

Preventing Type Flaw Attacks on Security Protocols With a Simplified Tagging Scheme

YAFEN LI, WUU YANG and CHING-WEI HUANG
National Chiao-Tung University, Hsin-Chu, Taiwan, R.O.C.^{1 2}

abstract. A *type flaw attack* on a security protocol is an attack where a field in a message that was originally intended to have one type is subsequently interpreted as having another type. Heather et al. proves that type flaw attacks can be prevented with the technique of tagging each field with the information that indicates its intended type. We simplify Heather et al.'s tagging scheme by combining all the tags inside each encrypted component into a single tag and by omitting the tags on the outmost level. The simplification reduces the sizes of messages in the security protocol. We also formally prove our simplified tagging scheme is as secure as Heather et al.'s with the strand space method. Note that Heather et al.'s tagging scheme and our simplified tagging are applicable to, not just one protocol, but a variety of security protocols.
keywords: network security, security protocol, type flaw, strand space, tagging

1 Introduction

A *type flaw attack* on a security protocol is an attack where a field of a message that was originally intended to have one type is subsequently interpreted as having another type [3]. For example, consider the Neuman-Stubblebine protocol[7]. (A pair of curly brackets with a suitable superscript and/or a subscript means encryption. Items in the same components are separated with commas.)

Initial exchange

Msg 1. $A \rightarrow B : A, N_a$

Msg 2. $B \rightarrow S : B, \{A, N_a, T_b\}_{Shared(B,S)}, N_b$

Msg 3. $S \rightarrow A : \{B, N_a, K_{ab}, T_b\}_{Shared(A,S)}, \{A, K_{ab}, T_b\}_{Shared(B,S)}, N_b$

Msg 4. $A \rightarrow B : \{A, K_{ab}, T_b\}_{Shared(B,S)}, \{N_b\}_{k_{ab}}$

Subsequent authentication

Msg 5. $A \rightarrow B : N'_a, \{A, K_{ab}, T_b\}_{Shared(B,S)}$

Msg 6. $B \rightarrow A : N'_b, \{N'_a\}_{K_{ab}}$

Msg 7. $A \rightarrow B : \{N'_b\}_{K_{ab}}$

In [1], Carlsen describes a type flaw attack on the Neuman-Stubblebine protocol. The attack is shown below, where P_x denotes the penetrator which masquerades as the principal x .

Msg 1. $P_a \rightarrow B : A, N_P$

Msg 2. $B \rightarrow P_s : B, \{A, N_P, T_b\}_{Shared(B,S)}, N_b$

Msg 4. $P_a \rightarrow B : \{A, N_P, T_b\}_{Shared(B,S)}, \{N_b\}_{N_P}$

¹The work reported here is supported by National Science Council, Taiwan, R.O.C., under grant NSC 92-2213-E-009-070

²The complete version of this paper, including all the proofs, is available on the internet at <http://www.cis.nctu.edu.tw/wuuyang/papers/PreventTypeFlaw.pdf>

In the attack, the two penetrators P_a and P_s (which could possibly be the same attacker) collaborate to cheat B . B will decode message 4 with the secret key $Shared(B, S)$ shared by B and the real server S to obtain N_P , which B is fooled to believe to be the secret session key between B and A . Once the protocol for the initial exchange is compromised, subsequent authentication can be attacked in a trivial manner.

Heather et al. [3] proves that type flaw attacks can be prevented with the technique of tagging each field with its intended type. For example, message 4 would then become:

Msg 4 (Heather et al.’s tagging scheme). $A \rightarrow B : (\{|agent, key, timestamp|\}^{shared}, \{(agent, A), (key, K_{ab}), (timestamp, T_b)\}_{Shared(B,S)}^{shared}, (\{|nonce|\}^{shared}, \{(nonce, N_b)\}_{K_{ab}}^{shared})$

Here, $\{|agent, key, timestamp|\}^{shared}$, $agent, key, timestamp$, $\{|nonce|\}^{shared}$, and $nonce$ are all added tags.

In this paper, we simplify the tagging scheme by combining all the tags inside each encrypted component into a single tag and by omitting the tags on the outmost level. For example, message 4 would then become:

Msg 4 (Our simplified tagging scheme). $A \rightarrow B : (\{(agent, key, timestamp), (A, K_{ab}, T_b)\}_{Shared(B,S)}^{shared}, \{(nonce, N_b)\}_{K_{ab}}^{shared})$

Note that, in our simplified tagging scheme, the outmost-level tags— $\{|agent, key, timestamp|\}^{shared}$ and $\{|nonce|\}^{shared}$ —are omitted. Furthermore, the three tags inside the encrypted component— $agent, key$ and $timestamp$ —are combined into a single tag— $(agent, key, timestamp)$. Because in a security protocol, the kinds of all possible tags are very limited, the combined tag can be represented with a single, small integer.

Following Heather-Lowe-Schneider’s proof method [3] (which will be referred to as the *HLS-scheme* in this paper), we also proved that our simplified tagging scheme is as secure as the HLS-scheme. The proofs proceed in two stages. In the first stage, we first define an *A-scheme* in which all tags inside each encrypted component are combined into a single tag. We show that the A-scheme is as secure as the HLS-scheme. In the second stage, the A-scheme is further simplified. We define a *B-scheme* in which the outmost-level tags are omitted in addition to the simplifications made in the A-scheme. We then show that the B-scheme is as secure as the A-scheme. We then conclude that the B-scheme, our simplified tagging scheme, is as secure as Heather et al.’s (full) tagging scheme.

Since we adopt the same model and proof techniques of [2, 3], the definitions in this paper (in Sections 3, 4, and 5) are adapted from the above references, with necessary modifications to reflect our simplified tagging scheme. We provide these definitions in order to facilitate the reader to understand our proofs given later.

Attacks based on type flaws are quite common in security protocols. Meadows [5] also discusses a similar type flaw attack on the Needham-Schroder protocol [6]. The Woo-Lam protocol π_1 [9] is also subject to a type flaw attack [3].

The next two subsections introduce the Neuman-Stubblebine protocol [7] under Heather et al.’s and our B tagging schemes. Section 2 defines the A-scheme and shows that the A-scheme is as secure as the HLS-scheme. In Section 3, we define the B-scheme and show that the B-scheme is as secure as the A-scheme. Section 4 concludes this paper.

1.1 The Neuman-Stubblebine Protocol Under the HLS Tagging Scheme

Heather-Lowe-Schneider’s Tagging Scheme was introduced in [3]. Because the A-scheme and the B-scheme, which are adapted from and similar to the HLS-scheme, will be defined later

in this paper, we will present the HLS-scheme informally with an example in this section.

Heather, Lowe, and Schneider [3] proves that type flaw attacks can be prevented with the technique of tagging each field with the information that indicates its intended type. Here is the modified Neuman-Stubblebine Protocol with the tags added:

Initial exchange

Msg 1. $A \rightarrow B : (agent, A), (nonce, N_a)$

Msg 2. $B \rightarrow S : (agent, B), (\{|nonce, timestamp|\}^{shared}, \{(nonce, N_a), (timestamp, T_b)\}_{Shared(B,S)}^{shared}, (nonce, N_b)$

Msg 3. $S \rightarrow A : (\{|agent, nonce, shared, timestamp|\}^{shared}, \{(agent, B), (nonce, N_a), (shared, K_{ab}), (timestamp, T_b)\}_{Shared(A,S)}^{shared}, (\{|agent, shared, timestamp|\}^{shared}, \{(agent, A), (shared, K_{ab}), (timestamp, T_b)\}_{Shared(B,S)}^{shared}, (nonce, N_b)$

Msg 4. $A \rightarrow B : (\{|(agent, shared, timestamp)|\}^{shared}, \{(agent, A), (shared, K_{ab}), (timestamp, T_b)\}_{Shared(B,S)}^{shared}, (\{|nonce|\}^{shared}, \{(nonce, N_b)\}_{k_{ab}}^{shared})$

Subsequent authentication

Msg 5. $A \rightarrow B : (nonce, N'_a), (\{|(agent, shared, timestamp)|\}^{shared}, \{(agent, A), (shared, K_{ab}), (timestamp, T_b)\}_{Shared(B,S)}^{shared})$

Msg 6. $B \rightarrow A : (nonce, N'_b), (\{|nonce|\}^{shared}, \{(nonce, N'_a)\}_{K_{ab}}^{shared})$

Msg 7. $A \rightarrow B : (\{|nonce|\}^{shared}, \{(nonce, N'_b)\}_{K_{ab}}^{shared})$

The above tagged protocol, though looked complicated, is obtained simply by adding a tag before every item. Removing these tags will result in the original Neuman-Stubblebine protocol.

1.2 The Neuman-Stubblebine Protocol Under the B-scheme

For ease of comparison, we include the Neuman-Stubblebine protocol under the B-scheme in this subsection.

Initial exchange

Msg 1. $A \rightarrow B : A, N_a$

Msg 2. $B \rightarrow S : B, \{(agent, nonce, timestamp), (A, N_a, T_b)\}_{Shared(B,S)}^{shared}, N_b$

Msg 3. $S \rightarrow A : \{(agent, nonce, shared, timestamp), (B, N_a, K_{ab}, T_b)\}_{Shared(A,S)}^{shared}, \{(agent, shared, timestamp), (A, K_{ab}, T_b)\}_{Shared(B,S)}^{shared}, N_b$

Msg 4. $A \rightarrow B : \{(agent, shared, timestamp), (A, K_{ab}, T_b)\}_{Shared(B,S)}^{shared}, \{(nonce, N_b)\}_{k_{ab}}^{shared}$

Subsequent authentication

Msg 5. $A \rightarrow B : N'_a, \{(agent, shared, timestamp), (A, K_{ab}, T_b)\}_{Shared(B,S)}^{shared}$

Msg 6. $B \rightarrow A : N'_b, \{(nonce, N'_a)\}_{K_{ab}}^{shared}$

Msg 7. $A \rightarrow B : \{(nonce, N'_b)\}_{K_{ab}}^{shared}$

2 The A-Scheme

In this subsection, we present the model that will be used to prove the first point—a protocol under our simplified tagging scheme (but with the outmost-level tags) is as secure as that under Heather et al.'s (full) tagging scheme. That is, if there is an attack upon a protocol under the A-scheme, then there must exist a corresponding attack under the HLS-scheme. This model is based on the strand space model of [2, 3, 7].

Our proof proceeds as follows. First we model the abilities of a penetrator with the *penetrator strands* (to be discussed later in this section) in the A-scheme. Then we show that every penetrator strand in the A-scheme corresponds to a penetrator strand in the HLS-scheme. Hence, the A-scheme would not introduce any new security attacks that are not present in the HLS-scheme.

Tags As in [3], we assume that atomic values are partitioned into types, including agent, nonce, time-stamp, public key, etc., and we will adopt the obvious names for each tag. The types of tags can be defined by:

$$Tag ::= agent \mid nonce \mid timestamp \mid public \mid \dots \mid \{|Tag^*|\}^{Tag}$$

Tagged Facts We represent the tagged facts as $(Tags, Facts)$ pairs, where the tags give the claimed types of the corresponding facts.

$$Fact ::= Atom \mid \{TaggedFact\}_{Fact}^{Tag}$$

$$TaggedFact ::= Tag^* \times Fact^*$$

Strand Templates As in [3], we use the strand templates to define roles in a protocol. A strand template is defined as follows:

$$StrandTemplate ::= (Sign \times TaggedTemplate)^*$$

$$Sign ::= + \mid -$$

$$TaggedTemplate ::= Tag^* \times Template^*$$

$$Template ::= Var \mid Fn(Var^*) \mid \{TaggedTemplate\}_{Template}^{Tag}$$

Definition. An *honest strand* is one that results from the application of a substitution to a strand template.

Honest Strand Assumption. If the simple tagged fact (t, f) originates on an honest strand, then $TopLevelWellTagged(t, f)$.

Penetrator Traces Following [1, 7], we assume that there are some set T of simple facts that the penetrator can produce and some set K_p of keys that the penetrator has available. Penetrator traces under the tagging scheme are exactly analogous to those in [3], but with the modifications to the M , C , S and R traces.

A penetrator trace is one of the following M , F , T , C , S , K , E , D , and R strands.

M Text message $\langle +(t, f) \rangle$ for $f \in T$.

The penetrator spontaneously generates a text message from the simple fact available to him/her. Note that the M strand only produces simple tagged facts and non-simple tagged facts can be produced by concatenation.

F Flushing $\langle -(ts, fs) \rangle$.

T Tee $\langle -(ts, fs), +(ts, fs), +(ts, fs) \rangle$.

C Concatenation $\langle -(ts_1, fs_1), \dots, -(ts_k, fs_k), +(ts_1, \dots, ts_k, fs_1, \dots, fs_k) \rangle$.

S Separation $\langle -(t_1, \dots, t_k, f_1, \dots, f_k), +(t_1, f_1), \dots, +(t_k, f_k) \rangle$.

K Key $\langle +(tk, k) \rangle$ with $WellTagged(tk, k)$ and $k \in K_p$.

E Encryption $\langle -(tk, k), -(ts, fs), +(\{|ts|\}^{tk}, \{(ts, fs)\}_k^{tk}) \rangle$.

D Decryption $\langle -(tk', k'), -(\{|ts|\}^{tk}, \{(ts, fs)\}_k^{tk}), +(ts, fs) \rangle$, where tk and tk' are tags representing inverse key types and k' is the decryption key corresponding to k when they are considered as keys of types tk' and tk , respectively.

R Retagging $\langle -(t, f), +(t', f) \rangle$

Note that the R strand only applies to simple tagged facts. Non-simple tagged facts can be retagged by separation, retagging particular components, and then concatenation. All other penetrator traces do not interfere with the tags of their messages.

We may transform arbitrary bundles (presumably the penetrator bundles) into *well-tagged* bundles. We will show that, given a bundle C in the A-scheme, we can construct a corresponding bundle C' in the HLS-scheme in which all terms are well-tagged.

Now we can prove our first result—if there is a type flaw attack on a protocol under A-scheme, there is a type flaw attack under HLS-scheme. We will discuss secrecy and authentication separately.

Theorem (Secrecy). Let $temp$ be the strand template for some role in the A-scheme. Let (t, v) be a tagged variable. Let h be a positive integer. Let $Keys$ be a set of function templates. Let C be a bundle in the A-scheme and C' be the well-tagged bundle obtained by transforming C . (Note that C' is in the HLS-scheme.) If there is a failure of secrecy in C , there will be a failure of secrecy in C' as well.

Theorem (Authentication). Let $temp1$ and $temp2$ be the templates for two roles in the A-scheme. Let X be the set of variables in the templates. Let $h1$ and $h2$ be two positive integers. Let $Keys$ be a set of function templates. Let C be a bundle in the A-scheme and C' be a well-tagged bundle in the HLS-scheme obtained by transforming C . If there is a failure of authentication in C , there will be a failure of authentication in C' as well.

3 The B-Scheme

We may further transform a security protocol in the A-scheme into a corresponding protocol in the B-scheme. The transformation is very similar to the one presented in the previous section and is hence omitted for the sake of brevity.

We may prove a result for the B-scheme similar to that for the A-scheme. Our result is that, if there is a type flaw attack on a protocol under the B-scheme, there is a type flaw attack under the A-scheme. We will discuss secrecy and authentication separately.

Theorem (Secrecy). Let $temp$ be a template in the B-scheme and $temp'$ be the template in the A-scheme for the same role. Let (t, v) be a tagged variable. Let h be a positive integer. Let $Keys$ be a set of function templates. Let C be a bundle in the B-scheme and C' be a tagged bundle (in the A-scheme) obtained by transforming C . If there is a failure of secrecy in C , there will a failure of secrecy in C' as well.

Theorem (Authentication). Let $temp1$ and $temp2$ be the templates for two roles in the B-scheme. Let $temp1'$ and $temp2'$ be the templates for the same roles in the A-scheme. Let X be the set of variables in the templates. Let $h1$ and $h2$ be two positive integers. Let $Keys$ be a set of function templates. Let C be a bundle in the B-scheme and C' be a tagged

bundle (in the A-scheme) obtained by transforming C . If there is a failure of authentication in C , there will be a failure of authentication in C' as well.

4 Conclusion

Heather, Lowe, and Schneider show that adding tags to fields in security messages can prevent type flaw attacks. We further simplified their scheme by combining and omitting certain tags. The main contribution of our work is the insight that an attacker *cannot* fake the outmost-level tags. The faked outmost-level tags will be detected by the real participants in a security protocol. All type flaw attacks, if they ever occur, must be related to fields inside an encrypted message in a security protocol.

Verification of security protocols is similar to verification of computer programs but the two carry complementing emphases. While we hope a computer program does what it is intended for, which is explicitly prescribed in the program's text, the most important properties of a security protocol is that it contains no *hidden* holes or weaknesses. These holes and weaknesses are not explicitly mentioned or even imagined in the description of the behavior of a security protocol.

References

- [1] U. Carlsen, "Cryptographic protocol flaws: know your enemy," In Proceedings of Computer Security Foundations Workshop VII, 192-200, June 1994.
- [2] F.J.T. Fábrega, J.C. Herzog, and J.D. Guttman, "Strand spaces: why is a security protocol correct?" In Proceedings of 1998 IEEE Symposium on Security and Privacy, 160-171, May 1998.
- [3] J. Heather, G. Lowe, and S. Schneider, "How to prevent type flaw attacks on security protocols," In Proceedings of 13th IEEE Computer Security Foundations Workshop, 255-268, 2000.
- [4] G. Lowe, "A hierarchy of authentication specifications," In Proceedings of 10th Computer Security Foundations Workshop, 31-43, June 1997.
- [5] C.A. Meadows, "Analyzing the Needham-Schroder public-key protocol: A comparison of two approaches," In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, ed. *ESORICS'96*, LNCS 1146, 351-346, 1996.
- [6] R. Needham and M. Schroeder, "Using encryption for authentication in large networks of computers," *Comm. ACM*, 21(12), 993-999, 1978.
- [7] B.C. Neuman and S.G. Stubblebine, "A note on the use of timestamps as nonces," *ACM Operating Systems Reviews*, 27(2), 10-14, April 1993.
- [8] L.C. Paulson, "Proving security protocols correct," In Proceedings of 14th Symposium on Logic in Computer Science, 370-381, 1999.
- [9] T.Y.C. Woo and S.S. Lam, "A lesson on authentication protocol design," *ACM Operating Systems Reviews*, 28(3), 24-37, 1994.