Testing Security Properties of Protocol Implementations – a Machine Learning Based Approach

Guoqiang Shu
The Ohio State University
shug@cse.ohio-state.edu

David Lee
The Ohio State University
lee@cse.ohio-state.edu

Abstract

Security and reliability of network protocol implementations are essential for communication services. Most of the approaches for verifying security and reliability, such as formal validation and black-box testing, are limited to checking the specification or conformance of implementation. However, in practice, a protocol implementation may contain engineering details, which are not included in the system specification but may result in security flaws. We propose a new learning-based approach to systematically and automatically test protocol implementation security properties. Protocols are specified using Symbolic Parameterized Extended Finite State Machine (SP-EFSM) model, and an important security property – message confidentiality under the general Dolev-Yao attacker model – is investigated. The new testing approach applies black-box checking theory and a supervised learning algorithm to explore the structure of an implementation under test while simulating the teacher with a conformance test generation scheme. We present the testing procedure, analyze its complexity, and report experimental results.

1. Introduction

Security protocols have become a foundation of the network and distributed applications. It is well known that ensuring the correctness of a security protocol is challenging [13] [10] mainly due to its inherent complexity and the potentially hostile execution environment. Various logic and model based approaches [13] [12] [25] have been developed to facilitate rigorous analysis of security protocols. However, existing works focus mainly on the specification of the protocol, while implementation flaws – another source of unreliability and insecurity – have been largely neglected by the research community. Protocol vulnerability caused by implementation and deployment is very common [22] [21] despite of the fact that most protocol specifications have gone through extensive validation and have been proven to be secure. Therefore, it is indispensable to detect protocol implementation security flaws.

Black-box testing is a commonly used approach to check conformance of a protocol implementation to its specification, and with recent advances in both formal modeling and automation [9] [14] [8] it can be conducted effectively. However, classic functional testing technique alone is not enough for security property testing due to the following fundamental reasons. Traditional conformance testing aims to verify that implementation achieves the same functionality as the specification. However, implementations are always more complicated than the specification with a lot of engineering details implemented, and conformance testing does not deal with the behaviors beyond what is specified. The limitation is implicitly reflected by assumptions about the implementation (e.g. number of states) in the existing testing approaches. Consequently, the practice of security testing in industry is presently conducted in an ad-hoc way, either relying on human insights or using heuristic strategies such as mutation testing.

To check protocol implementation security properties we propose a learning-based approach to conduct rigorous active testing. The key idea is to discover the internal structure of black-box implementations by active testing and to validate the security property on-the-fly. In this work we focus on a specific security property – Message Confidentiality (secrecy) under Dolev-Yao model of attackers [3]. We apply a learning procedure to explore the implementation structure that is beyond the specification. Inspired by the black-box automata

27th International Conference on Distributed Computing Systems (ICDCS'07)
0-7695-2837-3/07 $20.00 © 2007 IEEE

model checking theory proposed first by [17], we use a supervised learning approach with a conformance test generator serving as the teacher. For a black-box protocol implementation under test, we maintain an estimation (conjecture) model of its structure and update it successively as more is learned by applying testing sequences. Every conjecture is closer to the implementation than the previous one and after each conjecture is constructed a validation process is applied to check against message confidentiality property. The process terminates when no more new behaviors could be learned or a security violation is detected. This methodology is implemented in a software tool that can execute tests automatically

We first present a Finite State Machine (FSM) based formal model for protocol specification and implementation. The learning procedure is then illustrated: we extend the classic supervised learning algorithm L^* [1] to fit our protocol model and discuss the technique of simulating the teacher using active testing experiments to fulfill its role of supplying counterexamples for the learning algorithm. Finally we report experiments on real world protocols using our prototype tool that implements the whole process. The proposed methodology is an important complement to the existing effort of improving the security, reliability and resilience of communication protocol systems.

2. Related Work

Having been extensively applied in industry [21] [2], testing for security and reliability of network protocols is still a relatively new research topic. The large body of study on protocol validation and conformance testing do not directly apply due to the complexity of unknown implementation model. Other earlier testing works such as [24] [20] used human insight of the protocol to generate test cases targeting at certain vulnerabilities. While effective for a limited type of protocols and properties these heuristic approaches fail to systematically identify the security flaws of implementations. In [4] the author studied the security flaws of implementation, which could be revealed under particular execution environment, and proposed a mutation scheme to check it. Peled et al. studied the theoretical problem of black-box checking [17]. They analyzed strategies for finite automata model. Particularly it was pointed out that for the supervised learning [1], conformance testing techniques could be employed to simulate the teacher. Other works related to black-box checking includes [7] [5]. Learning technique is also used in validation of system model [23] where Gold's unsupervised learning theory is used to learn the reachable state set of a model. Formal technique for model-based test generation is another well studied theory that enables a core part of this work. In our investigation of security testing with supervised learning approach, we use the results from our previous work of efficient checking sequence generation [9] and extended FSM reachability analysis [19].

3. Security Protocol Models

In this section we describe the formal model we use to specify communication protocols and their implementations, as well as security properties.

3.1. Protocol Specification

We define the security *protocol message* type as follows. First, there are three atom types: *Int*, *Key* and *Nonce* and they contain bounded integers, a finite set K of symbolic public/private key pairs, and a finite set N of nonces, respectively. A protocol message is recursively defined as: (1) An atom; (2) Encryption of a message with a key; (3) Secure hash of a message; or (4) Concatenation of messages. A message can be represented by a string such as $E(k_b, (k_a. H(n_a)))$. Given $A=<K, N>$, denote $L(A)$ as the message type and also the set of messages formed using atoms in A. For $\Omega \subseteq L(A)$, denote $Encl(\Omega)$ as the enclosure of Ω under valid message operations: encryption, decryption, hash, concatenation and decomposition. With these we can define a SP-EFSM model that uses protocol message set $L(A)$ as input and output alphabet.

Definition 1 (SP-EFSM). A Symbolic Parameterized Extended Finite State Machine (SP-EFSM) is a 7-tuple $M=<S, s_{init}, A, I, O, X, T>$ where

1. S is a finite set of states;
2. s_{init} is the initial state;
3. A is a finite set of atoms, and $L(A)$ is the set of messages formed using A;
4. I is the input alphabet of size P; each input symbol contains a parameter of type $L(A)$;
5. O is the output alphabet of size Q; each output

symbol contains a parameter of type $L(A)$;

6. X is a vector denoting a finite set of state variables of type $L(A)$ with default initial values;

7. T is a finite set of transitions; for each $t \in T$, $t = <s, s', i, o, p, a>$ is a transition where s and s' are the start and end state; i and o and parameterized input and output; p is a predicate on state variables and input parameters, and a is an action on the state variables and input/output parameters.

The meaning of predicate and action follows the convention of state machine theory [9]. SP-EFSM is a compact representation of an equivalent Finite State Machine (FSM), referred to as its *Reachability Graph,* which can be obtained by exploring the configuration space and eventually removing the guard and action from the transitions [9]. Both our validation and learning algorithm use the reachability graph representation.

A *Finite Session Security Protocol Component* is specified by an SP-EFSM with some additional constraints. (1) It contains an integer variable *id*: the unique identifier of the component within the protocol system. (2) The input/output alphabet is each divided into two subsets: internal (interaction within the component) and external (interaction with another component). External input/output symbols carry a parameter to denote the message receiver (sender). (3) An integer session variable is initialized with *0* and incremented each time the machine leaves state S_{init}. Each atom from the nonce set N is associated with a specific session. According to (2) we denote an external input message received from M_a as $M_a?i$ and an output to M_a as $M_a!o$. The semantics of message sending/receiving follow the typical synchronous communication model such as in CSP (Communicating Sequential Processes).

Definition 2 (Protocol Specification). A security protocol is modeled as a set of finite session protocol components: $M_{spec} = \{M_1, M_2, \cdots, M_C\}$, where each component M_i has a unique *id*, and that they all share the same message type L.

Each component machine M_k represents a principal in the protocol system. A trace from the specification is a sequence of ordered I/O events generated by the component machines. Any external I/O event from the traces corresponds to a message exchange between two components, and they are observable by the network monitor. An observable trace is therefore defined as a sequence of messages exchanged among components.

Definition 3 (Observable Message Trace)**.** An observable message trace from a protocol M_{spec} with message type L is $tr = \{<s_1, r_1, msg_1>, <s_2, r_2, msg_2>, \ldots, <s_k, r_k, msg_k>\}$, where the messages are sequentially generated by M_{spec} with no other messages in between, and each triple contains sender id s_i, receiver id r_i and the message msg_i in L. Denote $Msg(tr)$ as the set of messages appeared in tr. Denote all message traces M_{spec} generates as $TR(M_{spec})$.

3.2. Attacker Model and Message Confidentiality

The specification formalism we have introduced so far defines the honest parties in the protocol system, while we shall also define the behaviors of the malicious party. In our previous study of testing security protocols [20] we adopt Dolev-Yao's assumption [3] of an active attacker. Essentially the attacker is a legitimate principal in the protocol with a valid *id* that can initiate or participate in a session. In addition, the attacker can intercept any messages and inject an arbitrary message it can construct using its current knowledge. The behavior of a Dolev-Yao attacker could be defined by a single-state SP-EFSM M_A with a variable Ω whose value is the set of messages the attacker could obtain (see [20]) and is initialized as Ω_0. Note that the existence of Dolev-Yao attacker alters the message exchange behavior of the whole system to allow message interception and injection. Passive attacker (or eavesdropper) could be treated as a special case of Dolev-Yao attacker that injects a message to the original recipient immediately after interception.

Among the various security properties, we focus on secrecy, also known as *Message Confidentiality*. Given the specification of protocol $\{M_1, M_2, \ldots, M_C\}$ and the attacker M_A, the global behavior of the whole protocol system under investigation is described by the Cartesian product of all the machines: $M_1 \times M_2 \times \cdots \times M_C \times M_A$. Since all the transitions may involve one of the two intruder transitions, a message trace produced by the system can hence be described by an interleaving sequence of *Intercept* and *Inject* transitions; each contains a message as parameters. Intuitively, the protocol is

insecure if the attacker could learn a secret from any message trace.

3.3. Validation and Testing Problem

Given a protocol specification M_{spec} and a set of secret messages S, it is of primary interest to check whether M_{spec} is secure, w.r.t. S. This is the classic protocol validation problem and has been extensively studied. It is well known that in general the problem is undecidable, and even with a finite number of messages and sessions it is NP-hard [6][18]. In our framework, we restrict the problem to finite session and it can be reduced to a state reachability problem [9] of the integrated model.

We now turn to black-box testing problem. An implementation of protocol component specified by M_i could be any deterministic machine B_i that shares the I/O alphabet and message language L with M_i. The definition of insecurity is naturally extended for protocol implementation as follows: If the attacker can derive the secret using its initial knowledge Ω_0 and message traces, then there is a security violation. Formally we have:

Definition 4. An implementation $B = \{B_1, B_2,\ldots, B_C\}$ is insecure with regard to the confidentiality of message $m \in L$ if and only if there exists a message trace tr from B such that $m \in Encl(\Omega_0 \cup Msg(tr))$.

Obviously, the goal of message confidentiality testing is to find security violation of a given black-box implementation B.

4. Active Testing

In this section we present our learning-based testing approach.

4.1. Learning-based Procedure

Given protocol specification $\{M_1, M_2,\ldots, M_C\}$ and an implementation $\{B_1, B_2,\ldots, B_C\}$, we want to find violation of message confidentiality of the latter. Notice first that in practice not all the components need to be involved in testing. Specifically, if the components are developed by different organizations or have different credibility, we only want to focus on some of them. Assume that the first k components are under test and we can compose a new system using B_1 to B_k and the specification for others, i.e. $\{B_1, B_2,\ldots, B_k, M_{k+1},\ldots, M_C\}$. The essence of testing a component is to learn the structure of its reachability graph which again is an equivalent representation of an SP-EFSM model. Following the theoretical approach of [17], we employ the following procedure shown in Fig. 1. For each black-box component B_i under test, an estimation of it (B^*_i) is maintained. B^*_i is represented as an FSM and initialized as an empty FSM. These estimations are updated every iteration according to our learning algorithm described later, and ideally they will converge to the target implementation B_i. In section 4.2 we present modification to the original learning algorithm L^* (for automata) to fit our need. Once a new estimation is formed, we will run a validation algorithm to validate message confidentiality according to definition 4. The algorithm calculates the composed reachability graph of $\{B^*_1, B^*_2,\ldots, B^*_k, M_{k+1},\ldots, M_C\}$ and search for a state of security violation. The validation can yield one of the following two results:

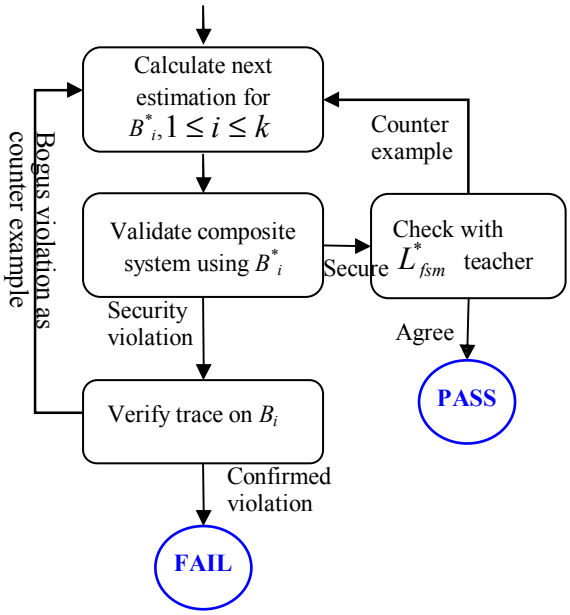


Fig. 1: Testing Procedure

Case 1. If no violation is found, we submit the estimation to the teacher of the learning algorithm. The teacher is responsible of answering the question of whether the current conjecture (estimation) is identical to the target black-box. In the case of a negative answer, the teacher will provide a counterexample - in our case a test sequence, to contribute to the next estimation. If the answer is positive, the test process terminates with result "PASS".

Case 2. On the other hand, the validation algorithm may also find a message trace leading to violation. Due to the nature of the learning algorithm, this trace may not actually exist in the implementation. In another word, it could be a false positive introduced by the estimation. We can simply experiment the trace on the black-box to filter those bogus traces. If the trace is verified, we terminate the testing procedure and claim "FAIL", otherwise the trace witnesses the discrepancy between the estimation and the target, and, therefore, is used as a counterexample as if it were provided by the teacher.

Finally, as shown later in this section, if the procedure finishes with a "PASS", it does not necessarily mean the last estimation is identical to the implementation; instead it means we can no longer find the difference or we choose not to do so.

In this architecture two issues are most important and we elaborate them in the following subsections. First, we study our variation of L^* learning algorithm; then we focus on how to simulate the teacher using active testing techniques.

4.2. FSM Learning Algorithm L^*

L^* is the classic supervised automata learning algorithm proposed by [1]. Here we modify it to suit our finite state machine model. The new algorithm ${L_{fsm}}^*$ assumes the presence of a teacher that can answer three types of queries: (1) *Output Query:* this corresponds to "membership query" in L^*. Given an input sequence, the teacher gives the corresponding output sequence produced by the implementation. (2) *Counterexample:* the teacher generates an input sequence for which the conjectured machine produces an incorrect output. (3) *Alphabet Augment:* the teacher gives a set of input symbols that are not presently included in the conjecture. This could be seen as a particular type of counterexample. Note that in L^* the learner starts with knowing the complete input alphabet, however, this assumption is not true in our protocol model because the alphabet usually has a formidable size (number of messages in L). Our algorithm will start with a subset (usually those included in the specification) and then gradually learn new inputs from the teacher.

The implementation of ${L_{fsm}}^*$ is similar to L^*. A data structure, called observation table, is maintained where each row represents a distinct state and each column represents a separating sequence of the current conjecture. The header of a row records a set of input sequences that can lead the machine to the state while the content of a row stores the output from the state for each of the separating sequence. The output is a Boolean value in L^* (for automata) and is a finite output sequence for FSM, instead. Due to the space limit we omit the complete formal description of this algorithm. Using a similar proof as [1] we can show that for a target minimized FSM with P input symbols and at most N states our algorithm makes at most $P+N-1$ incorrect conjectures before it eventually learns the target. As we shall see in next section, the efficiency and cost of learning algorithm depend significantly on how the teacher is implemented, particularly, how it generates counterexamples and alphabet augments.

4.3. Implementation of Teacher in ${L_{fsm}}^*$

This section discusses the implementation of teacher in our learning framework. Given a black-box B_i and the SP-EFSM specification M_i with protocol language L, the teacher replies to three types of queries listed in Section 4.2. Among them output query is the easiest one: for an input sequence, we send it to B_i and the reply is the observed output. However, conjecture is much more complicated; black-box testing technique is used [17]. For a guess ${B^*}_i$, the teacher is to show the difference between it and the target B_i; we have to figure out a test sequence for this. We now present our algorithm.

Algorithm 1 (Counterexample for ${L_{fsm}}^*$)

Input: Conjecture ${B^*}_i$, target B_i, protocol language L, Δ
Parameters:
$0 \le \boldsymbol{P_1} \le 1$: fraction of checking sequences to apply
$0 \le \boldsymbol{P_2} \le 1$: fraction of new symbols learned iteration
$1 \le \boldsymbol{T} \le |L|^{\Delta}$: number of state probing sequences to use
Output: counterexample or new symbol set or "*agree*"
begin

1. $\{Seq_{1a}Seq_{1b}, Seq_{2a}Seq_{2b},...,Seq_{Ka}Seq_{Kb}\}$ = checking sequences of ${B^*}_i$;
2. select KP_1 sequences from above randomly;
3. **foreach** selected sequence $Seq_{xa}Seq_{xb}$ **do**
4. if $output({B^*}_i, Seq_{xa}Seq_{xb}) \neq output(B_i, Seq_{xa}Seq_{xb})$ return $Seq_{xa}Seq_{xb}$;
5. **do** T times
6. generate random sequence $I_1 I_2 ... I_\Delta$;

7. select a random checking sequence $Seq_{xa}Seq_{xb}$;
8. if $output(B^*_i, Seq_{xa} I_1 I_2 ... I_\Delta Seq_{xb}) \neq output(B_i, Seq_{xa} I_1 I_2 ... I_\Delta Seq_{xb})$ return $Seq_{xa} I_1 I_2 ... I_\Delta Seq_{xb}$;
9. $L' = L - I(B^*_i)$;
10. $S = \{\}$;
11. **foreach** I_x in L' **do**
12. with probability $P_2 |L|/|L'|$ do
13. $S=S+\{I_x\}$;
14. if ($S \neq \{\}$) return S;
15. return "*agree*";

end

The first step of the algorithm is to test whether B_i conforms to B^*_i by using a conformance testing suite. Let P_i and N_i be the number of inputs and states of B^*_i. Using the classic checking sequence algorithm we construct (line 1) a set of input sequences of cardinality $O(P_i N_i^2)$ of total length $O(P_i N_i^3)$. Each sequence contains two sub-sequences: one transfers the machine to a certain state and the other verifies that state. The time complexity of this algorithm is $O(P_i N_i^3)$. We select a subset of the test suite (line 2) and execute it on B_i. If any test case fails it is returned as a desired counterexample.

If B_i passes all checking sequences, it is either identical to the guess or contains more states. Unfortunately the cost of probing extra states is inherently exponential [9]. To find new states we generate probing sequences of a form: $Seq_x I_1 I_2 ... I_\Delta Seq_y$, where the head and tail component $Seq_x Seq_y$ come from the constructed testing sequence, and the middle part is a sequence of random input symbols of B^*_i of length Δ (lines 7-8). This type of probing helps us learn up to Δ extra states, and, obviously, we can only afford to try a small number (T, line 5) of all such sequences.

However, the conjecture can be incorrect because it may miss some inputs. In our protocol model not all the messages in L are necessarily used as input in the specification. In fact specifications of real protocols are usually very sparsely defined on inputs, while a large number of unspecified (informally referred to as ill-formatted, abnormal) inputs are ignored by the implementations. Due to errors an implementation may have more inputs enabled than we explore. Naturally, the first conjecture is made by using only those specified input symbols, and at each iteration the teacher chooses a set of unexplored symbols (lines 10-14) as a reply to the learning algorithm.

For conformance test sequences each of length $O(N^*)$, a total number of $O(P_1 K) = O(P_1 P^* N^{*2})$ will be constructed and executed where N^* is the number of states in B^*_i and P^* is the size of its alphabet. Similarly $O(T)$ probing sequences and $O(P_2 L)$ new inputs will be processed. Hence the total cost of the teacher in one iteration is $O(P_1 P^* N^{*3} + TN^* + P_2 P)$. On the other hand, each counterexample it generates has length $O(N^*)$.

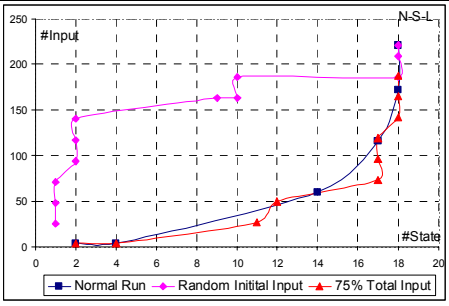
Note that at line 1 we do not actually produce any test sequence; instead it is postponed until it is decided which one to execute. On the other hand, in every round $L_{fsm}{}^*$ algorithm takes time $O(P^* N^{*2})$ to update the observation table and compute the next conjecture. Since there are at most $O(N+1/P_2) = O(N)$ iterations, the total cost of the learning procedure for one protocol component is at most $O(P_1 PN^4 + TN^2 + PN^3 + Nf_{ver}(N))$ where $f_{ver}(n)$ is the cost of validating the security for a conjecture with n states.

5. Software Tool and Experiments

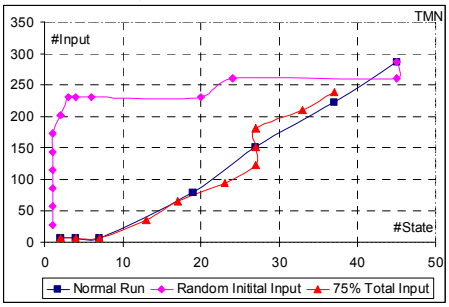
We conduct controlled experiments using a prototype tool developed for this work that implements the learning algorithm as well as the teacher simulation algorithm. We apply the tool to three well-known security protocols: Needham-Schroeder-Lowe (N-S-L) mutual authentication protocol, TMN key exchange protocol, and SSL 3.0 handshake protocol. For each of them, one component is chosen to be tested and an implementation is used as the target black-box. In N-S-L protocol implementation we introduce a flaw first discovered by [10] in the initiator component. For TMN, we use the specification of client as the implementation, which itself is insecure [11]. For SSL, we use one open source implementation of the client, and use the Sun Java Development Kit (JDK) 1.3 server implementation as specification. From manual inspection of the client source code, we observe the discrepancy between it and the specification, however, both our test approach and human reasoning come to a same conclusion that such discrepancy does not lead to security flaws. Details of the three implementations are summarized in Table I.

Table 1. Three protocols used in experiments (a) N-S-L (b) TMN (c) SSL 3.0

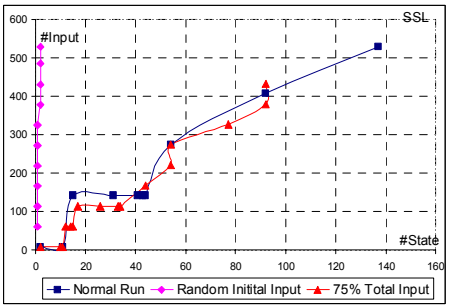
	# State	# of msg ($\|L_0\|/\|L\|$)	# of input in spec.	Spec. is secure	Impl. is secure
N-S-L	18	4/978	221	Yes	No
TMN	45	6/6665	287	No	No
SSL	137	7/1575	529	Yes	Yes



(a) M_{init} in N-S-L Protocol



(b) M_{client} in TMN Protocol



(c) M_{client} in SSL Protocol

Fig. 2: Result of Learning-based Testing

For each protocol implementation, we run our testing algorithm three times. The results are depicted as Fig. 2. We record the information of each conjecture our algorithm makes: number of state (X-axis) and number of input (Y-axis). Each point represents one conjecture and the big one on top-right corner is the target implementation. For the first run, we start the learning with a set of normal inputs (L_0 in Table 1). We use parameters P_1=1, T=1024, P_2=0.25, Δ=16. For all three protocols the target is successfully learned, though the message confidentiality validation result is different: for N-S-L, at the 5th conjecture violation is identified and confirmed; for TMN all conjectures are insecure (since the specification itself is flawed); and for SSL the implementation is confirmed secure. We also run the algorithm with other two different configurations. Since the total number of input is large, we run the algorithm once with P_2 = 0.1 and only try up to 75% of the total input. Not surprisingly, the result shows that missing inputs can be critical; it prevents us from learning all the states (hence we may possibly miss the security violation). On the other hand, instead of starting with normal inputs, we try random initial input set (with P_2=1, T=1024 and P_2=0.1). For both N-S-L and TMN the algorithm finishes with the correct conjecture, and for SSL it finishes with a 2-state wrong conjecture. This particular case shows the importance of selecting adequate initial input set. A reasonable selection (e.g. from protocol specification) helps the learning algorithm identify the basic structure of the black-box and some critical states, which otherwise can only be discovered by extensive and expensive probing for counterexamples, and may never succeed. In summary, the experiments on these three real protocols indicate that proposed learning-based testing methodology is effective.

6. Conclusion

In this paper we propose a learning-based testing method for checking security protocols message confidentiality. The main challenge is the complexity of protocol implementation beyond its specification. An SP-EFSM protocol model as well as message confidentiality property is formally defined. The new testing architecture modifies and applies the classic supervised machine learning algorithm to explore the structure of black-box implementations for checking security properties. We present algorithms to simulate the teacher using conformance testing techniques, and our experiments on real protocols show that the proposed approach is effective.

Acknowledgement

This work has been supported in part by NSF awards CNS-0403342 and CNS-0548403 and by DoD award N41756-06-C-5541.

We are deeply indebted to Mohamed Gouda for his constructive and insightful comments for preparing the final version of this paper. We thank the anonymous reviewer of ICDCS07 for their valuable reviews and suggestions.

References

[1] D. Angulin. Learning regular sets from queries and counterexamples. Information and Computation, 75, pages 87-106, 1987.

[2] M. Curphey and R. Arawo. Web application security assessment tools. IEEE Security & Privacy Magazine,Vol.4(4), pages 32 - 41,2006.

[3] D. Dolev and A. Yao. On the security of public-key protocols. IEEE Transaction on Information Theory 29, pages 198-208, 1983.

[4] W. Du and A P. Mathur. Testing for software vulnerability using environment perturbation. In Proceeding of the International Conference on Dependable Systems and Networks (DSN), 2000.

[5] E. Elkind, B. Genest, D. Peled, H. Qu: Grey-box checking. In Proceedings of IFIP FORTE, pages 420-435, 2006.

[6] S. Even and O. Goldreich. On the Security of Multi-Party Ping-Pong Protocols. In Proceedings of 24th IEEE Symposium on the Foundation of Computer Science, pages 34-39, 1983

[7] A. Groce, D. Peled, M. Yannakakis, Adaptive model checking, In Proceedings of International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TSCAS), pages 357–370, Apr. 2002.

[8] D. Lee, D. Chen, R. Hao, R. E. Miller, J. Wu, and X. Yin. A formal approach for passive testing of protocol data portions. In 10th IEEE International Conference on Network Protocols (ICNP), pages 122–131, 2002.

[9] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines - A survey. In Proceedings of the IEEE, pages 1090–1123, 1996.

[10] G. Lowe. Breaking and fixing the needham-schroeder public-key protocol using FDR. In Proceedings of International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TSCAS), 1996.

[11] G. Lowe and B. Roscoe. Using CSP to detect errors in the TMN protocol. IEEE Transactions on Software Engineering, 23(10), 1997.

[12] W. Marrero, E. Clarke, and S. Jha. Model checking for security protocols. Technical Report CMU-CS-97-139, Carnegie Mellon University, 1997

[13] C. Meadows. Applying formal methods to the analysis of a key management protocol, Journal of Computer Security, 1, pages 5-53, 1992.

[14] J. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using MurOE. In Proceedings of the 1997 IEEE Symposium on Security and Privacy, pages 141-151, 1997.

[15] J.C. Mitchell, V.Shmatikov, and U. Stern. Finite-state analysis of SSL 3.0, Seventh USENIX Security Symposium, 1998.

[16] R. Needham, M. Schroeder. Using encryption for authentication in large networks of computers. Communications of the ACM, 21(12), pages 993-999, 1978.

[17] D. Peled, M. Y. Vardi, M. Yannakakis. Black-box checking, In Proceedings of IFIP FORTE/PSTV, 1999.

[18] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In 14th IEEE Computer Security Foundations Workshop, pages 174—190, 2001.

[19] G. Shu and D. Lee. Network protocol system fingerprinting – A formal approach. In Proceedings of IEEE INFOCOM 2006.

[20] G. Shu and D. Lee. Message confidentiality testing of security protocols - Passive Monitoring and Active Checking. In Proceedings of IFIP TestCom, 2006.

[21] H. Thompson. Application penetration testing. IEEE Symposium on Security and Privacy. 3(1), pages 66–69, 2005.

[22] H. Thompson. Why security testing is hard. IEEE Symposium on Security and Privacy. 1(4), pages 83-86, 2003.

[23] A. Vardhan, K. Sen, M. Viswanathan, G. Agha. Learning to verify safely properties. International Conference on Formal Engineering Methods (ICFEM), 2004.

[24] G. Wimmel, J. Jürjens, Specification-based test generation for security-critical systems using mutations. In Proceedings of ICFEM, pages 471-482, 2002.

[25] A Yasinsac and J. Childs. Analyzing Internet security protocols. In Proceedings of the Sixth IEEE International Symposium on High Assurance Systems Engineering (HASE), pages 149 -159, 2001