RUHR-UNIVERSITÄT BOCHUM

RUB

RUHR-UNIVERSITÄT BOCHUM

Learning State Machines of SSH Server Implementations

Fabian Bäumer

Exposé Master's Thesis – May 16, 2021.
Chair for Network and Data Security

hgi Lehrstuhl für Netz- und Datensicherheit

Contents

1 Introduction

In the age of Smart Cities and the Internet of Things, the security of a single host has become an integral component in securing the technical environment and is often taken as granted in high-level applications. However, the fact that individual hosts can represent a single point of failure in terms of data privacy and data integrity must not be forgotten. As previously observed in cybercriminal attacks against large enterprises, governmental institutions or even individuals, a single compromised host can already yield a significant financial, organizational or even individual damage. Therefore, a secure infrastructure does not only consist out of inter-host security aspects like network isolation, organizational security and trust management but also requires secure hosts in the first place.

Nowadays the majority of desktop systems uses Windows rather than alternatives like MacOS, Linux or Chrome OS as its operating system [16]. While the market share of Linux based distributions in the desktop segment is negligible compared to Windows, the market share of those distributions in the website hosting segment is significant with over 75 % of websites in May 2021 running on Linux based operating systems [18]. Remote management on Windows based operating systems is often done using the built-in Remote Desktop Protocol (RDP) to establish a remote desktop session. In contrast, Linux based systems may not have a graphical User Interface (UI) installed thus requiring a text based approach to remote management. Historically in the early 90s, remote management of these systems was done using unprotected protocols like telnet or rlogin. The lack of cryptographic protection when using these protocols lead to the development of the Secure Shell (SSH) protocol, which is commonly used for remote management of Linux based operating systems nowadays.

As a direct consequence of SSH being common among Linux based hosts and the significant amount of servers using Linux as their operating system, any flaw in the protocol or its implementation has the potential to result in a devastating effect on the global IT security landscape. A facet of the security of network protocol implementations is implementing the underlying protocol state machines' transitions according to the standard. An approach in finding implementation flaws within these transitions is the usage of network protocol state fuzzing tools to extract the state machine from the implementation. This in turn allows one to compare the extracted state machine to the state machine defined within the according standards. Any deviation from the standards' state machine can either be analyzed and classified manually or verified using well-known model checking algorithms in order to find such implementation flaws.

2 State of Knowledge

The field of network protocol state machine learning is mainly based upon state machine learning algorithms. One of the most popular algorithms for state machine learning, the L* algorithm, was presented by Angluin in 1987 [2] and established the concept of the minimally adequate teacher framework, which restricts the type of questions an algorithm may ask during the process of learning a regular set. Several modern state machine learning algorithms, like the TTT algorithm developed by Isberner et al. in 2014 [8], work within the minimally adequate teacher framework and are based upon the L* algorithm.

Previous works in the field of network protocol state machine learning applied the aforementioned algorithms to Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) implementations in order to analyze the implemented state machines in terms of implementation flaws and basic security properties [5][13][19]. Furthermore, tools like SNOOZE [3] and Pulsar [7], developed in 2006 by Banks et al. and 2015 by Gascon et al. respectively, try to provide the required functionality within a framework in order to ease the adoption of these techniques to attack different network protocols.

In regards to the SSH protocol Fiterau-Brostean et al. applied state machine learning and model checking algorithms to SSH server implementations in 2017 [6]. In their paper they heavily restricted the learning alphabet to only include valid messages and messages expected to be sent by the client. A similar approach was used by Verleg in his Master's thesis in 2016 [17] and later improved by Yan in 2017 in his Master's thesis [20]. Verleg and Yan also heavily restricted the learning alphabet in the same way Fiterau-Brostean et al. did.

3 Research Questions

Implementations of complex protocols tend to disobey the protocols' specifications, especially in situations of unknown or unexpected messages. The SSH protocol maintains an internal state machine, where every transitioning edge is well defined by the standard. For example, on connect the client waits for the server to answer his initial version exchange with another version exchange message. If the server however does not obey the standards' specifications and start the key exchange prematurely, the client must not continue.

Another obstacle when implementing SSH is the fact that the SSH standard does not describe the underlying state machines explicitly. This may lead to ambiguity on certain transitions if the standards description lacks clarity. Poll and Schubert give explicit descriptions for the different flows of the transport layers' state machine when using a Diffie Hellman key exchange in [12]. As the protocol flow of the transport layer protocol depends on the negotiated cipher suite, their results can not be directly applied when using other common cipher suites. Inferring the standards' state machines therefore requires advanced study of the SSH standards and the extensions in question.

A systematical approach to verifying the implemented state machine of SSH server implementations uses well-known state machine learning algorithms in order to extract the state machine by choosing adequate inputs and observing the automatons corresponding output. The state machine learning algorithms deduce a model of the internal state machine which in turn can be analyzed further manually or by using model checking approaches.

As the original RFCs from 2006 have been expanded by multiple other RFCs over the course of the years, studying the entirety of the SSH ecosystem is not practical. Restrictions are required to narrow down the scope to specific key exchange algorithms and extensions. The restrictions applied within this thesis are based on the technical guideline TR-02102 published by the Federal Office for Information Security [4] and limit the scope of this thesis to at least the following key exchange algorithms:

- `diffie-hellman-group16-sha512`
- `ecdh-sha2-nistp256`

Both algorithms are recommended to be used up to the year 2027 and beyond. Furthermore, the `curve25519-sha256` key exchange algorithm, which is not yet present within the technical guideline, is considered optional. For encryption algorithms as well as client and server authentication this thesis will only cover the bare minimum as these parameters do not affect the protocol flow directly.

This thesis focuses on giving answers on the following three open research questions:

1. Do recent implementations of the internal SSH state machine comply with the SSH standards' state machine, especially in terms of unexpected (out of order), malformed (e. g. incorrect signatures or invalid content) or unknown (invalid) messages?

2. How do the internal state machines of recent SSH server implementations react in terms of state machine transitions and performed cryptographical operations when the client sends messages which are defined in the standard as server-side only?

3. What are the limitations of the network protocol fuzzing approach in terms of accuracy, time and space complexity and ease of use?

4 Methodology

In order to be able to answer the research questions raised in the previous chapter an appropriate SSH state machine learning tool is required. A tool capable of extracting an SSH state machine consists out of three main components, an SSH library used to generate and send SSH messages to the server and receive and parse SSH messages received from the server. Furthermore, a state machine learning component is required in order to extract the state machine based upon the messages sent and received to and from the server. The last component translates the protocol specific messages obtained by the underlying SSH component to the generic representation of the machine state learning component. A simplified visualization of the components and their respective interaction is depicted in figure 4.1.

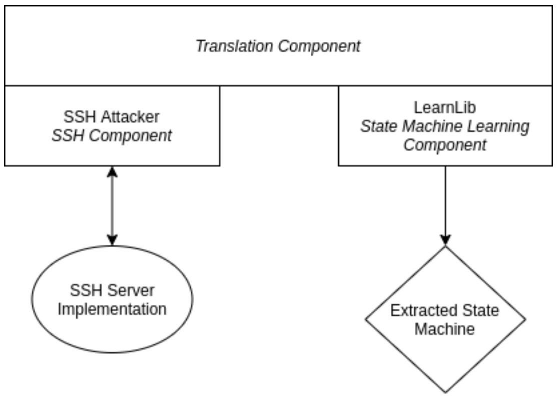

Figure 4.1: Conceptual Structure of the SSH State Learning Tool

As part of this thesis the translation component will be newly developed while already existing frameworks and libraries for the SSH and state learning components will be used. The SSH attacker framework developed by the Chair for Network and Data Security of the Ruhr University of Bochum allows for highly customizable workflows and payloads. Due to the lack of support in terms of server-side messages and the repository becoming stale in recent time, major updates to the framework are required in order to be able to use the framework as the underlying SSH component. For state machine learning one can use LearnLib [9] which is a framework actively developed by the Chair for Programming Systems at the TU Dortmund supporting several automata learning algorithms for mealy machines as well as Deterministic Finite Automaton (DFA) and Nondeterministic Finite Automaton (NFA) machines.

Once the state machine learning tool is able to reliably extract state machines from SSH server implementations, the tool will be used in order to extract state machines from a list of popular implementations. The list of server implementations to be tested strongly depends on the availability and popularity and has yet to be determined as part of this thesis. This will be done using a banner scan similar to the one conducted by Albrecht et al. in 2016 [1] using a combination of ZMap and ZGrab. The results allow for quantization of the usage percentages of different SSH server implementations and their respective versions. Based on these percentages, this thesis will at least cover the five most popular ones.

For analytical purposes the extracted state machines will be visualized using an adequate state machine visualization framework. This step may require minor adjustments to the output of the state fuzzing tool in order to be compatible with the visualization framework used. The visualized state machines will then be inspected and analyzed manually to verify basic security properties. An automated approach to verifying the extracted state machines is not focus of this thesis due to the complexity of model checking and the limited time available.

5 Draft Structure

1 Introduction

1.1 Motivation

1.2 Related Work

1.3 Contribution

1.4 Organization of this Thesis

2 Background

2.1 SSH

2.1.1 Architecture

2.1.2 Transport Layer Protocol

2.1.3 Authentication Protocol

2.1.4 Connection Protocol

2.1.5 Implementations

2.2 SSH Attacker Framework

2.2.1 Framework Architecture

2.2.2 Executing Static Workflows

2.2.3 Executing Adaptive Workflows

2.3 Automata

2.3.1 Deterministic Finite Automaton

2.3.2 Mealy Machine

2.4 Active Automata Learning

2.4.1 Definition

2.4.2 Minimally Adequate Teacher Framework

2.4.3 L* Algorithm

2.4.4 [...] *(If another algorithm will be used, it will be described here)*

6 Literature List

[1] M.R. Albrecht, J.P. Degabriele, T.B. Hansen, and K.G. Paterson: *A surfeit of ssh cipher suites.* In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, p. 1480–1491, New York, NY, USA, 2016. Association for Computing Machinery, ISBN 9781450341394. `https://doi.org/10.1145/2976749.2978364`.

[2] D. Angluin: *Learning regular sets from queries and counterexamples.* Information and Computation, 75(2):87–106, 1987, ISSN 0890-5401. `https://www.sciencedirect.com/science/article/pii/0890540187900526`, visited on 16.05.2021.

[3] G. Banks, M. Cova, V. Felmetsger, K. Almeroth, R. Kemmerer, and G. Vigna: *SNOOZE: Toward a Stateful NetwOrk prOtocol fuzZEr.* In S.K. Katsikas, J. López, M. Backes, S. Gritzalis, and B. Preneel (eds.): *Information Security*, pp. 343–358, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg, ISBN 978-3-540-38343-7.

[4] Federal Office for Information Security: *Cryptographic Mechanisms: Recommendations and Key Lengths.* Technical guideline TR-02102-4 (2021-01), Federal Office for Information Security, Mar. 2021.

[5] P. Fiterau-Brostean, B. Jonsson, R. Merget, J. de Ruiter, K. Sagonas, and J. Somorovsky: *Analysis of DTLS Implementations Using Protocol State Fuzzing.* In *29th USENIX Security Symposium (USENIX Security 20)*, pp. 2523–2540. USENIX Association, Aug. 2020, ISBN 978-1-939133-17-5. `https://www.usenix.org/conference/usenixsecurity20/presentation/fiterau-brostean`, visited on 16.05.2021.

[6] P. Fiterău-Broştean, T. Lenaerts, E. Poll, J. de Ruiter, F. Vaandrager, and P. Verleg: *Model Learning and Model Checking of SSH Implementations.* In *Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software*, SPIN 2017, p. 142–151, New York, NY, USA, 2017. Association for Computing Machinery, ISBN 9781450350778. `https://doi.org/10.1145/3092282.3092289`, visited on 16.05.2021.

[7] H. Gascon, C. Wressnegger, F. Yamaguchi, D. Arp, and K. Rieck: *Pulsar: Stateful Black-Box Fuzzing of Proprietary Network Protocols.* In B. Thuraisingham, X. Wang, and V. Yegneswaran (eds.): *Security and Privacy in Communication Networks*, pp. 330–347, Cham, 2015. Springer International Publishing, ISBN 978-3-319-28865-9.

[8] M. Isberner, F. Howar, and B. Steffen: *The TTT Algorithm: A Redundancy-Free Approach to Active Automata Learning.* In B. Bonakdarpour and S.A. Smolka (eds.): *Runtime Verification*, pp. 307–322, Cham, 2014. Springer International Publishing, ISBN 978-3-319-11164-3.

[9] M. Isberner, F. Howar, and B. Steffen: *The Open-Source LearnLib.* In D. Kroening and C.S. Păsăreanu (eds.): *Computer Aided Verification*, pp. 487–495, Cham, 2015. Springer International Publishing, ISBN 978-3-319-21690-4.

[10] H. Liang, X. Pei, X. Jia, W. Shen, and J. Zhang: *Fuzzing: State of the Art.* IEEE Transactions on Reliability, 67(3):1199–1218, 2018.

[11] R. McNally, K. Yiu, D. Grove, and D. Gerhardy: *Fuzzing: the state of the art.* Techn. rep., Defence Science and Technology Organisation Edinburgh (Australia), 2012.

[12] E. Poll and A. Schubert: *Rigorous specifications of the ssh transport layer.* Oct. 2011. `https://www.researchgate.net/publication/241880255_Rigorous_specifications_of_the_SSH_Transport_Layer`, visited on 16.05.2021.

[13] J. de Ruiter and E. Poll: *Protocol State Fuzzing of TLS Implementations.* In *24th USENIX Security Symposium (USENIX Security 15)*, pp. 193–206, Washington, D.C., Aug. 2015. USENIX Association, ISBN 978-1-939133-11-3. `https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/de-ruiter`, visited on 16.05.2021.

[14] M. Shahbaz and R. Groz: *Inferring Mealy Machines.* In A. Cavalcanti and D.R. Dams (eds.): *FM 2009: Formal Methods*, pp. 207–222, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg, ISBN 978-3-642-05089-3.

[15] J. Somorovsky: *Systematic Fuzzing and Testing of TLS Libraries.* In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, p. 1492–1504, New York, NY, USA, 2016. Association for Computing Machinery, ISBN 9781450341394. `https://doi.org/10.1145/2976749.2978411`, visited on 16.05.2021.

[16] StatCounter Global Stats: *Desktop Operating System Market Share Worldwide 2020.* `https://gs.statcounter.com/os-market-share/desktop/worldwide/2020`, visited on 16.05.2021.

[17] P. Verleg: *Inferring SSH state machines using protocol state fuzzing.* Master's thesis, Radboud University, Feb. 2016. `https://www.ru.nl/publish/pages/769526/z07_patrick_verleg.pdf`, visited on 16.05.2021.

[18] W3Techs: *Comparison of the usage statistics of Unix vs. Windows for websites.* `https://w3techs.com/technologies/comparison/os-unix,os-windows`, visited on 16.05.2021.

[19] T. Yadav and K. Sadhukhan: *Identification of Bugs and Vulnerabilities in TLS Implementation for Windows Operating System Using State Machine Learning.* In S.M. Thampi, S. Madria, G. Wang, D.B. Rawat, and J.M. Alcaraz Calero (eds.): *Security in Computing and Communications*, pp. 348–362, Singapore, 2019. Springer Singapore, ISBN 978-981-13-5826-5.

[20] Y. Yan: *SSH Implementations: State Machine Learning and Analysis.* Master's thesis, Delft University of Technology, Sept. 2017. `https://repository.tudelft.nl/islandora/object/uuid:8c807ce9-0ad6-4525-b7f3-c0271448040d`, visited on 16.05.2021.

[21] T. Ylonen and C. Lonvick: *The Secure Shell (SSH) Authentication Protocol.* RFC 4252, RFC Editor, January 2006. `http://www.rfc-editor.org/rfc/rfc4252.txt`, visited on 16.05.2021.

[22] T. Ylonen and C. Lonvick: *The Secure Shell (SSH) Connection Protocol.* RFC 4254, RFC Editor, January 2006. `http://www.rfc-editor.org/rfc/rfc4254.txt`, visited on 16.05.2021.

[23] T. Ylonen and C. Lonvick: *The Secure Shell (SSH) Protocol Architecture.* RFC 4251, RFC Editor, January 2006. `http://www.rfc-editor.org/rfc/rfc4251.txt`, visited on 16.05.2021.

[24] T. Ylonen and C. Lonvick: *The Secure Shell (SSH) Transport Layer Protocol.* RFC 4253, RFC Editor, January 2006. `http://www.rfc-editor.org/rfc/rfc4253.txt`, visited on 16.05.2021.

7 Schedule

		Week																									
Work Steps					**4**				**8**				**12**				**16**				**20**				**24**		
1	Literature Research	🟦	🟦	🟦																							
2	State Fuzzing Tool				🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦														
2.1	Def. Requirements				🟥																						
2.2	Ext. SSH Attacker					🟥	🟥	🟥																			
2.3	Main Tool Development								🟥	🟥	🟥	🟥	🟥														
3	Visualization													🟦	🟦	🟦											
4	Evaluation																🟦	🟦	🟦	🟦	🟦						
5	Thesis Writing																					🟦	🟦	🟦	🟦		
6	Buffer																									🟦	🟦