A Formal Verification Method for Security Protocol Implementations Based on Model Learning and Tamarin

Xieli Zhang¹,², Yuefei Zhu¹,², Chunxiang Gu¹,²*, Xuyang Miao¹,²

¹ State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou, Henan, 450001, China
² Henan Key Laboratory of Network Cryptography Technology, Zhengzhou, Henan, 450002, China
*Corresponding author’s e-mail: chunxianggu.meac@gmail.com

Abstract. The verification of security protocol implementations is notoriously difficult and important. In this paper, combining with the model learning using Tamarin, a formal verification tool of protocol specification, a formal verification method for security protocol implementations is proposed. We extract state machine information from protocol implementations by model learning, and determine suspicious paths in the finite-state machines by cross-validation between different implementations of the same protocol; and then verify whether the suspicious paths violate the security properties of the protocol using Tamarin. It can be used to detect logical errors in protocol implementations and avoid relying on expert experience to make compliance rules from protocol documents when using model checking tools. The effectiveness of this method is demonstrated by the vulnerability detection of the typical ChangeCipherSpec in the TLS protocol implementation. The method proposed can help developers to develop more robust implementations of security protocols.

1. Introduction

The verification of security protocol has always been a research hotspot. Many formal verification methods for protocol specifications have been proposed, and lots of automatic formal verification tools have been developed[1-3], among which Tamarin is one of the most powerful tools. However, some protocols that have been verified still have vulnerabilities in implementations. Security verification technology for protocol implementations is gradually getting more attention[4,5].

It is not feasible to verify the protocol implementations directly in practice[6]. General software analysis methods are used to analyse security protocol implementations, such as rule-based method to detect protocol implementations[7]. In recent years, symbolic execution technology has been used to analyse security protocols and achieved good results[8-10]. However, these methods do little to detect logic errors in complex protocol interactions.

Compared with the protocol specification, a protocol implementation has finer granularity and more complex logic. Therefore, the general method is to extract the model from the protocol implementation and then formally verify the abstracted model. Based on this idea, many schemes on protocol code verification have emerged[4,5]. A security protocol program is a complex system interacting with environment, and its interaction behaviours can be modeled by finite-state machine (FSM). A protocol's FSM contains the sequential logic of message exchange. Model learning[11] technology provides support for obtaining high-level abstract model of protocol from a protocol.
implementation. Some attempts have been made to combine model learning and symbol execution techniques to improve the efficiency of protocol analysis[12].

This technology can be used to learn the FSM model from a protocol implementation in the form of black-box, by interacting with the protocol program[13-18]. After extracting the FSM model from a protocol implementation, we need to verify whether the state machine model complies with the protocol specification. However, the correct protocol state machine is not explicitly given in the protocol specification, and the model learned by model learning is always complex. How to check that the FSM learned from a protocol implementation does not violate the protocol specification is a problem worth studying and exploring. How to check that the FSM corresponding to a protocol implementation does not violate the protocol specification is worth studying and exploring.

Ruiter et al.[13] use manual experience to extract rules from protocol specifications to detect TLS state machine models inferred by model learning. Paul et al.[14,15,18] combine model learning and model checking to check whether the state machine is correct in the implementation of security protocol. Firstly, the FSM model of a security protocol program is inferred by model learning; then the FSM model is described using the model language supported by the model checking tool; finally, the FSM model is automatically detected by the tool whether it complies with the predetermined rules. The violation path in the FSM is derived for further analysis if it exists. Guo et al.[17] use a similar method to analyse the StrongSwan, an open source project of IPsec protocol implementation, and find a new logic vulnerability. However, the disadvantage of this approach is that the rules required for model checking need to be made manually according to protocol specifications, which rely heavily on expert experience. When analysing different protocols, it requires repeated manual work to read protocol specifications in detail in order to extract rules. Therefore, this paper proposes a formal verification method for security protocol implementations. The method combines the model learning technology and the formal verification tool Tamarin. Tamarin tool is used to verify the security properties of the FSMs of protocol implementations inferred by model learning. This method can address the problem that model checking method needs to rely on expert experience to extract rules from protocol specifications.

The main contributions of this paper are as follows.
1. A method combining model learning and formal verification tool Tamarin is proposed, which can be used to detect whether protocol implementations violates the protocol specifications.
2. The method of detecting protocol FSMs with Tamarin tool is given. Firstly, the suspicious path in the protocol state machine is preliminarily determined by cross-validation, and then the path in the protocol state machine is transformed into a Tamarin session. Finally, the Tamarin tool is used to automatically verify the transformed session.
3. Experimental results show that the proposed method can be used to detect logic errors in protocol implementations. The ChangeCipherSpec vulnerability in the TLS implementation is successfully detected by experiments, which shows the effectiveness of the proposed method in detecting logical errors in the protocol implementations.

The remaining Sections are organized as follows. Section 2 introduces the model learning and Tamarin. The details of the proposed method are described in Section 3. In Section 4, we apply this method to a earlier version of OpenSSL. Section 5 summarizes the work of the full paper.

2. Preliminaries

2.1. Model learning
This section introduces Mealy machine and a classical framework of model learning.

A Mealy machine can be represented by a six-tuple \( (S, I, O, \delta, X, s_0) \), where \( S \) is a finite set of states, \( I \) is a finite set of inputs, \( O \) is a finite set of outputs, \( \delta \) is a set of state transition relations: \( S \times I \rightarrow S \), \( X \) is an output transition relations: \( S \times I \rightarrow O \), \( s_0 \) is an initial state and \( s_0 \in S \). A transition relation in \( \delta: s_k \times i_k \rightarrow s_{k+1} \), means that after receiving the input \( i_k \) in the current state \( s_k \), the state is converted to \( s_{k+1} \). A transition relation in \( X: s_k \times i_k \rightarrow o_k \) means that after receiving the input \( i_k \) in
the current state $s_k$, the output is $o_k$, in which $s_k \in S$, $s_{k+1} \in S$, $i_k \in I$, $o_k \in O$. Mealy machine's output depends on the current state and input. Further, starting from the initial state, given an input sequence, the state transition sequence and the output sequence are also determined.

Minimally Adequate Teacher (MAT)\(^{[11]}\) is a classical framework of model learning. MAT is suitable for learning abstract models of software systems in a black box way. The model learning of target system is viewed as a query-response game between a learner and a teacher. The learner infers the state machine model by querying to the teacher, that is a software system under test. At the initial stage of the game, the learner has the knowledge of input set $I$ and the output set $O$.

The learner obtains the Mealy machine model of a software through two types of queries:

1) Membership query. The learner queries for the output sequence corresponding to an input sequence, and the teacher gives the correct output sequence.

2) Equivalence query. The learner asks the teacher whether a Mealy machine $M'$ is correct, that is, whether $M'$ is equal to the real state machine, and the teacher gives the answer "yes" or a counterexample. If there is a counterexample given, the Mealy machine should be modified until there is no counterexample in the limited queries.

After that, it is considered that the Mealy machine learned is approximately equivalent to the correct Mealy machine model of the teacher.

2.2. Tamarin

Tamarin is a well-known analysis tool for security protocols. It can automatically perform formal modeling and verification of security protocols based on symbolic model. Through its so-called rewrite rules, the behaviours of different roles of the protocol is specified. A labeled transition system is defined by these rewrite rules, and its states consists of adversary's knowledge, network message, protocol state, etc. Tamarin uses equality theory to support common cryptographic primitives such as encryption and hashing. Security properties are modeled using trace properties.

Here is a brief introduction to the two important ingredients in Tamarin modeling. For more details, please refer to the relevant manuals\(^{[20]}\).

Rules: A Tamarin rewrite rule consists of a name and three parts. Each part is a set of facts: one is the premise section of the rule, one is what is called an "action fact," and one is the result section of the application of the rule. For example, the following rule is named rule_example, and its premise is $[\text{Fact1}, \text{Fact2}]$. This rule can be applied to obtain $[\text{Fact3}, \text{Fact4}]$ when the premise is satisfied.

```
rule rule_example:
[ Fact1, Fact2 ] --[action fact]--> [ Fact3, Fact4 ]
```

Facts: Facts in Tamarin model are of the form $F(t_1, ..., t_n)$ for a fact symbol $F$ and terms $t_i$, and $F$ have a fixed arity $n$. In, Out and Fr are three types of built-in facts.

- Fact In is used to model receiving messages over untrusted network controlled by Dolev-Yao adversary.
- Fact Out is used to model sending messages over untrusted network controlled by Dolev-Yao adversary.
- Fact Fr is used to model generating random values.

3. Method

The verification framework is shown in figure 1. It is a combination of model learning and formal verification tool Tamarin. Model learning is used as the front end to infer the Mealy machine model of a security protocol implementation. And the verification tool Tamarin is used as the back end to verify whether Mealy Machine obtained by model learning conforms to security properties of the protocol specification. A transition path in Mealy Machine from the initial state to the termination state can be transformed into a protocol session described in sphy, which is the Tamarin's modeling language. In this way, the Tamarin tool can automatically verify whether a path of Mealy machine complies with the protocol security properties. Path filter is optional and is used to filter suspicious paths in the
Mealy Machine, in order to reduce the workload of Tarmarin analysis. Some details are discussed below.

3.1. Learning model from security protocol implementations

We learn the high-level abstract model (Mealy machine) of the target protocol implementation in a black box way according to the MAT learning framework. In order to avoid the model learning falling into indefinite, it is necessary to ensure that the learning target protocol program is deterministic, meaning that it produces consistent output for the same input under the same state. In other words, in the initial state of the protocol program, for a certain message sequence, its output message sequence is determined. This is usually guaranteed by fixing the initialization configuration.

In a Mealy machine, the state is determined by the sequence of input and output. Input and output sequences are the most fundamental information in the Mealy Machine. Before inferring the model of a protocol implementation, we need to determine what the input and output sets of the model are. Concrete packets can be replaced by abstract message types when describing a security protocol. Similarly, a sequence of input packets for a protocol program can be abstracted into a sequence of the input message types, and so can the output packet sequence. A finite set of input and output message types for a specific protocol can be determined according to the protocol specification.

For any protocol program to be analysed, a mapper is required to convert abstract message types into concrete binary messages. The mapper can be implemented by modifying the peer side code that the protocol program interacts with. We use LearnLib, a variant library of the L* algorithm implemented in Java, to infer the Mealy machine of target protocol program. The learning steps are as follows.

1) LearnLib generates a sequence of input message types through L* algorithm.
2) Mapper convert input message types into concrete input packets and sends it to the target protocol program.
3) The target protocol program generates output packets after receiving the input messages.
4) The Mapper convert the output packets into abstract message types.
5) LearnLib updates the states information according to the output message types and generates a new sequence of input message types.
6) Iterate steps 2-5 until the learning goal is completed, that is, the model converges or reaches the set learning time.
7) Random sampling is used to check the consistency of the learned model, and if there is any inconsistency, the model is corrected.

3.2. Path filter

The path filter is optional and its main purpose is to manually screen out the transition paths of interest or sensitivity in the state machine for further analysis. Most paths in the state machine conform to the protocol standard specification. These paths can be directly filtered out by experience.

For a Mealy Machine $A = (S, I, O, \delta, X, s_0)$, there is always an output transition in $X$ corresponding to a state transition in $\delta$. $s_i \xrightarrow{i/o}s_j$ is the syntactic sugar for the pair of state transition $s_i \xrightarrow{i}s_j$ and corresponding output transition $s_i \xrightarrow{o}s_j$. For a termination state $s_x$, the sequence of transition
conditions (input/output) from the initial state $s_0$ to $s_k$ is called a path from $s_0$ to $s_k$. At least one path exists for state $s_k$. For instance, $[i_0/o_0, i_1/o_1, \ldots, i_k/o_k]$ is the path for transformation relations $s_0 \xrightarrow{i_0/o_0} s_1 \xrightarrow{i_1/o_1} \ldots \xrightarrow{i_k/o_k} s_k$.

An idea for filtering suspect paths is presented here. Here is an idea for filtering paths, based on the basic idea that most paths are compliant. For the reason that the protocol specification specifies no standard FSM model, it is not possible to find the suspect paths directly by comparing. Mealy machines between different implementations of the same protocol are cross-verified to get the suspect paths. And in more detail, the set of the same path between them is the normal path ignored, and the different path between them is the suspicious paths.

The formal description is that, $P_i$ and $P_j$ are two different implementations of the same protocol $P$, and Mealy machines that are inferred using the same input and output table and model learning algorithm are denoted as $M_i$ and $M_j$, the set of suspicious paths $SP_i$ in $M_i$ is defined as:

$$SP_i = \text{Paths}_i \setminus (\text{Paths}_i \cap \text{Paths}_j),$$

where $\text{Paths}_i$ and $\text{Paths}_j$ are the set of all paths in $M_i$ and $M_j$ respectively.

### 3.3. Path converter

When analysing a Mealy machine, the specific path is taken as the object. Although the termination state is not defined, but it can be easily observed in the state machine of a protocol implementation. The specific path from the initial state to the termination state, represents an entire session of the protocol. For example, in the Mealy Machine shown in Figure 2, the path highlighted in red can be viewed as a communication between two entity of the protocol. Each element in the sequence is an I/O pair, which can be divided into a message sent by an entity of the protocol and a response message from a peer entity. A path sequence is an abstract definition of the communication behaviours between two entities of a protocol. Figure 3 shows an example of how a path in a Mealy Machine translates into a protocol session.

As mentioned in the inferring Mealy Machine phase, the input and output sets are protocol message types extracted from the protocol specification. The session corresponding to a path can be modeled using Tamarin’s multi-set rewrite rules. Tamarin's rules are used to model the behaviours of transformed session participants in the same way as modeling protocol specifications. Each step in the interaction behaviours of a participating entity is defined as a rule, and the initiator and responder of the protocol have a set of rules respectively.

![Figure 2](image-url)  
Figure 2. An example of a Mealy Machine.

A session can be viewed as two state transition paths like $s_i \xrightarrow{\text{action}_1} s_2 \xrightarrow{\text{action}_2} \ldots \xrightarrow{\text{action}_{n-1}} s_n$, one for each participant of the session. For a session participant $A$, the behaviour of $A$ that receiving or sending a specific message at state $s_i$ to state $s_j$ is denoted as $s_i \xrightarrow{\text{action}} s_j$. The action consists of two basic operations, receiving some messages $m_1$ and sending message $m_2$, denoted as $\text{Recv}(m_1), \text{Send}(m_2)$. For some messages that need to be verified, such as digital signatures, the authenticity of the message $m_3$ is denoted as $\text{Verify}(m_3)$. 
Figure 3. A conversion example of a path in Mealy machine into a protocol session.

For each state transition $s_i \xrightarrow{\text{action}} s_j$, we can model it by a rule in Tamarin. State before transition is recorded in premise section of the rule, and state after the transition is recorded in result section of the rule. Action fact section records all parameters that appear in the rule. For actions, Recv appears in the premise section, Send appears in the result section, and Verify appears in the action section.

For example, a simple one-round interaction:

$A \xrightarrow{m} B$

$A \xleftarrow{h(m)} B$

The state transition paths for the two participants of this protocol is as follows:

$L_A = a_0 \xrightarrow{\text{Send}(m)} a_1 \xrightarrow{\text{Recv}(h(m))} a_2$;

$L_B = b_0 \xrightarrow{\text{Recv}(m), \text{Send}(h(m))} b_1$.

The rule representing transition path from $a_0$ to $a_1$, can be written as follows:

rule $a_0\_a_1$:

- $[ \text{Fr}(-m) ]$  // The initial state is empty; generate a fresh value $m$
- $[ \text{Action}_a0\_a1(-m) ]$  // Label the term used for this transformation
- $[ a1(-m), \text{Out}(-m) ]$  // Switch to state $a1$ and send message $m$

Similarly, the rules for the transition path from $A1$ to $A2$ and the transition path from $B0$ to $B1$, are as follows:

rule $a_1\_a_2$:

- $[ a1(m), \text{In}(h(m)) ]$
- $[ \text{Action}_a1\_a2(m) ]$
- $[ a2(m) ]$

rule $b_0\_b_1$:

- $[ \text{In}(m) ]$
- $[ \text{Action}_b0\_b1(m) ]$
- $[ b1(m), \text{Out}(h(m)) ]$

3.4. Path verification

It is useless to model the obtained session directly, since only the sequence of abstract message types is contained in the session interaction information. It requires detailed modeling of message types.

In the model learning stage, the Mapper program converts the abstract message types into the concrete binary packets, and packets are constructed in strict accordance with the legal message formats in the protocol specifications. Therefore, when modeling a path of the Mealy machine, it is reasonable to refine the abstract message types with reference to valid message formats in the protocol specification without affecting security analysis. Modeling the behaviours of different entities of the protocol, it needs to add the relevant Cryptographic primitives support and lemmas to model the protocol security properties. These can be done by referring to the examples in the Tamarin manual.
4. Experiments

To verify the effectiveness of the proposed method, we apply the proposed method to automate the detection of ChangeCipherSpec error, a typical vulnerability caused by timing error in OpenSSL. The target object of the analysis is OpenSSL1.0.1g, which acts as the server of TLS protocol. The learner based on LearnLib library runs in a virtual machine. The implementation of model learning is based on Paul’s work[15]. The learned Mealy machine is shown in the figure 4. For the convenience of display, the original model is simplified manually.

Figure 4. Learned state machine for OpenSSL1.0.1g.

Then we show how a suspicious path in Mealy Machine is modeled by using Tamarin's multi-set rewriting rules. This path corresponds to [S0-> S1-> S6-> S9-> S12], and the specific content is as follows:

```
[ ClientHelloRSA/ServerHello+Certificate+ServerHelloDone,
  ChangeCipherSpec/empty, ClientKeyExchange/Empty,
  Finished/ ChangeCipherSpec+DecryptError
]
```

When modeling the session, we use the built-in hashing, symmetric-encryption, asymmetric encryption, and signing to model the cryptographic primitives in the TLS protocol. 6 rules are defined, of which for path session modeling and 1 for public key infrastructure modeling. Some more details about these rules are given in figure 5.

Figure 5. Six rules for modeling the session.
Two lemmas are used to model the protocol's reachability and master key (MS)’s confidentiality.

**lemma reachable:**
exists-trace
" Ex C S nc ns sid MS#i.
Client_Done(C,S,nc,ns,sid,MS) @ I "

**lemma session_key_secrecy:**
"not( Ex C S nc ns sid MS#i.
Client_Done(C,S,nc,ns,sid,MS) @ i &
Ex #j. Server_Done(C,S,nc,ns,sid,MS) @ j &
(Ex #r. K(MS) @r) ) "

Figure 6. two lemmas for modeling the properties.

The reachable lemma states that, there is a path to fact Client_Done for protocol termination. Using the auto-proof strategy, Tamarin finds a reachable path that proved the integrity of the input model.

The confidentiality lemma states that, there is no such a situation: fact Client_Done is reached in process i, and fact Server_Done is reached in another process j at the same time, and the attacker knows MS denoted as K(MS). Using the auto-proof strategy, Tamarin denies this statement and report the attack path. See the link for more experimental details. 1

5. Conclusions

To solve the problem of formal verification of security protocol implementations, this paper proposes a verification method that combines model learning with Tamarin, a powerful formal verification tool. The method infers the Mealy machine model of a protocol implementation based on model learning technology, and then transforms the paths in the Mealy machine into a protocol session. Finally, the Tamarin tool is used to model the converted protocol session and automatically verify whether the path violates the protocol security properties. It can be used to automatically discover a typical logic vulnerability in the early version of OpenSSL, which verifies the effectiveness of the method. This method provides a new idea for automatic analysis of protocol state machine model and is more versatile than the existing methods.

References

[1] Blanchet B. (2016) Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif. Foundations and Trends® in Privacy and Security, 1 (1-2):1 - 135.
[2] Cremers C J F. (2008) The Scyther Tool: Verification, falsification, and analysis of security protocols. In: International conference on computer aided verification. Springer, Berlin. pp. 414-418.
[3] Van Schmidt B. (2012) Formal analysis of key exchange protocols and physical protocols[D]. ETH Zurich.
[4] Goubault-Larrecq J, Parrennes F. (2005) Cryptographic protocol analysis on real C code. In: International Workshop on Verification, Model Checking, and Abstract Interpretation. Berlin. pp. 363-379.
[5] Aizatulin M, Gordon A D, Jürjens J. (2011) Extracting and verifying cryptographic models from C protocol code by symbolic execution. In: Proceedings of the 18th ACM conference on Computer and communications security. New York. pp. 331-340.
[6] Zhang HG, Wu FS, Wang HZ, Wang ZY. (2018) A Survey: Security Verification Analysis of Cryptographic Protocols Implementations on Real Code. Chinese Journal of Computers, 41(2): 288-308.

1 https://github.com/85lx/ML-Tamarin/blob/main/TamarinOutput.svg.
[7] Udrea O, Lumezanu C, Foster J S. (2008) Rule-based static analysis of network protocol implementations. Information and Computation, 206(2-4): 130-157.
[8] Chau S Y, Chowdhury O, Hoque E, et al. (2017) Symcerts: Practical symbolic execution for exposing noncompliance in X. 509 certificate validation implementations. In: IEEE Symposium on Security and Privacy (S&P). pp. 503-520.
[9] Vanhoef M, Piessens F. (2018) Symbolic execution of security protocol implementations: handling cryptographic primitives. In: 12th {USENIX} Workshop on Offensive Technologies.
[10] Chau S Y, Yahyazadeh M, Chowdhury O, et al. (2019) Analyzing semantic correctness with symbolic execution: A case study on pkcs# 1 v1. 5 signature verification. In: Network and Distributed Systems Security (NDSS) Symposium.
[11] Angluin D. (1987) Learning regular sets from queries and counterexamples. Information and computation, 1987,75(2): 87-106.
[12] Cho C Y, Babic D, Poosankam P, et al. (2011) MACE: Model-inference-Assisted Concolic Exploration for Protocol and Vulnerability Discovery. In: USENIX Security Symposium. pp. 139.
[13] De Ruiter J, Poll E. (2015) Protocol State Fuzzing of {TLS} Implementations. In: 24th {USENIX} Security Symposium ({USENIX} Security 15). pp.193-206.
[14] Fiterau-Brostean P, Janssen R, Vaandrager F. (2016) Combining model learning and model checking to analyze TCP implementations. In: International Conference on Computer Aided Verification. Cham. pp. 454-471.
[15] Fiterau-Brostean P, et al. (2017) Model learning and model checking of SSH implementations. In: Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software.
[16] Cimatti A, Clarke E, Giunchiglia E, et al. (2002) Nusmv2: An opensource tool for symbolic model checking. In: International Conference on Computer Aided Verification. Berlin. pp. 359-364.
[17] Guo JX, Gu CX, Chen X, et al. (2019) Model learning and model checking of ipsec implementations for internet of things[J]. IEEE Access, 7:171322-171332.
[18] Aichernig B K, Mostowski W, Mousavi M R, et al. (2018) Model learning and model-based testing. Machine Learning for Dynamic Software Analysis: Potentials and Limits,74-100.
[19] Fiterau-Broștean P, Jonsson B, Merget R, et al. (2020) Analysis of DTLS Implementations Using Protocol State Fuzzing. In: 29th {USENIX} Security Symposium ({USENIX} Security 20). pp. 2523-2540.
[20] The Tamarin Team. (2021). Tamarin-Prover Manual, Security Protocol Analysis in the Symbolic Model. https://tamarin-prover.github.io/manual/tex/tamarin-manual.pdf.