# Detecting and Resolving Type Flaws in Security Protocols

Michael Banks

Department of Computer Science, University of York

26th February 2009

# Outline

# What is a Security Protocol?

> *A security protocol comprises a prescribed sequence of interactions between entities [designed to provide] security services across a distributed system.*
>
> *(Ryan and Schneider, 2001)*

Principals (entities) interact by sending and receiving sets of terms:

| | |
|---|---|
| Principal IDs | $A$, $B$, $S$ |
| Secret Keys | $K_{AS}$, $K_{BS}$, $K_{AB}$ |
| Nonces | $N_A$, $N_B$ |
| Timestamps | $T_A$, $T_B$, $T_S$ |

Terms may be encrypted to ensure their secrecy or integrity.

# The Otway-Rees Key Transport Protocol

Key distribution problem (informal description):

- $A$ and $B$ wish to communicate with each other in private.
- $S$ is a key server, trusted by $A$ and $B$ to generate session keys.

Otway and Rees (1987) proposed a key transport protocol:

1. $A \rightarrow B : M, A, B, \{N_A, M, A, B\}_{K_{AS}}$
2. $B \rightarrow S : M, A, B, \{N_A, M, A, B\}_{K_{AS}}, \{N_B, M, A, B\}_{K_{BS}}$
3. $S \rightarrow B : M, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}}$
4. $B \rightarrow A : M, \{N_A, K_{AB}\}_{K_{AS}}$

# Outline

# A Type Flaw in the Otway-Rees Protocol (Boyd, 1990)

Recall the Otway-Rees protocol:

1. $A \rightarrow B : M, A, B, \{N_A, M, A, B\}_{K_{AS}}$
2. $B \rightarrow S : M, A, B, \{N_A, M, A, B\}_{K_{AS}}, \{N_B, M, A, B\}_{K_{BS}}$
3. $S \rightarrow B : M, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}}$
4. $B \rightarrow A : M, \{N_A, K_{AB}\}_{K_{AS}}$

Let's assume that $|K_{AB}| = |M| + |A| + |B|$.

An adversary $Z$ may intercept (1) and impersonate $B$ in (4):

1. $A \rightarrow Z(B) : M, A, B, \{N_A, M, A, B\}_{K_{AS}}$
4. $Z(B) \rightarrow A : M, \{N_A, M, A, B\}_{K_{AS}}$

Upon receiving message (4), $A$ may interpret $\langle M, A, B \rangle$ as $K_{AB}$!

# Why Are Type Flaw Attacks Possible?

> *A type flaw attack on a security protocol is an attack where a field that was originally intended to have one type is subsequently interpreted as having another type.*
>
> *(Heather, Lowe, and Schneider, 2000)*

Type flaws are implementation-dependent: (Carlsen, 1994)

- A type flaw may be exploited if term widths coincide in the protocol implementation.
- Belief logics do not consider implementation details! (Burrows et al., 1989; Syverson and van Oorschot, 1996)

# Preventing Type Flaws by Adding Contextual Information

Record the intended type of each term in a packet in a "type tag":

1. $A \rightarrow B : M, A, B, \{\langle nonce, sid, pid, pid \rangle, N_A, M, A, B\}_{K_{AS}}$
2. $B \rightarrow S : M, A, B, \{\langle \cdots \rangle, N_A, M, A, B\}_{K_{AS}}, \{\langle \cdots \rangle, N_B, M, A, B\}_{K_{BS}}$
3. $S \rightarrow B : M, \{\langle nonce, key \rangle, N_A, K_{AB}\}_{K_{AS}}, \{\langle nonce, key \rangle, N_B, K_{AB}\}_{K_{BS}}$
4. $B \rightarrow A : M, \{\langle nonce, key \rangle, N_A, K_{AB}\}_{K_{AS}}$

Potential drawbacks:

- Recipients need to check that packets are tagged correctly.
- Tagging all packets is sometimes unnecessary (Carlsen, 1994) and may allow cryptanalysis. (Mao and Boyd, 1994)
- Meadows (2002, 2003) argues that tagging may not prevent complex type flaw attacks which confuse tags with terms.

Possible optimisation: type tag indexes. (Heather et al., 2000)

# Preventing Type Flaws by Reordering Terms

Change the order of terms in individual packets:

1. $A \rightarrow B : M, A, B, \{N_A, M, A, B\}_{K_{AS}}$
2. $B \rightarrow S : M, A, B, \{N_A, M, A, B\}_{K_{AS}}, \{N_B, M, A, B\}_{K_{BS}}$
3. $S \rightarrow B : M, \{K_{AB}, N_A\}_{K_{AS}}, \{K_{AB}, N_B\}_{K_{BS}}$
4. $B \rightarrow A : M, \{K_{AB}, N_A\}_{K_{AS}}$

**The Good** No extra communication or processing overheads.

**The Bad** Not a general solution: for some protocols, no term reordering eliminates all type flaws.

**The Ugly** Considered an *ad hoc* method by Boyd and Mathuria (2003); few references to reordering in the literature.

# Reordering Terms Cannot Resolve All Type Flaws

Consider the Andrew Secure RPC protocol (Satyanarayanan, 1989):

$$
\begin{array}{ll}
1. & A \rightarrow B : A, \{N_A\}_{K_{AB}} \\
2. & B \rightarrow A : \{N_A + 1, N_B\}_{K_{AB}} \\
3. & A \rightarrow B : \{N_B + 1\}_{K_{AB}} \\
4. & B \rightarrow A : \{K'_{AB}, N'_B\}_{K_{AB}}
\end{array}
$$

If $|N_A| + |N_B| = |K'_{AB}| + |N'_B|$, then $Z$ may mislead $A$ in (4):

$$
\begin{array}{lllll}
1. & A & \rightarrow B & : & A, \{N_A\}_{K_{AB}} \\
2. & B & \rightarrow A & : & \{N_A + 1, N_B\}_{K_{AB}} \\
3. & A & \rightarrow Z(B) & : & \{N_B + 1\}_{K_{AB}} \\
4. & Z(B) & \rightarrow A & : & \{N_A + 1, N_B\}_{K_{AB}}
\end{array}
$$

This protocol also contains a freshness flaw in (4).
(Burrows et al., 1989; Clark and Jacob, 1997)

# Naïvely Reordering Terms May Introduce New Type Flaws

Consider the Neuman and Stubblebine (1993) initial exchange:

1. $A \rightarrow B : A, N_A$
2. $B \rightarrow S : B, \{A, N_A, T_B\}_{K_{BS}}, N_B$
3. $S \rightarrow A : \{B, N_A, K_{AB}, T_B\}_{K_{AS}}, \{A, K_{AB}, T_B\}_{K_{BS}}, N_B$
4. $A \rightarrow B : \{A, K_{AB}, T_B\}_{K_{BS}}, \{N_B\}_{K_{AB}}$

If $|N_A| = |K_{AB}|$, then $Z$ may masquerade as $A$ and $S$ to mislead $B$:

1. $Z(A) \rightarrow B \quad : A, N_A$
2. $B \quad \rightarrow Z(S) : B, \{A, N_A, T_B\}_{K_{BS}}, N_B$
4. $Z(A) \rightarrow B \quad : \{A, N_A, T_B\}_{K_{BS}}, \{N_B\}_{N_A}$

This flaw was discovered independently by Syverson (1993) and Hwang et al. (1995); the protocol is also vulnerable to a parallel session attack. (Hwang et al., 1995; Clark and Jacob, 1997)

# Naïvely Reordering Terms May Introduce New Type Flaws

Is the "permuted protocol" (Syverson, 1993) free of type flaws?

1. $A \rightarrow B : A, N_A$
2. $B \rightarrow S : B, \{A, T_B, N_A\}_{K_{BS}}, N_B$
3. $S \rightarrow A : \{B, N_A, K_{AB}, T_B\}_{K_{AS}}, \{A, K_{AB}, T_B\}_{K_{BS}}, N_B$
4. $A \rightarrow B : \{A, K_{AB}, T_B\}_{K_{BS}}, \{N_B\}_{K_{AB}}$

Hint: what if $|N_A| = |K_{AB}| = 2 \times |T_B|$ ?

# Naïvely Reordering Terms May Introduce New Type Flaws

Is the "permuted protocol" (Syverson, 1993) free of type flaws?

1. $A \rightarrow B : A, N_A$
2. $B \rightarrow S : B, \{A, T_B, N_A\}_{K_{BS}}, N_B$
3. $S \rightarrow A : \{B, N_A, K_{AB}, T_B\}_{K_{AS}}, \{A, K_{AB}, T_B\}_{K_{BS}}, N_B$
4. $A \rightarrow B : \{A, K_{AB}, T_B\}_{K_{BS}}, \{N_B\}_{K_{AB}}$

Hint: what if $|N_A| = |K_{AB}| = 2 \times |T_B|$ ?

Assuming that $Z$ knows a recent timestamp $T'_B \simeq T_B$:

1. $Z(A) \rightarrow B : A, \langle X, T'_B \rangle$
2. $B \rightarrow Z(S) : B, \{A, T_B, \langle X, T'_B \rangle\}_{K_{BS}}, N_B$
4. $Z(A) \rightarrow B : \{A, T_B, \langle X, T'_B \rangle\}_{K_{BS}}, \{N_B\}_{\langle T_B, X \rangle}$

$B$ may interpret $\langle T_B, X \rangle$ as $K_{AB}$. (Syverson, 1993)

# Classifying Type Flaws

We assume the Dolev and Yao (1983) threat model for adversary.

## Class I

An adversary exploits a type flaw by substituting one encrypted packet for another encrypted packet.

## Class II

An adversary induces a type flaw by changing plaintext terms, or by substituting a plaintext term for an encrypted packet.

Note: this classification is original and provisional!

# Outline

# Extracting Terms from Messages

Let $rec_n$ denote the recipient of $msg_n$.

Let readable($msg_n$) denote the terms in $msg_n$ which $rec_n$ can read:

$$\begin{aligned}
\text{readable}(msg_1) &= \{M, A, B\} && (rec_1 = B)\\
\text{readable}(msg_2) &= \{M, A, B, N_A, N_B\} && (rec_2 = S)\\
\text{readable}(msg_3) &= \{M, K_{AB}, N_B\} && (rec_3 = B)\\
\text{readable}(msg_4) &= \{M, K_{AB}, N_A\} && (rec_4 = A)
\end{aligned}$$

# Constructing Principal Knowledge

Each principal $P$ "knows" an initial set of terms, initial($P$):

$$\begin{aligned}\text{initial}(A) &= \{M, A, B, S, N_A, K_{AS}\}\\ \text{initial}(B) &= \{B, S, N_B, K_{BS}\}\\ \text{initial}(S) &= \{A, B, S, K_{AS}, K_{BS}\}\end{aligned}$$

Let $\text{knows}_n(P)$ denote $P$'s knowledge after $msg_n$ has been received:

$$\text{knows}_0(P) = \text{initial}(P)$$

$$\text{knows}_n(P) = \text{knows}_{n-1} \cup \begin{cases}\text{readable}(msg_n) & \text{if} \quad P = rec_n\\ \emptyset & \text{otherwise}\end{cases}$$

# Applying Principal Knowledge

Let packets($msg_n$) denote the set of packets $p_k$ in $msg_n$ for which $rec_n$ knows the key $k$ to decrypt $p_k$:

$$\text{packets}(msg_n) = \{p_k \mid p_k \in msg_n \wedge k \in \text{knows}_n(rec_n)\}$$

We can identify the terms in each packet which $rec_n$ recognises from previous knowledge:

$$\forall p_k \in \text{packets}(msg_n) \bullet \text{checkables}_n(p_k) = p_k \cap \text{knows}_{n-1}(rec_n)$$

$$\text{checkables}_2(\{N_A, M, A, B\}_{K_{AS}}) = \{A, B\}$$
$$\text{checkables}_2(\{N_B, M, A, B\}_{K_{BS}}) = \{A, B\}$$
$$\text{checkables}_3(\{N_B, K_{AB}\}_{K_{BS}}) = \{N_B\}$$
$$\text{checkables}_4(\{N_A, K_{AB}\}_{K_{AS}}) = \{N_A\}$$

# Constructing Packet Templates

Treating each $p_k$ as a sequence of terms, $\text{template}_n(p_k)$ replaces each subsequence of non-checkable terms in $p_k$ with a wildcard:

$$\begin{aligned}
\text{template}_2(\{N_A, M, A, B\}_{K_{AS}}) &= \langle \star, A, B \rangle_{K_{AS}} \\
\text{template}_2(\{N_B, M, A, B\}_{K_{BS}}) &= \langle \star, A, B \rangle_{K_{BS}} \\
\text{template}_3(\{N_B, K_{AB}\}_{K_{BS}}) &= \langle N_B, \star \rangle_{K_{BS}} \\
\text{template}_4(\{N_A, K_{AB}\}_{K_{AS}}) &= \langle N_A, \star \rangle_{K_{AS}}
\end{aligned}$$

For each $p_k$ in $\text{packets}(msg_n)$, we require $rec_n$ to verify that all $\text{checkables}_n(p_k)$ are located at their respective offsets in $p_k$.

# Finding (Class I) Type Flaws by Template Matching

Will the recipient of $msg_n$, expecting to find $p_k$, accept $q_{k'}$ instead?

To find out, we check whether $\text{template}_n(p_k)$ matches any encrypted packet $q_{k'}$ $(\neq p_k)$ in messages $1..n$:

$$\text{flaws}_n(p_k) = \left\{ q_{k'} \;\middle|\; \begin{array}{ll} & m \in 1..n \wedge q_{k'} \in \text{packets}(msg_m) \\ \wedge & q_{k'} \neq p_k \wedge k' = k \\ \wedge & q_{k'} \text{ matches } \text{template}_n(p_k) \end{array} \right\}$$

$$\text{flaws}_3(\{N_B, K_{AB}\}_{K_{BS}}) = \{\{N_B, M, A, B\}_{K_{BS}}\}$$
$$\text{flaws}_4(\{N_A, K_{AB}\}_{K_{AS}}) = \{\{N_A, M, A, B\}_{K_{AS}}\}$$

# Resolving Potential Type Flaws

1. Generate a (lazy) list of reordered protocols, corresponding to the Cartesian product of all packet permutations.

$$\begin{aligned} \text{Otway-Rees:} \quad & 4! \times 4! \times 2! \times 2! = 2\,304 \\ \text{Neuman-Stubblebine:} \quad & 3! \times 4! \times 3! \times 1! = 864 \end{aligned}$$

2. Exhaustive search: apply type flaw detection procedure to each reordered protocol in list, until

   either a protocol without any type flaws is found;
   or all reordered protocols have been tested.

# Implementation

Type flaw detection procedure implemented in Haskell.

- Separate detectors for class I and II type flaws.
- Reports a safe protocol reordering if one is found.

Results:

- Finds class I flaws in all known-flawed protocols tested.
- Finds class II flaws in some (not all!) flawed protocols.
- Performance: $<$ 1s to detect and resolve flaws. (GHCi)

# Outline

# Open Questions

1. Does the detection procedure find all potential type flaws?
   - Is the classification of type flaws adequate?
   - Constraints on term widths rule out some reported flaws.
2. Can reordering packets introduce other implementation-dependent protocol flaws?
   - eg. cut-and-paste attacks (Mao and Boyd, 1994)
   - Protocol efficiency is important, but it may be prudent to accept the overheads of introducing contextual information.

# Final Remarks

- Reordering terms is a viable and efficient method for resolving type flaws in many — but not all — security protocols.
- We have exhibited a systematic procedure for finding and resolving potential class I type flaws.

## Any Questions?