Active Automata Learning of an IPsec IKEv1 Server using AALpy

Benjamin Wunderling[1][0000−1111−2222−3333]

TU Graz IST, Graz 8010, Austria
benjamin.wunderling@student.tugraz.at

Abstract. Virtual Private Network (VPN) protocols are widely used to create a secure mode of communication between several parties over an insecure channel. A common use-case for VPNs is to secure access to company networks. Therefore errors in VPN software are often severe. IPsec is a VPN protocol that uses the Internet Key Exchange protocol (IKE). IKE has two versions, IKEv1 and the newer IKEv2. While several papers have investigated IPsec-IKEv2 in the context of Automata learning, no such work has been performed for IPsec-IKEv1. This short paper describes the IPsec-IKEv1 protocol and show the steps taken to learn the state-machine of an IPsec server. We present a learned model and discuss its potential applications for model-based fuzzing and fingerprinting of IPsec implementations.

Keywords: IPsec · Automata Learning · AALpy.

1 Introduction

VPNs are used to allow secure communication over an insecure channel. The importance of VPN software has increased dramatically since the beginning of the COVID-19 pandemic due to the influx of people working from home [1]. This makes finding vulnerabilities in VPN software more critical than ever. IPsec is a VPN protocol and most commonly uses the IKE protocol to share authenticated keying material between involved parties. Therefore, IKE and IPsec are sometimes used interchangeably. We will stick to the official nomenclature of using IPsec for the full protocol and IKE for the key exchange only. IKE has two versions, IKEv1 IKEv2, with IKEv2 being the newer and recommended version [4]. However, despite IKEv2 supposedly replacing its predecessor, IKEv1 is still in widespread use today. This is reflected by the company AVM to this day only offering IKEv1 support for their popular Fritzbox routers [9].

State models of protocol implementations are useful tools in testing. They can for example be used to detect software versions [14], or generate test cases automatically [15]. One method of generating such models is to use Active Automata Learning. A notable example of an Active Automata Learning algorithm is the L* algorithm by Angluin [2]. In L*, a teacher queries the System under Learning (SUL) and through its responses can construct an automaton describing the behavior of the SUL. This automaton can then be compared with the

SUL, adapting it if they show different behaviors. Several papers have investigated IPsec-IKEv2 using Automata Learning, however so far none have looked at IKEv1.

We show the process of learning a state model from an example IPsec-IKEv1 server. We use the Active Automata Learning framework AALpy [12] for L* Automata Learning, with a custom Python interface between AALpy and the IPsec server.

In this short paper we first go over preliminary information on VPNs and Automata Learning in chapter 2. In 3 we discuss other related work. In chapter 4 we briefly introduce AALpy and our learning setup. Then we will present our custom interface between AALpy an the IPsec server, discussing design choices and implementation difficulties. Finally we present the learned model and discuss its potential applications and further work in chapters 5 and 6.

2 Preliminaries

2.1 Automata Learning

Automata Learning refers to methods of learning a state model, or automata, of a system through an algorithm or process. We differentiate between Active and Passive Automata Learning. In Passive Automata Learning (PAL), models are learned based on collected logs or other information describing the behavior of the system under learning (SUL). In contrast, in Active Automata Learning (AAL) the SUL is queried directly. In this short paper, we will focus on AAL and will moving on refer to it as Automata Learning or AAL interchangeably.

AAL began in 1987 with a paper by Dana Angluin, titled "Learning regular sets from queries and counterexamples"[3]. In this seminal paper, Dana introduced the L* algorithm which is still used for learning deterministic automata. L* works using a Minimally Adequate Teacher (MAT) model in which a learner queries a teacher about a SUL. The teacher must be able to answer equivalence and membership queries posed by the learner regarding the SUL. Equivalence queries are used to check if a learned model accurately matches the SUL and membership queries are used to check how the SUL reacts to a particular input. The learner, using the responses to its queries, then updates its model of the SUL. Learned models are commonly represented as Mealy machines, finite-state machines with outputs depending on the current state as well as external inputs.

2.2 IPsec

IPsec or IP Security, is a VPN layer 3 protocol used to securely communicate over an insecure channel. It is based on three sub-protocols, IKE, the Authentication Header protocol (AH) and the Encapsulating Security Payload protocol (ESP). IKE is mainly used to handle authentication and to securely exchange and manage keys. Following a successful IKE round, either AH or ESP is used to send packets securely between parties. The main difference between AH and ESP is that AH only ensures the integrity and authenticity of messages while ESP also ensures the confidentiality through encryption. Compared to other protocols, IPsec offers a high degree of customizability, allowing it to be fitted for many use-cases. However, in a cryptographic evaluation of the protocol, Niels Ferguson and Bruce Schneier criticized that customizability stating that the biggest weakness of IPsec is its complexity [6]. To address its main criticism IPsec-IKEv2 was introduced in RFC 7296 to replace IKEv1 [11]. Regardless, IPsec-IKEv1 is still in wide-spread use to this day, with the largest router producer in Germany AVM still only supporting IKEv1 in their routers [9]. We use IPsec-IKEv1 with ESP and main mode in this short paper and focus on the IKE protocol as it is the most interesting from an AAL and security standpoint.

The IKEv1 protocol works in two main phases, both relying on the Internet Security Association and Key Management Protocol (ISAKMP) protocol. A typical exchange between two parties can be seen in figure 1. Assuming two participants, an initiator and a responder, in phase one (Main Mode) the initiator sends a Security Association (SA) to the responder. A SA essentially details important security attributes for a connection like the encryption algorithm and key-size to use, as well as the authentication method and the used hash algorithm. These options are bundled in containers called proposals, with each proposal describing a possible security configuration. While the initiator can send multiple proposals to give the responder more options to choose from, the responder must answer with only one, provided it supports one of the suggested proposals. Next, the two parties perform a Diffie-Hellman key exchange and exchange nonces to generate a shared secret key. This secret key is used as a seed key for all further session keys. Following a successful key exchange, all further messages are encrypted. Finally, both parties exchange hashed authentication material (usually pre-shared keys or certificates) and verify the received hash. If successful, a secure channel is created and is used for phase two communication. The shorter phase two (Quick Mode) begins with another SA exchange, this time however the SA describes the security parameters of the ensuing ESP/AH communication. This is followed by a single acknowledge message from the initiator to confirming the agreed upon proposal. After the acknowledgment, all further communication is done via ESP/AH packets.

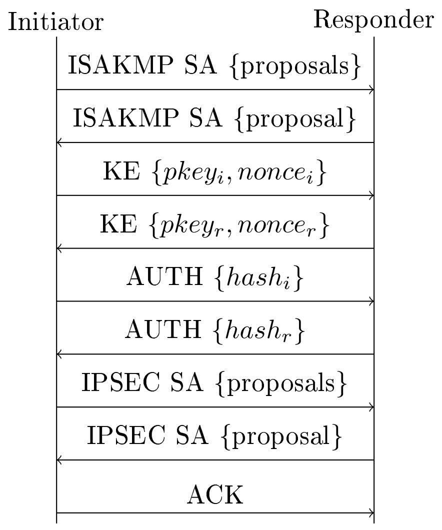

Fig. 1. IKEv1 between two parties

3 Related Work

Model learning of other network protocols like SSH [8], or TCP [7] has been performed in the past, with learned model being used for model checking the learned protocols. Both [13] and [5] learned models of the more related OpenVPN protocol [13] and used the learned models for fuzzing. In [14], learned models were used to fingerprint Bluetooth Low Energy Devices. Guo et al. used Automata Learning to learn and test the IPsec-IKEv2 protocol [10]. However, they used the LearnLib library to learn their automata and used the learned model for model checking. No fuzzing or fingerprinting was performed. Our goal is to in future work use the learned model for either fuzzing or to fingerprint different IPsec implementations.

4 Learning IPsec

4.1 Setup

We developed and tested our interface using two Virtual Box virtual machines (VMs) running standard Ubuntu distributions. All communication took place in an isolated virtual network to eliminate possible external influences. We designated one VM as the initiator and one as the responder to model a typical client server setup. The open source IPsec implementation Strongswan was installed on the responder VM and set to listen for incoming connections from the initiator VM. The Strongswan server was configured to used pre-shared keys for authentication, default recommended security settings and to allow unencrypted notify messages, which we used in our interface to reset the connection. To learn the model of the Strongswan server, simply run the *IPSEC_IKEv1_SUL* Python script which uses AALpy learning libraries with our custom Python IPsec client implementation to communicate with and learn the model of the server.

4.2 AALpy

AALpy is an Active Automata Learning library written in Python. It boasts support for deterministic, non-deterministic and stochastic automata, with support for various formalisms for each automata type. We use Deterministic Mealy machines to describe the IPsec server. However, learning automata with AALpy follows the same pattern, regarldess of the type of automata. To begin learning an automaton, we implement the AALpy SUL interface. This interface requires defining a *step* and *reset* method. We use *step* to model one step in the automata learning algorithm and *reset* to revert the SUL to an initial clean state. Before we can begin learning, we must also decide upon a suitable input alphabet encompassing the language known by the server. The chosen input alphabet is essentially made up of the initiator-to-responder messages shown in figure 1. AALpy then passes on words from the input alphabet to the *step* method and sends corresponding queries to the SUL. Finally, we decide on the type of equivalence oracle to use and can enable a few more options like caching and non-determinism checks. The chosen equivalence oracle is used in L* to test for conformance between the current hypothesis and the SUL, giving either a counterexample on failure or confirmation that we have learned the SUL. To separate our SUL interface class from the actual communication code, we designed a separate mapper class, implementing the necessary communication code for each message used in the IPsec-IKEv1 protocol. AALpy calls these mapper class methods in the *step* method of the SUL interface. This separation allows for easy future modifications to the message implementations as well as increasing the readability of the SUL interface class.

4.3 Mapper Class

The mapper class implements methods for each communication step in a typical IPsec-IKEv1 exchange, as described in chapter 2. We use the Python library Scapy[1] to construct ISAKMP packets as required by the IKEv1 protocol. This approach allows us to change fields and values of generated packets at will, opening the possibility of fuzzing for future work. This approach was made more difficult by the fact that Scapy does not support all the packets required by IPsec-IKEv1. To solve this problem, we implemented the missing packets in the Scapy ISAKMP class and used this modified version. We use another custom class to handle the sending and receiving of messages via UDP sockets, as well as parsing responses into valid Scapy objects.

As messages will be sent in random order during learning, we require a robust framework that correctly handles en/decryption of messages. For key management, we simply store the current base-keys but keep track of Initialization Vectors (IVs) on a per-message id (M-ID) basis. Additionally, we keep track of the M-IDs of server responses to detect and handle retransmissions of old messages. Each request, we store the response for use in the next message and update affected key material as needed. Most notably, the IVs are updated almost every request and differ between M-IDs. Informational requests also handle their IVs separately. For each request that we send, if available, we try to parse the response, decrypting it if necessary and resetting or adjusting our internal variables as required to match the server. This is required to continuously be able to parse server responses and extract meaningful information.

Automata learning requires a SUL reset method to be able to return to an initial starting point after each run. We implement this using a combination of the ISAKMP delete request and general ISAKMP informational error messages. While delete works for established connections currently in phase two of IKE, we require informational error messages to trigger a reset in phase one, as delete does not apply here. Implementation was hindered by at times unclear RFC-specifications, but this was overcome by manually comparing packet dumps and Strongswan logs to fix encryption errors.

[1] https://scapy.readthedocs.io/en/latest

5 Evaluation

6 Conclusion

6.1 A Subsection Sample

Please note that the first paragraph of a section or subsection is not indented. The first paragraph that follows a table, figure, equation etc. does not need an indent, either.

Subsequent paragraphs, however, are indented.

Sample Heading (Third Level) Only two levels of headings should be numbered. Lower level headings remain unnumbered; they are formatted as run-in headings.

Sample Heading (Fourth Level) The contribution should contain no more than four levels of headings. Table 1 gives a summary of all heading levels.

Table 1. Table captions should be placed above the tables.

Heading level	Example	Font size and style
Title (centered)	**Lecture Notes**	14 point, bold
1st-level heading	**1 Introduction**	12 point, bold
2nd-level heading	**2.1 Printing Area**	10 point, bold
3rd-level heading	**Run-in Heading in Bold.** Text follows	10 point, bold
4th-level heading	*Lowest Level Heading.* Text follows	10 point, italic

Displayed equations are centered and set on a separate line.

$$x + y = z \tag{1}$$

Please try to avoid rasterized images for line-art diagrams and schemas. Whenever possible, use vector graphics instead (see Fig. **??**).

Theorem 1. *This is a sample theorem. The run-in heading is set in bold, while the following text appears in italics. Definitions, lemmas, propositions, and corollaries are styled the same way.*

Proof. Proofs, examples, and remarks have the initial word in italics, while the following text appears in normal font.

For citations of references, we prefer the use of square brackets and consecutive numbers. Citations using labels or the author/year convention are also acceptable. The following bibliography provides a sample reference list with entries for journal articles [**?**], an LNCS chapter [1], a book [**?**], proceedings without editors [**?**], and a homepage [**?**]. Multiple citations are grouped [**?**,**?**,**?**], [**?**,**?**,**?**,**?**].

Acknowledgements Please place your acknowledgments at the end of the paper, preceded by an unnumbered run-in heading (i.e. 3rd-level heading).

References

1. Abhijith, M., Senthilvadivu, K.: Impact of vpn technology on it industry during covid-19 pandemic. In: IJEAST (2020)
2. Angluin, D.: Learning regular sets from queries and counterexamples. Information and computation **75**(2), 87–106 (1987)
3. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation **75**(2), 87–106 (1987). https://doi.org/https://doi.org/10.1016/0890-5401(87)90052-6, https://www.sciencedirect.com/science/article/pii/0890540187900526
4. Barker, E., Dang, Q., Frankel, S., Scarfone, K., Wouters, P.: Guide to ipsec vpns (2020-06-30 00:06:00 2020). https://doi.org/https://doi.org/10.6028/NIST.SP.800-77r1
5. Daniel, L.A., Poll, E., de Ruiter, J.: Inferring openvpn state machines using protocol state fuzzing. In: 2018 IEEE European Symposium On Security And Privacy Workshops (Euros&PW). pp. 11–19. IEEE (2018)
6. Ferguson, N., Schneier, B.: A cryptographic evaluation of ipsec (1999)
7. Fiterău-Broştean, P., Janssen, R., Vaandrager, F.: Combining model learning and model checking to analyze tcp implementations. In: International Conference on Computer Aided Verification. pp. 454–471. Springer (2016)
8. Fiterău-Broştean, P., Lenaerts, T., Poll, E., de Ruiter, J., Vaandrager, F., Verleg, P.: Model learning and model checking of ssh implementations. In: Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software. pp. 142–151 (2017)
9. GmbH, A.C.V.: Connecting the fritz!box with a company's vpn. https://en.avm.de/service/vpn/tips-tricks/connecting-the-fritzbox-with-a-companys-vpn/, accessed: 2022-09-09
10. Guo, J., Gu, C., Chen, X., Wei, F.: Model learning and model checking of ipsec implementations for internet of things. IEEE Access **7**, 171322–171332 (2019)
11. Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., Kivinen, T.: Internet key exchange protocol version 2 (ikev2). Tech. rep. (2014)
12. Muškardin, E., Aichernig, B.K., Pill, I., Pferscher, A., Tappler, M.: Aalpy: an active automata learning library. Innovations in Systems and Software Engineering pp. 1–10 (2022)
13. Novickis, T., Poll, E., Altan, K.: Protocol state fuzzing of an OpenVPN. Ph.D. thesis, PhD thesis. MS thesis, Fac. Sci. Master Kerckhoffs Comput. Secur., Radboud Univ (2016)
14. Pferscher, A., Aichernig, B.K.: Fingerprinting bluetooth low energy devices via active automata learning. In: International Symposium on Formal Methods. pp. 524–542. Springer (2021)
15. Pferscher, A., Aichernig, B.K.: Stateful black-box fuzzing of bluetooth devices using automata learning. In: NASA Formal Methods Symposium. pp. 373–392. Springer (2022)