A Syntactic Approach for Identifying Multi-Protocol Attacks

Béla Genge and Piroska Haller
"Petru Maior" University of Târgu Mureş
Electrical Engineering Department
N. Iorga St., No. 1, Tg. Mureş, Romania
{bgenge,phaller}@engineering.upm.ro

***Abstract*—In the context of multiple security protocols running in the same environment, we propose a syntactical approach for identifying multi-protocol attacks. The proposed approach uses a canonical security protocol model, where terms that can be verified by protocol participants are denoted by canonical terms. In order to enable the identification of subtle "type-flaw" attacks, where terms can be substituted with other types of terms, we introduce a canonical identifier. The approach is validated by analyzing several security protocol pairs. The attacks discovered by our approach are also discovered by existing security protocol verification tools.**

Keywords: Security Protocols, Multi-Protocol Attacks.

I. Introduction

Security protocols are "communication protocols dedicated to achieving security goals" (C.J.F. Cremers and S. Mauw) [1] such as confidentiality, integrity or availability. Achieving such security goals is made through the use of cryptography. The explosive development of today's Internet and the technological advances made it possible to implement and use security protocols in a wide range of applications such as sensor networks, electronic commerce or routing environments.

Security protocols have been intensively analyzed throughout the last few decades, resulting in a variety of dedicated formal methods and tools [2], [3], [10]. The majority of these methods consider a Dolev-Yao-like intruder model [5], [6] to capture the actions available to a intruder which has complete control over the network. By analyzing each individual protocol in the presence of this penetrator model, the literature has reported numerous types of attacks [7], [3].

However, in practice, there can be multiple protocols running over the same network, thus the intruder is given new opportunities to construct attacks by combining messages from several protocols, also known as multi-protocol attacks [4].

In order to identify these attacks, in the literature we find several approaches. For instance, in [8] Catherine Meadows proposes a "zipper" (i.e. comparison) procedure for identifying "type-flaw" attacks specific to multi-protocol environments. "Type-flaw" attacks explore the lack of knowledge of protocol participants because of which message components can be substituted with others found in other protocols.

Other approaches, such as the one proposed by Guttman [9], tackle with the "protocol independence" property. According to this, multiple protocols maintain their properties when running in the same environment if they use a disjoint encryption. In this approach, the user must identify similar protocol messages based on a protocol specification provided by the authors.

In the last decade, there have been several tools proposed to enable the verification of the correctness of security protocols running in isolation. However, only a few can be used to verify the correctness of multiple protocols running in the same environment. One of the most recent tools capable to identify these attacks is the *Scyther* tool, proposed by Cremers [10]. This tool was originally designed to prove the correctness of security protocols running in isolation. However, by concatenating multiple specifications it can be used to verify the correctness of multiple protocols running in the same environment. The tool uses a state-space exploration approach with an implementation of the Dolev-Yao intruder model. It can also be used to identify "type-flaw" attacks.

In this paper we propose a syntactical approach for identifying multi-protocol attacks. The advantages of using such an approach are multiple. First, if specifications are provided it can be fully automatized, without the need of human intervention, which was the case of approaches proposed by Meadows and Guttman. Second, implementations are much faster than state-space exploration methods, such as the one proposed by Cremers. These advantages make our approach a candidate for an on-line automatic multi-protocol verification tool that could be used in mobile ad-hoc networks, where newly discovered services implementing new security protocols must be automatically executed [16].

In order to introduce this method, we first present a protocol model that includes information related to protocol preconditions, effects, participant knowledge, and the sequence of sent and received messages. Based on this protocol model, we define a canonical model that allows a syntactical analysis of the modeled protocols by eliminating instance-based information through the use of message component *types*. By doing so, we syntactically model the knowledge of protocol participants, used to identify and verify message components.

The method is validated by verifying the correctness of several protocol pairs running in the same environment using the widely-adopted security protocol verification tool Scyther

[10].

The paper is structured as follows. The proposed security protocol and canonical models are constructed in section II. In section III we define a predicate for identifying messages that can be excepted from other protocols and we present our experimental results. We end with a conclusion in section IV.

II. Modeling Security Protocols

A. Protocol Model

Protocol participants communicate by exchanging *terms* constructed from elements belonging to the following basic sets: P, denoting the set of role names; N, denoting the set of random numbers or *nonces* (i.e. "number once used"); K, denoting the set of cryptographic keys; C, denoting the set of certificates and M, denoting the set of user-defined message components.

In order for the protocol model to capture the message component types found in security protocol implementations [11], [12] we specialize the basic sets with the following subsets:

- $\mathsf{P}_{DN} \subseteq \mathsf{P}$, denoting the set of distinguished names; $\mathsf{P}_{UD} \subseteq \mathsf{P}$, denoting the set of user-domain names; $\mathsf{P}_{IP} \subseteq \mathsf{P}$, denoting the set of user-ip names; $\mathsf{P}_U = \{\mathsf{P} \setminus \{\mathsf{P}_{DN} \cup \mathsf{P}_{UD} \cup \mathsf{P}_{IP}\}\}$, denoting the set of names that do not belong to the previous subsets;
- N_T, denoting the set of timestamps; N_{DH}, denoting the set of random numbers specific to the Diffie-Hellman key exchange; $\mathsf{N}_A = \{\mathsf{N} \setminus \{\mathsf{N}_{DH} \cup \mathsf{N}_T\}\}$, denoting the set of random numbers;
- $\mathsf{K}_S \subseteq \mathsf{K}$, denoting the set of symmetric keys; $\mathsf{K}_{DH} \subseteq \mathsf{K}$, denoting the set of keys generated from a Diffie-Hellman key exchange; $\mathsf{K}_{PUB} \subseteq \mathsf{K}$, denoting the set of public keys; $\mathsf{K}_{PRV} \subseteq \mathsf{K}$, denoting the set of private keys;

To denote the encryption type used to create cryptographic terms, we define the following *function names*:

$$\begin{aligned} FuncName ::= &\ sk && (symmetric\, function)\\ |&\ pk && (asymmetric\, function)\\ |&\ h && (hash\, function)\\ |&\ hmac && (keyed\, hash\, function) \end{aligned}$$

The encryption and decryption process makes use of cryptographic keys. Decrypting an encrypted term is only possible if participants are in the possession of the decryption key pair. In case of symmetric cryptography, the decryption key is the same as the encryption key. In case of asymmetric cryptography, there is a public-private key pair. Determining the corresponding key pair is done using the function $\_^{-1} : \mathsf{K} \rightarrow \mathsf{K}$.

The above-defined basic sets and function names are used in the definition of *terms*, where we also introduce constructors for pairing and encryption:

$$\mathsf{T} ::= . \mid \mathsf{R} \mid \mathsf{N} \mid \mathsf{K} \mid \mathsf{C} \mid \mathsf{M} \mid (\mathsf{T},\mathsf{T}) \mid \{\mathsf{T}\}_{FuncName(\mathsf{T})},$$

where the '.' symbol is used to denote an empty term.

Having defined the terms exchanged by participants, we can proceed with the definition of a *node* and a *participant chain*. To capture the sending and receiving of terms, the definition of nodes uses *signed terms*. The occurrence of a term with a positive sign denotes transmission, while the occurrence of a term with a negative sign denotes reception.

Definition 1. *A* node *is any transmission or reception of a term denoted as* $\langle \sigma, t\rangle$*, with* $t \in \mathsf{T}$ *and* σ *one of the symbols* $+, -$*. A node is written as* $-t$ *or* $+t$*. We use* $(\pm\mathsf{T})$ *to denote a set of nodes. Let* $n \in (\pm\mathsf{T})$*, then we define the function* $sign(n)$ *to map the sign and the function* $term(n)$ *to map the term corresponding to a given node.*

Definition 2. *A* participant chain *is a sequence of nodes. We use* $(\pm\mathsf{T})^*$ *to denote the set of finite sequences of nodes and* $\langle \pm t_1, \pm t_2, \ldots, \pm t_i\rangle$ *to denote an element of* $(\pm\mathsf{T})^*$*.*

In order to define a participant model we also need to define the preconditions that must be met such that a participant is able to execute a given protocol. In addition, we also need to define the effects resulting from a participant executing a protocol.

Preconditions and effects are defined using predicates applied on terms: *CON_TERM* : T, denoting a term that must be previously generated (preconditions) or it is generated (effects); *CON_PARTAUTH* : T, denoting a participant that must be previously authenticated (preconditions) or a participant that is authenticated (effects); *CON_CONF* : T, denoting that a given term must be confidential (preconditions) or it is kept confidential (effects); *CON_INTEG* : T, denoting that for a given term the integrity property must be provided (preconditions) or that the protocol ensures the integrity property for the given term (effects); *CON_NONREP* : T, denoting that for a given term the non-repudiation property must be provided (preconditions) or that the protocol ensures the non-repudiation property for the given term (effects); *CON_KEYEX* : T, denoting that a key exchange protocol must be executed before (preconditions) or that this protocol provides a key exchange resulting the given term (effects).

The set of precondition-effect predicates is denoted by $\mathsf{PR\_CC}$ and the set of precondition-effect predicate subsets is denoted by $\mathsf{PR\_CC}^*$. Next, we define predicates for each type of term exchanged by protocol participants. These predicates are based on the basic and specialized sets provided at the beginning of this section. We use the *TYPE_DN* : T predicate to denote distinguished name terms, *TYPE_UD* : T to denote user-domain name terms, $TYPE\_IP$: T to denote user-ip name terms, *TYPE_U* : T user name terms, *TYPE_NT* : T to denote timestamp terms, *TYPE_NDH* : T to denote Diffie-Hellman random number terms, *TYPE_NA* : T to denote other random number terms, *TYPE_NDH* : $\mathsf{T} \times \mathsf{T} \times \mathsf{T} \times \mathsf{P} \times \mathsf{P}$ to denote Diffie-Hellman symmetric key terms $(term, number_1, number_2, \quad participant_1, participant_2)$, *TYPE_KSYM* : $\mathsf{T} \times \mathsf{P} \times \mathsf{P}$ to denote symmetric key terms $(term, participant_1, participant_2)$, *TYPE_KPUB* : $\mathsf{T} \times \mathsf{P}$ to denote public key terms $(term, participant)$, *TYPE_KPRV* : $\mathsf{T} \times \mathsf{P}$ to denote private key terms $(term, participant)$, *TYPE_CERT* : $\mathsf{T} \times \mathsf{P}$ do denote certificate terms $(term, participant)$ and *TYPE_MSG* : T to denote user-defined terms.

The set of type predicates is denoted by PR_TYPE and the set of type predicate subsets is denoted by PR_TYPE*. Based on the defined sets and predicates we are now ready to define the participant and protocol models.

Definition 3. *A* participant model *is a tuple* $\langle prec, eff, type, gen, part, chain\rangle$*, where* $prec \in$ PR_CC* *is a set of precondition predicates,* $eff \in$ PR_CC* *is a set of effect predicates,* $type \in$ PR_TYPE *is a set of type predicates,* $gen \in \mathsf{T}^*$ *is a set of generated terms,* $part \in \mathsf{P}$ *is a participant name and* $chain \in (\pm\mathsf{T})^*$ *is a participant chain. We use the* MPART *symbol to denote the set of all participant models.*

Definition 4. *A* protocol model *is a collection of participant models such that for each positive node* n_1 *there is exactly one negative node* n_2 *with* $term(n_1) = term(n_2)$*. We use the* MPROT *symbol to denote the set of all protocol models.*

B. Canonical Protocol Model

In order to identify attacks, we construct a canonical model that focuses on terms that can be verified by protocol participants. For each term from the protocol model, defined in the previous section, the canonical model provides a corresponding syntactical representation through the use of *basic types*. These denote the terms that can be verified by protocol participants also including a representation for terms that can not be verified because of limited participant knowledge. The verification process makes use of these types to decide if attacks can be constructed on each protocol model by using terms extracted from the other considered protocol models.

The *basic types* we consider are based on the specialized basic sets introduced in the protocol model:

$$BasicType ::= \mathsf{p}_{DN} \mid \mathsf{p}_{UD} \mid \mathsf{p}_{IP} \mid \mathsf{p}_{U} \mid \mathsf{n}_{T} \mid \mathsf{n}_{DH} \mid \mathsf{n}_{A} \mid \mathcal{K} \mid \mathsf{m} \mid \mathsf{c} \mid \mathsf{u},$$

where the given symbols correspond to participant distinguished names, user-domain names, user-ip names, other user names, timestamps, Diffie-Hellman random numbers, other random numbers, keys, user defined terms, certificates and unknown terms, respectively.

In the encryption process of the same plaintext, the use of two different keys, K_1 and K_2, will produce two different ciphertexts. This is also true for the decryption process, where the use of two different keys results in two different plaintexts. Because of this, we consider that the type of the encrypted terms after decryption will change too, according to the keys that are used. Thus we use an indexed key type k_i, such that $\mathsf{k}_i \neq \mathsf{k}_j$, where $i \neq j$, to distinguish between key types corresponding to different keys. In the definition of *BasicType*, the set of all typed keys is denoted by $\mathcal{K}$. We define the $\_^{-1} : \mathcal{K} \rightarrow \mathcal{K}$ function to map the canonical key pair corresponding to a given canonical key.

The *unknown* type u corresponds to terms that can not be validated because of limited role knowledge. By including this information in the specification we are able to detect subtle type-flaw attacks using a syntactical comparison of typed terms, that otherwise would require the construction of a state-space that can become rather large if we consider the existence of multiple protocols in the same system.

Based on the defined basic terms we can now proceed with the definition of *canonical terms* that makes use of the previously defined function names:

$$\mathcal{T} ::= . \mid BasicType \mid (\mathcal{T}, \mathcal{T}) \mid \{\mathcal{T}\}_{FuncName(\mathcal{T})}.$$

A canonical node is defined as a signed canonical term using the following definition.

Definition 5. *A* canonical node *is any transmission or reception of a canonical term denoted as* $\langle\sigma, t\rangle$*, with* $t \in \mathcal{T}$ *and* σ *one of the symbols* $+, -$*. We use* $(\pm\mathcal{T})$ *to denote a set of canonical nodes. Let* $n \in (\pm\mathcal{T})$*, then we define the function* $csign(n)$ *to map the sign and the function* $cterm(n)$ *to map the canonical term corresponding to a given canonical node.*

Before we proceed with the definition of canonical chains and canonical participant models we need to define *classifiers*. These are attached to participant chains and are used to transform canonical terms received from other participants based on local participant knowledge. We define two such classifiers:

$$Classifier ::= CL_P \mid CL_V.$$

The first classifier CL_P denotes the processing chain corresponding to a participant. This chain contains canonical terms that correspond to participant knowledge. The second classifier CL_V denotes the virtual chain used to transform received terms from the transmitted form to the received form based on the knowledge of the receiving participant.

Definition 6. *A* canonical participant chain *is a sequence of canonical nodes. A* classified canonical participant chain *is a pair* $\langle CL, l_{cc}\rangle$*, where* $CL \in Classifier$ *and* $l_{cc} \in (\pm\mathcal{T})^*$*. We use* $(\pm\mathcal{T})^*$ *to denote a set of canonical participant chains.*

Definition 7. *A* canonical participant model *is a pair* $\langle part, sl_{cc}\rangle$*, where* $part \in \mathsf{P}$ *is a participant name and* $sl_{cc} \in (Classifier \times (\pm\mathcal{T})^*)^*$ *is a set of classified canonical participant chains. We use* MPART-C *to denote the set of all canonical participant models.*

Next, we define a canonical protocol model as a set of canonical participant models.

Definition 8. *A* canonical protocol model *is a collection of canonical participant models such that for each positive canonical node* n_1 *there is exactly one negative canonical node* n_2 *with* $cterm(n_1) = cterm(n_2)$*. We use the* MPROT-C *symbol to denote the set of all canonical protocol models.*

III. Identifying Multi-Protocol Attacks

A. Mathematical Constructions

Multi-protocol attacks are possible to create by intruders if messages from protocols are accepted as valid in other protocols.

In order to identify these attacks, we use the *CONSTR* predicate that expresses the fact that a given canonical term can be constructed by instantiation from another canonical term.

Informally, we consider that this construction is possible if canonical terms from the same position are equal or that the second term contains an undefined canonical term.

The *CONSTR* : $\mathcal{T} \times \mathcal{T}$ predicate is defined as:

$$CONSTR(t,t') = \begin{cases} True, & \text{if } t = t' \vee (t \in BasicType \wedge t' = \mathsf{u}) \\ & \quad \vee(t = \mathsf{u} \wedge t' \in BasicType), \\ CONSTR(t_1,t_1')\wedge & \text{if } (t = (t_1,t_2) \wedge t' = (t_1',t_2'))\vee \\ \;CONSTR(t_2,t_2'), & \quad t = \{t_1\}_{f(t_2)} \wedge t' = \{t_1'\}_{f(t_2')}\wedge \\ & \quad (t_2 = t_2' \vee t_2' = \mathsf{u}), \\ False, & \text{otherwise.} \end{cases}$$

B. Experimental Results

In order to validate our approach we used existing protocol verification tools. The purpose of the verification was to determine if new attacks become available when other protocols are also present and if these attacks are also discovered by our approach. One of the few tools allowing the verification of multi-protocol attacks is Scyther [10], which is the only tool currently available that also detects type-flaw attacks [13], commonly found in multi-protocol environments.

We have applied our method to several pairs of security protocols defined in the library maintained by Clark and Jacob [14], for which there is also an online version available [15]. Through our experiments we have verified the correctness of protocol pairs such as Yahalom-Lowe and Kao-Chow, Lowe-Needham-Schroeder and ISO9798, Lowe-Denning-Sacco and Lowe-Wide-Mouthed-Frog, Andrew-Secure-RPC and CCITT X.509, Denning-Sacco and Otway-Reese.

By applying the *CONSTR* predicate we discovered several new multi-protocol attacks. For example, in case of the protocol pair Yahalom-Lowe and Kao-Chow, a new attack was discovered that gave the intruder the possibility to replay valid messages from the Kao-Chow protocol in the Yahalom-Lowe protocol. We have created a composed protocol and used the Scyther tool to verify it. The result was that 2 new attacks were possible. After correcting the problem, the Scyther tool did not detect any attacks, which was also confirmed by our method.

IV. Conclusion

We have developed a new method for the automated identification of multi-protocol attacks. The novelty of our approach is the fact that it provides a syntactical verification of the involved protocols, that makes it appropriate for on-line applications.

Our proposal makes use of an enriched protocol model that embodies protocol preconditions and effects. Messages exchanged by participants are modeled as sequences of nodes called participant chains. Based on these, we constructed the protocol model. In order to identify multi-protocol attacks we constructed a canonical model that outlines the knowledge available to protocol participants when running the protocols. Using the constructed canonical terms, we are able to detect message components that can be replaced by others, thus allowing intruders to construct multi-protocol attacks.

TABLE I
Protocol composition results

Protocol 1	Protocol 2	Proposed approach	Scyther
Lowe-B	ISO9798	No Attack	No Attack
Lowe-B	X509v1	No Attack	No Attack
ISO9798	X509v1	No Attack	No Attack
ISO9798	X509v1c	No Attack	No Attack
X509v1	X509v1c	No Attack	No Attack
X509v1	X509v1c	No Attack	No Attack
BAN-RPC	Lowe-B	Found Attack	Found Attack
L-D-S	K-Cv1	Found Attack	Found Attack
K-Cv1	K-Cv2	No Attack	No Attack
L-D-S	Kerbv5	Found Attack	Found Attack
Lowe-Kerb	Neuman-S	Found Attack	Found Attack
H-N-S	Neuman-S	No Attack	No Attack
Needh-S	X509v1	No Attack	No Attack
L-N-S	ISO9798	No Attack	No Attack
Otway-R	Lowe-B	No Attack	No Attack
SPLICE	Needh-S	No Attack	No Attack
TMN	Andr-RPC	No Attack	No Attack
Y-L	K-Cv1	Found Attack	Found Attack

In order to validate our approach, we applied it on several pairs of security protocols. The results have been confirmed by using an existing well-established protocol verification tool: Scyther. The advantage of using our proposal is that by using a syntactical approach, analysis can be executed in real time, as opposed to Scyther, which is based on a state-space exploration method. As a disadvantage, we can note that our approach does not identify the effects of the discovered attack. However, in case protocols can not be modified, the decision on discovering a multi-protocol attack can be to simply not execute the given protocol, in which case the effects of the discovered attack is of no relevance.

References

[1] C. Cremers, S. Mauw, "Checking secrecy by means of partial order reduction", In S. Leue and T. Systa, editors, Germany, september 7-12, 2003, revised selected papers LNCS, Vol. 3466, 2005, Springer.

[2] F. J. T. Fabrega, J. C. Herzog, J. D. Guttman, 11Strand spaces: Proving security protocols correct", *Journal of Computer Security*, 7: 191–230, 1999.

[3] C. Weidenbach, 11Towards an automatic analysis of security protocols", *Lecture Notes in Artificial Intelligence* 1632: 378–382, 1999.

[4] Cas J. F. Cremers, 11Compositionality of Security Protocols: A Research Agenda", Electr. Notes Theor. Comput. Sci., 142, pp. 99–110, 2006.

[5] D. Dolev, A.C. Yao, "On the security of public key protocols", *IEEE Transactions on Information Theory*, 29: 198–208, 1983.

[6] I. Cervesato, "The Dolev-Yao Intruder is the Most Powerful Attacker", 16th Annual Symposium on Logic in Computer Science, LICS'01, IEEE Computer Society Press, Boston, MA, 2001.

[7] Gavin Lowe, "Some new attacks upon security protocols", In Proceedings of the 9th Computer Security Foundations Workshop, IEEE Computer Society Press, 1996, pp. 162–169.

[8] C. Meadows, "A Procedure for Verifying Security Against Type Confusion Attacks", In the Proc. of the 16th CSFW, 2003, p. 62.

[9] J.D. Guttman, F.J.T. Fabrega, "Protocol Independence through Disjoint Encryption", In the Proc. of the 13th CSFW, 2000, pp. 24–34.

[10] C.J.F. Cremers, "Scyther", Semantics and Verification of Security Protocols, Thesis, University Press Eindhoven, 2006.

[11] *SAML V2.0 OASIS Standard Specification*, Organization for the Advancement of Structured Information Standards, http://saml.xml.org/, 2007.

[12] *OASIS Web Services Security (WSS)*, Organization for the Advancement of Structured Information Standards, http://saml.xml.org/, 2006.

[13] J. Heather, G. Lowe, S. Schneider, "How to Prevent Type Flaw Attacks on Security Protocols", In the Proc. of the 13th Computer Security Foundations Workshop, IEEE Computer Society Press, July 2000.

[14] J. Clark, J. Jacob, "A Survey of Authentication Protocol Literature: Version 1.0", York University, 17 November 1997.

[15] Laboratoire Specification et Verification, Security Protocol Open Repository, http:// www.lsv.ens-cachan.fr/spore/, 2008.

[16] Genge Bela, Haller Piroska, "Middleware for Automated Implementation of Security Protocols", 6th European Semantic Web Conference, Heraklion, Greece, 31 May - 4 June, Lecture Notes in Computer Science (LNCS 5554), L. Aroyo et al. (Eds.), Springer-Verlag, pp. 476-490, 2009.