Casimiro Greco

Analysis of Attacks to Multi-Protocols

mWSF05

The 2005 miniWorkshop on Security Frameworks

Catania, 16 Dicembre 2005
Many formal methods in last years have been developed

These analysis supposed that protocols run in isolation
Is it realistic to assume that one protocol is the only protocol on the network?
Can protocols interact?

Could this interaction be used by an intruder?
Use different key structures

Not always a good idea:

1) Key distribution/management problems

2) Limited Resources (smartcards)
An attack that involves more than one protocol sharing network and key structures is called multi-protocol attack

The intruder uses messages from different instances of different protocols
Kelsey, Schneier, Wagner

Given a correct security protocol, there exists another correct security protocol, such that their composition is incorrect.

- Chosen Protocol
- Target Protocol
Lowe’s Version of Needham-Schroeder

Message 1. \( A \rightarrow S : A.S.B \)
Message 2. \( S \rightarrow A : S.A.\{PK(B),B\}_{PKS(S)} \)
Message 3. \( A \rightarrow B : A.B.\{N_a.A\}_{PK(B)} \)
Message 4. \( B \rightarrow S : B.S.A \)
Message 5. \( S \rightarrow B : S.B.\{PK(A),A\}_{PKS(S)} \)
Message 7. \( A \rightarrow B : A.B.\{N_b\}_{PK(B)} \)
Message i.  \( B \rightarrow A : B.A.\{M.N_b.B\}_{PK(A)} \)

Message ii.  \( A \rightarrow B : A.B.\{N_b.B\}_{PKS(A)} \)

Message I) has the same structure of 6)
CHOSEN PROTOCOL ATTACK

target  chosen

Message 3. $E_A \rightarrow B : A.B. \{N_a.A\}_{PK(B)}$

Message ii. $A \rightarrow E_B : A.B. \{N_b.B\}_{PKS(A)}$

Message 7. $E_A \rightarrow B : A.B. \{N_b\}_{PK(B)}$

B authenticates $E_A$ as $A$
Guessing Attack: attack where an attacker guesses a poorly chosen secret and then seeks to verify that guess using other information.

Multi-Protocol Guessing Attack: information comes from messages of other protocols
1) The passwords being guessed have low entropy

2) The verification of a guess does not need on-line interaction with other parties

Failed attempts are undetectable
EKE (Encrypted Key Exchange)

Msg 1. \[a \rightarrow b : \{p_{ka}\}_{\text{passwd}(a,b)}\]

Msg 2. \[b \rightarrow a : \{\{k\}_{p_{ka}}\}_{\text{passwd}(a,b)}\]

Msg 3. \[a \rightarrow s : \{n_{a}\}_{k}\]

Msg 4. \[s \rightarrow a : \{(n_{a}, n_{b})\}_{k}\]

Msg 5. \[a \rightarrow s : \{n_{b}\}_{k}\]

GONG

Msg 1. \[a \rightarrow b : \{x_{1}, n\}_{k1}\]

Msg 2. \[b \rightarrow a : \{f(n)\}_{\text{passwd}(a)}\]
ATTACK

EKE (Encrypted Key Exchange)

Msg 1. \( a \rightarrow b: \{pk_a\}_{\text{passwd}(a,b)} \)
Msg 2. \( b \rightarrow a: \{ \{k\}_{pk_a} \}_{\text{passwd}(a,b)} \)
Msg 3. \( a \rightarrow s: \{n_a\}_k \)
Msg 4. \( s \rightarrow a: \{(n_a,n_b)\}_k \)
Msg 5. \( a \rightarrow s: \{n_b\}_k \)

1. Guess \( \text{passwd}(a,b) \) from Msg 1 to obtain \( pk_a \)

2. Guess \( \text{passwd}(a) \) from Msg ii. to obtain \( f(n) \)

3. Learn \( n \) from \( f(n) \), encode it with \( pk_a \)

4. Compare this value with Msg i

If the values coincide, **attack takes place.**
For all correct protocols, there exists a protocol attack such that the composition contains a security flaw.

What happens with composition of actual protocols?

If all the protocols have been designed according to the guidelines, executing them in parallel will not introduce any new attack.
Analysis and tests of Multi-protocol attacks by Cas Cremers

Analyzed two and three concurrent protocols from Clark and Jacob library, SPORE library and works of Boyd and Mathuria

Analyzed 30 different protocols
Protocols have been tested on three properties:

1) Secrecy

2) Agreement

3) Synchronisation
Most agents cannot verify the values received

When an agent expects e.g. a Nonce and accepts:

1) Only constants of type Nonce → **No Type Flaws**

2) Any simple constants → **Basic Type Flaws**

3) Any terms → **Full Type Flaws**
RESULTS

No Type Flaws allowed:
38 Multi-protocols attacks

Basic Type Flaws allowed:
41 Multi-protocols attacks

Full Type Flaws allowed:
83 Multi-protocols attacks
protocol Woo and Lam Mutual Authentication

\[ K(I, S) \]

\[ I \]

nonce \( n_i \)

\[ I, n_i \]

\[ K(R, S) \]

\[ R \]

nonce \( n_r \)

\[ R, n_r \]

\[ \{I, R, n_i, n_r\}_{K(I,S)} \]

\[ \{I, R, n_i, n_r\}_{K(I,S)}, \{n_i, n_r\}_k \]

\[ \{n_r\}_k \]

\[ \{R, n_i, n_r, k\}_{K(I,S)}, \{I, n_i, n_r, k\}_{K(R,S)} \]

\[ K(I, S), K(R, S) \]

\[ S \]

key \( k \)

\[ key \ k \]

\[ \{R, n_i, n_r, k\}_{K(I,S)}, \{I, n_i, n_r, k\}_{K(R,S)} \]

\[ \{n_r\}_k \]

\[ \{n_i, n_r\}_k \]

\[ \{I, R, n_i, n_r\}_{K(R,S)} \]

\[ \{I, R, n_i, n_r\}_{K(I,S)} \]

secret \( k \)

\[ secret \ k \]

\[ secret \ k \]
trace Attack on Woo-Lam and Yahalom-Lowe

\[ K(a,s) \]

Woo-Lam
\[ a: \text{role } I \]

nonce \( n1 \)

\[ a, n1 \]

Intruder

Yahalom-Lowe
\[ a: \text{role } I \]

nonce \( n2 \)

\[ a, n2 \]

\[ \{a, a, n1, n2\}_{K(a,s)} \]

\[ a, n2 \]

learn \( n1, n2 \)

\[ \{a, a, n1, n2\}_{K(a,s)} \]

Woo-Lam
\[ s: \text{role } S \]

\[ \{a, a, n1, n2\}_{K(a,s)}, \{a, a, n1, n2\}_{K(s,s)} \]

key \( k \)

\[ \{a, n1, n2, k\}_{K(s,s)}, \{a, n1, n2, k\}_{K(s,s)} \]

\[ \{a, a, n1, n2, k\}_{K(s,s)}, \{a, n1, n2, k\}_{K(s,s)} \]

\[ a, n1, n2, k \] type confusion

\[ \{a, a, s, k\}_{n1} \]

learn \( k \)

\[ \{a, n1, n2, k\}_{K(s,s)}, \{n1, n2\}_k \]

secret \( k \)

secret \( n1 \)
An agent $a$ starts the Woo-Lam protocol in the $I$ role and sends a fresh nonce $n1$. The agent starts a Yahalom-Lowe session in parallel, in the $I$ role. $a$ creates and sends $n2$. The two nonces are intercepted by the intruder.

The intruder sends the nonce $n2$ to $a$ in the Woo-Lam protocol, as if it was sent by a Woo-Lam responder role. The agent responds with a server request with the names of both the agents and the nonces.
The last message sent from the agent is intercepted by the intruder, concatenated with itself and sent to the Woo-Lam server S.

The server generates a fresh session key $k$ and sends back two identical messages. One of this is redirected to the Yahalom-Love $I$ role.
I is waiting a message in the form \( \{a, \text{Key}, n2, \text{Nonce}\}_{K(a,s)} \)

the agent can’t distinguish the difference between a Session Key and a Nonce (Basic Type Flaw)

The intruder knows \( n1 \). He can decrypt the message and learn the session key \( k \)
so he accepts \( \{a, n1, n2, k\}_{K(a,s)} \) and sends \( \{a, a, s, k\}_{n1} \) encoding the session key \( k \) with \( n1 \)

sending the message \( \{a, n1, n2, k\}_{K(a,s)}, \{n1, n2\}_k \) the intruder completes the attack
**EN PASSANT...**

trace Attack on Denning-Sacco shared key, Yahalom-Lowe, and Yahalom BAN

$K(a, s)$

**Denning-Sacco**

a: role A

$\alpha, \beta$

key $ke$

timestamp $t$

$K(b, s)$

**Yahalom BAN**

b: role B

nonce $n1$

$K(b, s)$

**Yahalom-Lowe**

b: role B

$\alpha, t$

nonce $n0$

$K(a, s), K(b, s)$

**Yahalom-Lowe**

s: role S

$\{a, t, n0\}_{K(b, s)}$

$\{b, k, t, n0\}_{K(s, b)}$

$\{a, k, e\}_{K(b, s)}$

$\{a, b, s, n0\}_{ke}$

$\{a, k, e\}_{K(b, s)}$

secret $ke$
Developed at the ECSS group of the Technical University of Eindhoven as a part of the PhD research performed by Cas Cremers

Scyther rev. 1410
Security protocols are analyzed with two different techniques:

1) Finite state model checker

2) Backward symbolic state search
   Use the Arachne engine, based on the Athena method. It supports tickets and type flaws
Usage: scyther [switches] [-|FILE] [-o FILE]

--help
print help and exit

--version
print version information and exit. It shows the Subversion revision number, and whether or not Scyther was built with debugging support.
- If the filename is set to '-', input is read from stdin.
- **-o, --output=FILE**
  output file (default is stdout)
- **-e, --empty**
  do not generate output
- **-P, --proof**
  generate proof output in ASCII
- **-C, --class**
  Do not instantiate variables that the intruder can instantiate at will, and leaves them visibly as variables.
- **-S, --summary**
  show summary on stdout instead of stderr
Algorithms and checks

-M, --modelchecker
use ModelChecker

-a, --arachne
use ArachneEngine

-m, --match=[int]
MatchingMethod (default is 0)
0: Typed matching
1: Allow for basic typeflaws
2: Detect all typeflaw attacks
Pruning of the searches (Bounds)

- \texttt{p, --prune=[int]}
  Pruning method (default is 2)
  0: Explore all traces.
  1: Do not explore traces which have more than one security violation.
  2: Once an attack is found, only scan shorter traces.

- \texttt{l, --max-length=[int]}
  prune traces longer than [int] events

- \texttt{r, --max-runs=[int]}
  create at most [int] runs. If [int] is zero, the Arachne method can perform an unbounded search.
  For the Modelchecker, this means that the number of runs is either this number, or the number of runs defined in the input file, whichever is smaller.
  For the Arachne (theorem proving) method, this is simply the maximum number of runs involved in the proof.

- \texttt{--max-attacks=[int]}
  stop exploring the state space after finding [int] attacks.
usertype Server, SessionKey, TimeStamp, TicketKey;
usertype ExpiredTimeStamp;

secret k: Function;

const a, b, e: Agent;
const s: Server;
const Fresh: Function;
const ne: Nonce;
const kee: SessionKey;
untrusted e;
INPUT: Hwang modified version of Neumann Stubblebine

protocol neustub-Hwang(I,R,S)
{
    role I
    {
        const Ni,Mi: Nonce;
        var Nr,Mr: Nonce;
        var T: Ticket;
        var Tb: TimeStamp;
        var Kir: SessionKey;

        send_1(I,R, I, Ni);
        read_3(S,I, { R,Ni,Kir,Tb}k(I,S), T, Nr);
        send_4(I,R,T,{Nr}Kir);
        send_5(I,R,Mi,T);
        read_6(R,I,Mr,{Mr}Kir);
        send_7(I,R,{Mr}Kir);

        claim_I1(I,Secret, Kir);
        claim_I2(I,Niagree);
        claim_I3(I,Nisynch);
        claim_I4(I,Empty,(Fresh,Kir));
    }
}
INPUT: Hwang modified version of Neumann Stubblebine

role R
{
  var Ni,Mi: Nonce;
  const Nr,Mr: Nonce;
  var Kir: SessionKey;
  const Tb: TimeStamp;
  var T: Ticket;

  read_1(l,R,l,Ni);
  send_2(R,S,R,{l,Ni,Tb,Nr}k(R,S));
  read_4(l,R,{l,Kir,Tb}k(R,S),{Nr}Kir);
  read_5(l,R,Mr,T);
  send_6(R,l,Mr,{Mr}Kir);
  read_7(l,R,{Mr}Kir);

  claim_R1(R,Secret,Kir);
  claim_R2(R,Niagree);
  claim_R3(R,Nisynch);
  claim_R4(R,Empty,(Fresh,Kir));
}
INPUT: Hwang modified version of Neumann Stubblebine

role S
{
  var Ni, Nr: Nonce;
  const Kir: SessionKey;
  var Tb: TimeStamp;
  read_2(R, S, R, {I, Ni, Tb, Nr}k(R, S));
  send_3(S, I, { R, Ni, Kir, Tb}k(I, S), { I, Kir, Tb}k(R, S), Nr );
}
secret k : Function;
usertype SessionKey;
const Fresh: Function;
protocol yahalom(I,R,S)
{
  role I
  {
    const Ni: Nonce;
    var Nr: Nonce;
    var T: Ticket;
    var Kir: SessionKey;
    send_1(I,R, I,Ni);
    read_3(S,I, {R,Kir,Ni,Nr}k(I,S), T );
    send_4(I,R, T, {Nr}Kir );
    claim_I1(I, Secret,Kir);
    claim_I2(I, Nisynch);
    claim_I3(I, Empty, (Fresh,Kir));
  }
}

role R
{
  const Nr: Nonce;
  var Ni: Nonce;
  var T: Ticket;
  var Kir: SessionKey;
  read_1(I,R, I,Ni);
  send_2(R,S, R, {I,Ni,Nr}k(R,S ));
  read_4(I,R, {I,Kir}k(R,S) , {Nr}Kir );
  claim_R1(R, Secret,Kir);
  claim_R2(R, Nisynch);
  claim_R3(R, Empty, (Fresh,Kir));
}

role S
{
  const Kir: SessionKey;
  var Ni,Nr: Nonce;
  read_2(R,S, R, {I,Ni,Nr}k(R,S ));
  send_3(S,I, {R,Kir,Ni,Nr}k(I,S), {I,Kir}k(R,S ));
}
Using --latex
digraph semiState1 {
  label = "[id 1] Protocol neustub-Hwang, role R, claim type Nisynch";
  r0i0 [shape=box,label="READ_1(Alice,(Alice,Ni#1 ))"];
  s0 [label="Run 0: neustub-Hwang, R\nR:Bob ((\!Alice, S:Simon)",
shape=diamond];
  s0 -> r0i0;
  r0i1 [shape=box,label="SEND_2(Simon,
(Bob,(Alice,Ni#1,Tb#0,Nr#0))k(Bob,Simon))"];
  r0i0 -> r0i1 [style="bold", weight="10.0"];  
  r0i2
  [shape=box,label="READ_4(Alice,\{(Alice,Kir#2,Tb#0)k(Bob,Simon),(Nr#0)K\{(Bob,Simon)\} )\}"
  ];
  r0i1 -> r0i2 [style="bold", weight="10.0"];  
  r0i3 [shape=box,label="READ_5(Alice,\{(Nonce#0,Ticket#0) \}"
  ];
  r0i2 -> r0i3 [style="bold", weight="10.0"];  
  r0i4 [shape=box,label="SEND_6(Alice, \{(M\{r#0\})K\{r#2 \})\}"
  ];
  r0i3 -> r0i4 [style="bold", weight="10.0"];  
  r0i5 [shape=box,label="READ_7(Alice,\{(Mr#0)K\{r#2 \})\}"
  ];
  r0i4 -> r0i5 [style="bold", weight="10.0"];  
  r0i6 [shape=box,label="CLAIM_R1 ( Secret, Kir#2 )"
  ];
  r0i5 -> r0i6 [style="bold", weight="10.0"];  
  r0i7 [shape=box,label="CLAIM_R2 ( Niagree, * )"
  ];
  r0i6 -> r0i7 [style="bold", weight="10.0"];  
  r0i8
  [style=filled,fillcolor=mistyrose,color=salmon,shape=doubleoctagon,label="CLAIM_R3(Nisynch, * )"
  ];
  r0i7 -> r0i8 [style="bold", weight="10.0"];  
  r1i0 [shape=box,label="SEND_1(Bob, \{(Alice,Ni#1) \})"
  ];
  s1 [label="Run 1: neustub-Hwang, I\!n!l:Alice (R:Bob, S:Simon)",
shape=diamond];
  s1 -> r1i0;
  r1i1 [shape=box,label="READ_3(Simon,\{(Bob,Ni#1,Kir#2,Tb#0)k(Alice,Simon),(Ticket#1,Nr#0) \})"
  ];
  r1i0 -> r1i1 [style="bold", weight="10.0"];  
  r1i2 [shape=box,label="SEND_4(Bob, (Ticket#1,\{(Nr#0)K\{Kir#2 \})\}"
  ];
  r1i1 -> r1i2 [style="bold", weight="10.0"];  
  r1i0 [shape=box,label="READ_2(Bob,\{(Bob,Ni#1,Tb#0,0,Nr#0)k(Bob,Simon)) \}"
  ];
  s2 [label="Run 2: neustub-Hwang, S\!n\!S:Simon ((\!Alice, R:Bob)" ,shape=diamond];
  s2 -> r2i0;
  r2i1 [shape=box,label="SEND_3(Alice,\{(Bob,Ni#1,Kir#2,Tb#0)k(Alice,Simon),(Alice,Kir#2,Tb#0)k(Bob,Simon),(Nr#0) \})"
  ];
  r2i0 -> r2i1 [style="bold", weight="10.0"];  
  r1i0 -> r0i0 [color=forestgreen];  
  r1i2 -> r0i2 [label="construct",color=red];  
  r2i1 -> r1i1 [label="construct",color=red];  
  r0i1 -> r2i0 [color=forestgreen];
  { rank = same; r1i0; }  // rank 0
  { rank = same; r0i0; }  // rank 1
  { rank = same; r0i1; }  // rank 2
  { rank = same; r2i0; }  // rank 3
  { rank = same; r2i1; }  // rank 4
  { rank = same; r1i1; }  // rank 5
  { rank = same; r1i2; }  // rank 6
  { rank = same; r0i2; }  // rank 7
  { rank = same; r0i3; }  // rank 8
  { rank = same; r0i4; }  // rank 9
  { rank = same; r0i5; }  // rank 10
  { rank = same; r0i6; }  // rank 11
  { rank = same; r0i7; }  // rank 12
  { rank = same; r0i8; }  // rank 13

GRAPHVIZ OUTPUT
<table>
<thead>
<tr>
<th>states</th>
<th>281</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>9.000e-02</td>
</tr>
<tr>
<td>st/sec</td>
<td>3.122e+03</td>
</tr>
<tr>
<td>claim</td>
<td>neustub-Hwang</td>
</tr>
<tr>
<td>claim</td>
<td>neustub-Hwang</td>
</tr>
<tr>
<td>claim</td>
<td>neustub-Hwang</td>
</tr>
<tr>
<td>complete_proof</td>
<td></td>
</tr>
<tr>
<td>claim</td>
<td>neustub-Hwang</td>
</tr>
<tr>
<td>claim</td>
<td>neustub-Hwang</td>
</tr>
<tr>
<td>claim</td>
<td>neustub-Hwang</td>
</tr>
<tr>
<td>complete_proof</td>
<td></td>
</tr>
<tr>
<td>claim</td>
<td>yahalom</td>
</tr>
<tr>
<td>claim</td>
<td>yahalom</td>
</tr>
<tr>
<td>claim</td>
<td>yahalom</td>
</tr>
<tr>
<td>claim</td>
<td>yahalom</td>
</tr>
<tr>
<td>Protocol name</td>
<td>Multi-protocol attack</td>
</tr>
<tr>
<td>-------------------------------------------------------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>Bilateral Key Exchange</td>
<td>yes</td>
</tr>
<tr>
<td>Boyd key agreement</td>
<td>yes</td>
</tr>
<tr>
<td>Denning-Sacco shared key</td>
<td>yes</td>
</tr>
<tr>
<td>Gong (nonce based)</td>
<td>yes</td>
</tr>
<tr>
<td>Gong (nonce based, version 2)</td>
<td>yes</td>
</tr>
<tr>
<td>ISO ccITT 509 (BAN version)</td>
<td>yes</td>
</tr>
<tr>
<td>ISO IEC 11770 2-13</td>
<td>yes</td>
</tr>
<tr>
<td>Kao-Chow</td>
<td>yes</td>
</tr>
<tr>
<td>Kao-Chow (version 2)</td>
<td>yes</td>
</tr>
<tr>
<td>Kao-Chow (version 3)</td>
<td>yes</td>
</tr>
<tr>
<td>KSL (based on Kerberos)</td>
<td>yes</td>
</tr>
<tr>
<td>Needham-Schroeder mutual authentication</td>
<td>yes</td>
</tr>
<tr>
<td>Needham-Schroeder-Lowe mutual authentication</td>
<td>yes</td>
</tr>
<tr>
<td>Needham-Schroeder symmetric</td>
<td>yes</td>
</tr>
<tr>
<td>Needham-Schroeder symmetric (amended)</td>
<td>yes</td>
</tr>
<tr>
<td>Otway-Rees</td>
<td>no*</td>
</tr>
<tr>
<td>SOPH Secret Out, Public Home</td>
<td>yes</td>
</tr>
<tr>
<td>Splice-AS</td>
<td>no*</td>
</tr>
<tr>
<td>Splice-AS Hwang and Chen modified</td>
<td>no*</td>
</tr>
<tr>
<td>Splice-AS Hwang and Chen modified (Clark Jacob)</td>
<td>yes</td>
</tr>
<tr>
<td>TMN</td>
<td>no*</td>
</tr>
<tr>
<td>Wide Mouthed Frog (Brutus version)</td>
<td>yes</td>
</tr>
<tr>
<td>Woo and Lam phi (unilateral authentication)</td>
<td>yes</td>
</tr>
<tr>
<td>Woo-Lam mutual authentication</td>
<td>yes</td>
</tr>
<tr>
<td>Yahalom</td>
<td>yes</td>
</tr>
<tr>
<td>Yahalom (BAN)</td>
<td>yes</td>
</tr>
<tr>
<td>Yahalom Lowe modified</td>
<td>yes</td>
</tr>
<tr>
<td>Yahalom Paulson strengthened</td>
<td>yes</td>
</tr>
</tbody>
</table>

(*) There are attacks for these protocols when running in isolation: multi-protocol attacks do not introduce any new attacks on claims that were correct in isolation.
Explicitness

If all protocols are **consistently tagged**, multi-protocols attacks cannot occur.

**Tagging:**

Instead of \( \{\ldots\}^k \) add a tag within the encryption.

- \( \{\text{“woo-lam”},\ldots\}^k \)
- \( \{\text{“yahalom”},\ldots\}^k \)

**Not always possible**
MULTI-PROTOCOLS ATTACK ARE A REAL PROBLEM!!!

We must be cautious with the deployment of “probably correct” protocols.

Ambiguos authentication can easily cause problems and is likely to occur.

Analyze the interactions of different protocols in the same network is very important.