diagram that consists of a finite number of protocol states with or without connections between them. The existence of a connection between the two selected states means that there is a real event causing the transition from one to another. Such state-machines of protocols can be found in standards [1] or the literature [2], but as indicated in Section I, the implementation of network protocols varies from one system to another and many times they miss the standard. Thus, these standard or well-known state-machine models can be only treated as a baseline or starting point to more advance research. Model inference techniques are adequate solutions in this case. Their aim is to learn (passively or actively) about a state-machine describing a system under examination. Protocols can be an example of such a system and they can be learnt through various proposed techniques, In fact, the model will only be an approximation of a real system.

There are two basic finite state-machine types: Moore [3] and Mealy [4] machines. They differ in the way that the states are distinguished – the former's states are accepting or not, while the latter has no accepting states and the sequence of outputs from the sequence of transitions is the key. The Mealy machine model is a more appropriate model for protocols as they are reactive systems and transitions are the core. Model Protocol Inference is in fact a problem of grammatical inference, as inferring the state-machine is equivalent to the problem of learning a regular language. There are two main types of inferring procedures for state-machines:

1) Active (also on-line) – is based on the active probing of the inferring system that is treated as a black-box, called a System Under Test (SUT). Simply describing the idea, the system is queried with a prepared set of inputs, next the outputs are observed and then based on the set of inputs and outputs the model is deducted.

2) Passive (also off-line) – is based on analyzing system traces collected over time and deducting the model from them.

B. Learning state-machines

1) Active learning

The two main papers about active learning protocol state-machines we want to consider are [4] and [5]. In these papers the active method for the inference model of the protocols is presented and applied to infer TCP (*Transmission Control Protocol*) and SIP (*Session Initiation Protocol*). The core of the method is the *L** algorithm of Angluin introduced in [7] that sets the theory and the practical realization of the active learning for the Mealy state-machine model. Next, it was adapted by Niese [8] and used by many authors, such as in [4] and [5]. Figure 1 presents a diagram of the active method.

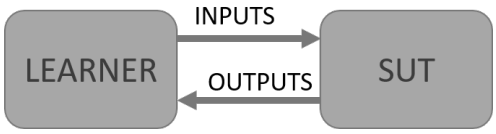

Figure 1 Active learning of SUT behavior [5].

We assume that the system to model (System Under Test (SUT)) can be modelled by the Mealy machine $<I, O, Q, q0, \rightarrow>$. The system maintains the current state, which at the beginning is q_0. The system accepts inputs from *I* and special reset signalization. The learner sends input queries to the SUT and observes the outputs. When the system receives a query, the transition q → q' is triggered. The current state q changes to state q' and the output o is generated. After each observation the system is reset. Based on the outputs a hypothesis H is built and then tested against the system. In a number of test cases, the conformation of the system and the hypothesis are checked. If all tests pass, the hypothesis is accepted; otherwise, a new counterexample is found and the hypothesis must be refined. The procedure is repeated until all tests pass without generating a counterexample.

Typically, network protocols are systems with large alphabets, e.g., inputs and outputs have parameters. This indicates the need to find solutions that reduce the problem and increase the effectiveness of inferring algorithms for network protocols. Aarts et al. in [5] proposed a solution. They introduced a third entity – the mapper – to be placed between the learner and the SUT, which is shown in figure 2.

Figure 2 Active learning of SUT behavior with mapper.

The learner operates on abstract inputs and outputs. Abstract inputs and outputs are established from diving the concrete ones into equivalence classes in a history-dependent way. The mapper transforms abstract inputs into concrete ones accepted by the SUT. In the opposite direction, it transforms the concrete outputs into abstract ones for the learner. Aarts et al. [5] proved that the resulting hypothesis combined with the mapper gives an over-approximation of the SUT.

The authors of [4] and [5] applied the presented ideas to experiments to infer the TCP and SIP protocols. As a learner they used the tool LearnLib [9], which effectively implements the *L** algorithm by Angluin [7]. As the SUT, the ns-2 simulator was used.

In a paper [5], the experiments were focused on inferring the SIP and TCP protocols. For SIP, an abstract model with nine states and 63 transitions was found. The simplification had the same number of states but less transitions – 48. LearnLib needed about 1000 queries as the inputs. For TCP,

11 states and 187 transitions were deducted. The pruned model consisted of 10 states and 41 transitions.

The authors of [6] conducted research on different implementations of the TCP protocol in network stacks of Windows and Linux Ubuntu. The experimental setup was as shown in figure 3.

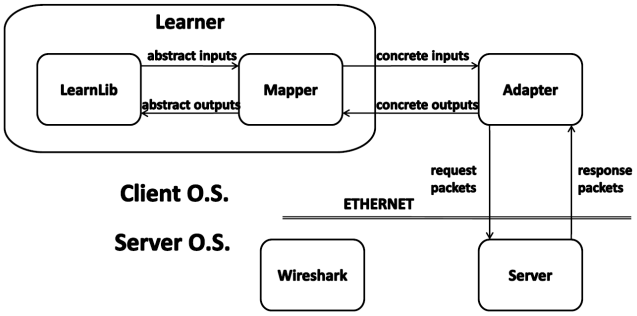

Figure 3 Overview of experimental setup from [6].

The server was a TCP server deployed on Windows 8 or Ubuntu 13.10. The learner acted as a TCP client, which sent messages to the server. The adapter was a Python application written using *Scapy* to craft requests and parse responses. The resulting models are shown in the original paper [6]. Some similarities and differences were found, and they are summarized in the table I.

TABLE I. COMPARISON OF INFERRED MODELS OF TCP [6].

Similarities	Differences
- Easy to identify a handshake and termination: S(v,v) → A(v,v) → AF(v,v). - Each input in the sequence has the same output generated.	- Verbosity of the listening state in Ubuntu. - RST-segments – Ubuntu has a 0 acknowledgment number whereas Windows takes the last acknowledgment number sent seulting in RST(las,las) outputs. - Flags found in response packets. - Slightly different transitions between states.

2) Passive learning

Passive learning is based on deducting models from historical traces of the system. For network protocols we consider network traces, e.g., in *tcpdump* or *Wireshark* formats. This type of learning process was used by the authors of [10]. They implemented *ReverX,* which infers the protocol specification from a network trace with samples of the protocol interactions. The tool was verified with publically accessible FTP (*File Transfer Protocol*) traces and examined against models known from standards.

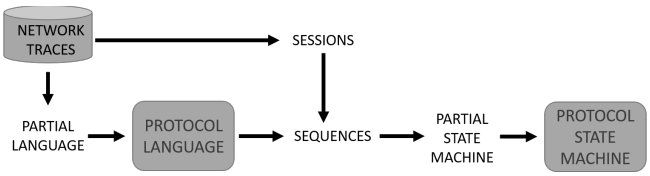

Figure 4 Passive learning example for ReverX tool [10].

To deduct models of network protocols passively, two models should be obtained:

1) A model of the language as network protocols can be seen as formal languages.

2) A model of the entire protocol for the relation of different types of messages.

The need for these two types of model inference in the passive learning is presented in figure 4, which is based on the scheme of *ReverX* [10].

a) Inferring the language

There are a few approaches to infer a language. In [10] the authors used the state-machine one, but other methods are known, e.g., in [11] the clustering, partitioning and characterization of the message types and fields was applied. Other methods can also be chosen, and it is an arbitrary decision by the researcher.

In [10], inferring the language consists of two steps: 1) construction of a state-machine, which only accepts messages from traces, and 2) generalization to accept different instances of the same type of messages. There should be some algorithms for constructing formal languages from the samples applied. The authors in [10] proposed their own function to infer the language protocol based on the construction of the automata of the partial language, but any other known algorithm for inferring this automata could be used.

This method could also be used in combination with active learning methods. In this system, the passive part would be an inferring language algorithm to prepare a set of inputs and outputs (the protocol alphabet). The active part would be exactly as that described in Section II.B.1 for the method to infer the protocol model by the active probing of the SUT. This idea is used by the *Netzob* tool [11].

b) Inferring protocol state-machine

The authors in [10] used the same approach for the protocol state-machine as for inferring the language. The process consisted of a few steps:

1) Extract the application sessions – the state-machine must be inferred for each application session, where one application session is a set of message sequences. The method to determine such a session should be adjusted to the characteristics of the operation of the network protocol.

2) Construction of the Partial State Machine – in [10] the authors used an analogous function to inferring the language, but any other known method could be applied here.

3) Reduction – the state-machines from step 2 can be minimized to a more general shape.

III. MODEL-BASED CYBER-SECURITY

The success of the field of theory for inferring models allows the development of a wide range of applications. In this paper we focus on cyber-security and the benefits of MDE for it. We present some key results on the application of a model-based approach to cyber-security for inferring network protocol models.

A. Botnet C&C protocol inference

Cho et al. [12] presented the method of inference and analysis of formal models of botnet command and control (C&C) protocols. Their contribution was to establish a novel approach to infer a complete state-machine in a realistic high-latency network environment. They used a Mealy machine modelling approach for protocol modelling. They used a highly effective procedure for minimizing the number of queries generated during the inference process. They proposed some further optimizations to the L^* algorithm that made the inferring procedure more effective. The design of the testbed should also be noticed. It could be considered a design proposition for this kind of inference system, which effectively infers models and provides empirical data proofs of applied optimizations. Experimental results showed that the approach is very attractive. The authors could infer C&C protocol models and discover some interesting things, such as the critical links and design flaws. They found for example:

1) What server is the key to the spam capabilities of the botnet.

2) A way to obtain spam templates from the C&C server easily.

3) How to detect background communication.

The main drawbacks of their method, as mentioned by the authors, are:

1) No hiding of the probing traffic and its high volume that could be detected by the botmaster.

2) Protocols triggered based on date/time cannot be inferred by this method.

3) The precision might not be enough for all potential applications.

4) Protocols with more complicated syntax (more expressive language) need other inferring methods for their language.

5) The reverse-engineering of the alphabet is done manually here, while it could be automatic, but it is an open issue.

Another insight into botnet infiltration was added by Caballero et al. [13]. They presented a problem of a botnet on a black-box set – where the C&C protocol language and protocol state-machine are unknown. They addressed the first problem – extracting the message formats from the given examples. They also based it on the MegaD botnet construction as in Cho et al. [12]. They introduced a buffer deconstruction technique that extracts the structure of the sending message by reconstructing the process of building the output buffer from other memory buffers. They proposed methods for inferring field semantics. They developed a tool Dispatcher that incorporates all the presented techniques. This was tested against known and unknown MegaD C&C protocol messages. The results showed that it enabled the infiltration of the botnet.

B. VoIP IDS

Sengar et al. [14] proposed a novel intrusion detection system based on inferring the network protocols behind IP telephony systems. Their system focused mainly on SIP-based and RTP-based model attacks: 1) for SIP: CANCEL DoS and BYE DoS, and 2) for RTP: media spamming and RTP packets flooding. They manually prepared patterns of such attacks as a specific sequence of states to occur. These signatures of the attacks were stored in a database of the VoIP IDS and used during operation. The VoIP IDS operated as follows: 1) the system tracked the progress of the SIP and RTP state-machines in parallel for every single VoIP, 2) the following sequences of changing states were observed and examined against the patterns and 3) if the misbehavior pattern was observed, the IDS raised an alert and notified the administrators.

The authors identified a problem of induced delay at around 100 ms, but it should be acceptable. Nevertheless, some solutions could be proposed to address this problem. The idea is worth developing as the results in [14] show 100% detection accuracy for the conducted attacks without any false positives. Furthermore, the system could be applied for unknown attacks, but in this case the performance of the classification depends on the quality of the inferred protocol models.

IV. CONCLUSIONS AND PROPOSALS

In this paper we evaluated the knowledge behind model-based cyber-security solutions. From this evaluation, the modelling of network protocols through the automatic inference of their state-machines looks to be a very promising solution for all types of cyber defense appliances, like anti-virus, anti-malware, intrusion detection and prevention systems, etc. We presented two successful deployments of these ideas: 1) compromising botnet C&C protocols and 2) intrusion detection system for VoIP applications. Based on the material, some key directions for the research community can be formulated and should be addressed in further work.

1) Prevent active learning against being detected – active learning is based on the active probing of the system of the attacker, so any techniques to hide this are needed, such as obfuscation and misleading, etc. Nevertheless, the case is not trivial and further research is needed.

2) Methods of inference – better methods for protocol languages and state-machines should be developed to address problems of expressive language in some of them and the wide range of protocol types. In addition, effective and automatic inference of the protocol language could eliminate the need for a protocol standard and any manual work related to this would be unnecessary.

3) Real time and automatic design – only fully automatic, timely and effective design could be considered as applicable to real applications. It should take into consideration the environment of operation, especially the speed of modern network flows, which should be treated properly according to the accuracy of the tool. There is a space for developing industry standard solutions.

4) Detection – the authors in [14] presented the signatures of the attack approaches, but techniques that do not require such preparations are welcome. Obviously, some manual work is needed, e.g., choosing the parameters to

observe, but we want to minimize this process as much as possible.

ACKNOWLEDGMENT

This scientific research work was co-financed by the European Union, project name: “The system for identification and monitoring of anomalies and risks in ICT networks.” The amount financed by the European Union was EUR 1,044,534.63. The investment outlay value for the entire project was EUR 1,407,526.46. The subsidy was allocated from the European Regional Development Fund, Operational Programme “Smart Growth,” sub-measure 1.1.1 “Industrial research and development work implemented by enterprises” (grant number: POIR.01.01.01-00-0554/15).

REFERENCES

[1] J. Rosenberg, H. Schulzrinne, and Ed R. Mahy, "An INVITE-Initiated Dialog Event Package for the Session Initiation Protocol (SIP)", RFC 4235, November 2005, http://www.rfc-editor.org/info/rfc4235.

[2] S. Seth and M. Ajaykumar Venkatesulu. “TCP/IP Architecture, Design and Implementation in Linux.” Wiley-IEEE Computer Society Pr, 2008.

[3] E. F. Moore "Gedanken-experiments on Sequential Machines". *Automata Studies, Annals of Math*. Studies (Princeton, N.J.: Princeton University Press) vol. 34 (1956) : pp. 129–153.

[4] George H. Mealy. ”A method for synthesizing sequential circuits.” *The Bell System technical journal*, vol. 34 (1955), pp. 1045–1079.

[5] F. Aarts, B. Jonsson, and J. Uijen. “Generating models of infinite-state communication protocols using regular inference with abstraction.” In Proceedings of the 22nd IFIP WG 6.1 international conference on Testing software and systems (ICTSS'10), Springer-Verlag, Berlin, Heidelberg, 2010. pp. 188-204.

[6] P. Fiterau-Brostean, R. Janssen and F.W. Vaandrager. “Learning Fragments of the TCP Network Protocol”. Proceedings 19th International Workshop on Formal Methods for Industrial Critical Systems, September 11-12, 2014, Florence, Italy. LNCS 8718, pp. 78--93, Springer-Verlag, 2014.

[7] D. Angluin. “Learning regular sets from queries and counterexamples.” Inf. Comput. 75, 2 (November 1987), 87-106

[8] O. Niese. “An integrated approach to testing complex systems”. Technical report, Dortmund University, 2003. Doctoral thesis.

[9] F. Howar, M. Isberner, M. Merten, and B. Steffen. “LearnLib tutorial: from finite automata to register interface programs.” In Proceedings of the 5th international conference on Leveraging Applications of Formal Methods, Verification and Validation: technologies for mastering change - Volume Part I (ISoLA'12), Vol. Part I. Springer-Verlag, Berlin, Heidelberg, 2012. pp. 587-590.

[10] J. Antunes, N. Neves and P. Verissimo, "Reverse Engineering of Protocols from Network Traces," 2011 18th Working Conference on Reverse Engineering, Limerick, 2011, pp. 169-178.

[11] G. Bossert et. al. “NetZob: tool for reverse engineering communication protocols”, https://www.netzob.org/ [Access: 15.07.2016]

[12] C. Y. Cho, D. Babic, E. C. R. Shin, and D. Song. “Inference and analysis of formal models of botnet command and control protocols.” In Proceedings of the 17th ACM conference on Computer and communications security (CCS '10). ACM, New York, NY, USA, 2010. pp. 426-439

[13] J. Caballero, P. Poosankam, C. Kreibich, and D. Song. “Dispatcher: enabling active botnet infiltration using automatic protocol reverse-engineering.” In Proceedings of the 16th ACM conference on Computer and communications security (CCS '09). ACM, New York, NY, USA, 2009. pp. 621-634

[14] H. Sengar, D. Wijesekera, Haining Wang and S. Jajodia, "VoIP Intrusion Detection Through Interacting Protocol State Machines," International Conference on Dependable Systems and Networks (DSN'06), Philadelphia, PA, 2006, pp. 393-402.