ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

S. Mödersheim
Ch. Sprenger
AS 2009

Formal Methods for Information Security

Exercise Sheet 2

Hand-in date: Oct 12, 2009

Note: In the latest version of OFMC, the default for the typed model has changed: by default OFMC checks protocols in a typed model, and to obtain type-flaw attacks, one has to explicitly specify the option `-untyped`.

Assignment 2.1: Breaking and Fixing Kerberos-PKInit

Consider the following protocol from module 2:

$$\begin{array}{ll} C \to KAS: & Cert_C, \{T_C, N_2\}_{\mathsf{inv}(\mathsf{sigk}(C))} \\ KAS \to C: & \{Cert_{KAS}, \{K, N_2\}_{\mathsf{inv}(\mathsf{sigk}(KAS))}\}_{\mathsf{pk}(C)} \end{array}$$

We have two roles here, C and KAS. Every role R of the protocol has two asymmetric key-pairs, namely $(\mathsf{sigk}(R), \mathsf{inv}(\mathsf{sigk}(R)))$ for signing and $(\mathsf{pk}(R), \mathsf{inv}(\mathsf{pk}(R))$ for encryption. We assume that agents do not (necessarily) know each other's public keys in advance. Instead each role R has a certificate $Cert_R$ which is a signed message containing R's name and public keys (and possibly other information) and that are signed using the key $\mathsf{inv}(\mathsf{sigk}(s))$ where s is an honest key-server. Note that s is a constant (of type agent) here and not a role of the protocol. Finally T_C and N_2 are nonces, and K is a symmetric key.

The goal of this protocol is $KAS \bullet\!\!\rightarrow\!\!\bullet B : K$, because this key should be used for further secure communication (in Kerberos).

(a) Formalize this protocol in AnB and run OFMC on your specification (links on the class homepage).
Hints:
- The file should have the extension `.AnB` (case-in-sensitive).
- Call OFMC as follows:
  `ofmc <yourfile.AnB> -typed -numSess 1`

(b) Explain the attack output (there should be an attack...)

(c) If you omit `-typed` you (should...) get a different attack. Explain the attack output, does it reveal a realistic vulnerability?

(d) Fix the protocol so that both attacks are prevented and check it with OFMC, maybe also for a larger number of sessions.

Solution

(a) A possible formalization:

```
Protocol: PKINIT

Types:
  Agent C,KAS,s;
  Number TC,N2;
  Symmetric_key K;
  Function pk,sigk


Knowledge:
  C:   C, KAS, s, sigk(s), pk(s), {C,pk(C),sigk(C)}inv(sigk(s)),
       sigk(C), pk(C), inv(sigk(C)), inv(pk(C));
  KAS: KAS, s, sigk(s), pk(s), {KAS,pk(KAS),sigk(KAS)}inv(sigk(s)),
       sigk(KAS), pk(KAS), inv(sigk(KAS)), inv(pk(KAS))

Actions:
  C -> KAS: {C,pk(C),sigk(C)}inv(sigk(s)),{TC,N2}inv(sigk(C))
  KAS -> C: {{KAS,pk(KAS),sigk(KAS)}inv(sigk(s)),
             {K,N2}inv(sigk(KAS))}pk(C)

Goals:
  KAS *->* C: K
```

(b) (OFMC accepts this specification.)

(c) Attack:

```
ATTACK TRACE
(x502,1) -> i: {x502.pk(x502).sigk(x502)}_inv(sigk(s)).
                {TC(1).N2(1)}_inv(sigk(x502))
i -> (x502,1): {i.pk(i).sigk(i)}_inv(sigk(s)).
                {x310.N2(1)}_inv(sigk(i))
(x502,1) -> i: {{x502.pk(x502).sigk(x502)}_inv(sigk(s)).
                 {K(2).N2(1)}_inv(sigk(x502))}_(pk(i))
i -> (x502,1): {{x502.pk(x502).sigk(x502)}_inv(sigk(s)).
                 {K(2).N2(1)}_inv(sigk(x502))}_(pk(x502))
```

Here `x502` is a variable that stands for any agent playing both role A and KAS. This is unrealistic, but possible and OFMC finds this attack state first. Besides that it is exactly the attack from module 2.

(d) Without option `-typed` we get a type flaw attack:

```
ATTACK TRACE
(x402,1) -> i: {x402.pk(x402).sigk(x402)}_inv(sigk(s)).
                {TC(1).N2(1)}_inv(sigk(x402))
i -> (x402,1): {{x402.pk(x402).sigk(x402)}_inv(sigk(s)).
                 {TC(1).N2(1)}_inv(sigk(x402))}_(pk(x402))
```

Here, the intruder simply takes the first message of an honest agent, and encrypts it with the public key of that agent. The agent accepts $TC(1)$ (which the intruder knows) as a key for further communication. Again the attack is with a schema of an agent talking to itself. This may be considered in the case that several (trusted) parties in an organization use the same key-pairs, even to exchange messages amongst each other, but usually this is not very realistic. Note however that this attack can work in general if some agent works both in role C in some session and as role KAS in another session.

(e) Both attacks are prevented by having KAS sign the name C together with the nonce and the key K:

```
KAS -> C: {{KAS,pk(KAS),sigk(KAS)}inv(sigk(s)),
           {K,N2,C}inv(sigk(KAS))}pk(C)
```

Assignment 2.2: Dolev-Yao

Consider the Dolev-Yao deduction rules of module 3, slide 26, and let us first interpret terms in a free algebra.

(a) Consider the intruder knowledge

$$M = \{\ \{|k|\}_{h(n_1,n_2)}, \{n_1\}_{\mathsf{pk}(i)}, \{n_2\}_{\mathsf{inv}(\mathsf{pk}(a))}, \mathsf{pk}(a), \mathsf{pk}(i), \mathsf{inv}(\mathsf{pk}(i)), \{|secret|\}_k\ \}$$

where $h \in \Sigma_p$ is a public function.

Formally prove or disprove:

(1) $secret \in \mathcal{DY}(M)$
(2) $\{secret\}_{\mathsf{inv}(\mathsf{pk}(a))} \in \mathcal{DY}(M)$
(3) $\{|n_1|\}_{h(k,secret)} \in \mathcal{DY}(M)$

(b) Consider a model with public functions $\pi_1, \pi_2 \in \Sigma_p$ and with the algebraic properties

$$\begin{aligned} \pi_1(\langle m_1, m_2\rangle) &= m_1 \\ \pi_2(\langle m_1, m_2\rangle) &= m_2 \end{aligned}$$

Prove formally: in this model, the projection rules Proj_i of Dolev-Yao deduction rules are redundant, i.e., removing these rule does not change the set $\mathcal{DY}(M)$ in this model.

Solution

(a) Deduction in the free algebra:

(1) $\boxed{secret \in \mathcal{DY}(M)}$

$$\dfrac{\dfrac{}{\{\!|secret|\!\}_k \in \mathcal{DY}(M)}\,\mathsf{Axiom} \quad \dfrac{\dfrac{}{\{\!|k|\!\}_{h(n_1,n_2)} \in \mathcal{DY}(M)}\,\mathsf{Axiom} \quad \boxed{\Pi_1}}{k \in \mathcal{DY}(M)}\,\mathsf{DecSym}}{secret \in \mathcal{DY}(M)}\,\mathsf{DecSym}$$

where $\boxed{\Pi_1}$ is the following sub-proof:

$$\dfrac{\dfrac{\dfrac{}{\{n_1\}_{\mathsf{pk}(i)} \in \mathcal{DY}(M)}\,\mathsf{Axiom} \quad \dfrac{}{\mathsf{inv}(\mathsf{pk}(i)) \in \mathcal{DY}(M)}\,\mathsf{Axiom}}{n_1 \in \mathcal{DY}(M)}\,\mathsf{DecAsym} \quad \dfrac{\dfrac{}{\{n_2\}_{\mathsf{inv}(\mathsf{pk}(a))} \in \mathcal{DY}(M)}\,\mathsf{Axiom}}{n_2 \in \mathcal{DY}(M)}\,\mathsf{OpenSig}}{\dfrac{\langle n_1, n_2\rangle \in \mathcal{DY}(M)}{h(n_1,n_2) \in \mathcal{DY}(M)}\,\mathsf{Comp.}(h \in \Sigma_p)}\,\mathsf{Comp.}(\langle\cdot,\cdot\rangle \in \Sigma_p)$$

(2) $\boxed{t = \{secret\}_{\mathsf{inv}(\mathsf{pk}(a))} \notin \mathcal{DY}(M)}$

There are basically three ways to obtain a term in the free algebra: either it is directly in M, or it can be obtained by decryption/decomposition, or one can compose it. Now $t \notin M$, and it is also not a subterm of any term of M, so we cannot obtain it by decryption and decomposition. For composition, we would first need the key $k = \mathsf{inv}(\mathsf{pk}(a))$. However, $k \notin M$ either, it cannot be obtained by decryption/decomposition, and it cannot be composed since $\mathsf{inv}(\cdot)$ is not a public function.

Note: this proof is quite sketchy, some details of these statements are not trivial! In module 5 we will see a procedure for deciding $t \in \mathcal{DY}(M)$ and prove its correctness. That allows us to easily write such proofs precisely (and even obtain them automatically).

(3) $\boxed{\{\!|n_1|\!\}_{h(k,secret)} \in \mathcal{DY}(M)}$

We use that we showed $k, secret, n_1 \in \mathcal{DY}(M)$ already in the first example (and we can reuse the proofs from there), which we denote by $\star$:

$$\dfrac{\dfrac{\dfrac{\overset{\star}{k} \quad \dfrac{\star}{secret}}{\langle k, secret\rangle \in \mathcal{DY}(M)}\,\mathsf{Comp.}}{h(k,secret) \in \mathcal{DY}(M)}\,\mathsf{Comp.} \quad \dfrac{\star}{n_1}}{\{\!|n_1|\!\}_{h(k,secret)} \in \mathcal{DY}(M)}\,\mathsf{Comp.}$$

(b) Redundancy of decomposition rules in presence of explicit decomposition:

Recall that the Proj_1 rule was defined as:

$$\frac{\langle t_1, t_2\rangle \in \mathcal{DY}(M)}{t_1 \in \mathcal{DY}(M)}\ \mathsf{Proj}_1$$

We can simulate this rule (and similarly all other decomposition rules) in a model with explicit destructors (like π_1 for the projection to the first component) and suitable algebraic properties: first we apply the destructor to the term that we want to decompose and then use algebraic equivalence to simplify the term to the result of the decomposition:

$$\frac{\dfrac{\langle t_1, t_2\rangle \in \mathcal{DY}(M)}{\pi_1(\langle t_1, t_2\rangle) \in \mathcal{DY}(M)}\ \mathsf{Comp.}\ (\pi_1 \in \Sigma_p)}{t_1 \in \mathcal{DY}(M)}\ \mathsf{Algebra}$$

Note that this sequence of two deduction steps together has exactly the same assumption and consequence as the single Proj_1 rule. Therefore, in a derivation of a term, we can replace each occurrence of the Proj_1 rule by this sequence of two deduction steps (instantiating t_1 and t_2 accordingly). Thus, the Proj_1 rule is redundant in the presence of explicit decryption.

Assignment 2.3: Role Description of PKInit

- Take the AnB-notation of the Kerberos PKInit example from the first assignment and split this into two role descriptions; is it realistic that the agents can check the given format of messages if they do not know each other's public keys in advance?
- What are the free variables of each role?
- Which initial threads th_0 (instances of the roles) do we need to find the attack? What initial intruder knowledge IK_0 do we need?
- Show that the attack trace indeed works for your role description according to the snd and rcv rules of module 3, slide 35.

Solution

- The roles are:

$$\begin{array}{ll} C: & \mathsf{snd}(Cert_C, \{T_C, N_2\}_{\mathsf{inv}(\mathsf{sigk}(C))}) \cdot \mathsf{rcv}(\{Cert_{KAS}, \{K, N_2\}_{\mathsf{inv}(\mathsf{sigk}(KAS))}\}_{\mathsf{pk}(C)}) \\ KAS: & \mathsf{rcv}(Cert_C, \{T_C, N_2\}_{\mathsf{inv}(\mathsf{sigk}(C))}) \cdot \mathsf{snd}(\{Cert_{KAS}, \{K, N_2\}_{\mathsf{inv}(\mathsf{sigk}(KAS))}\}_{\mathsf{pk}(C)}) \end{array}$$

for $Cert_C = \{C, \mathsf{pk}(C), \mathsf{sigk}(C)\}_{\mathsf{inv}(\mathsf{pk}(s))}$ and similar for KAS.

The fact that all encryptions/signatures are received as such is indeed realistic: both roles have the server's public signing key $\mathsf{sigk}(s)$ and can thus verify the certificates $Cert_C$ and $Cert_{KAS}$ and also obtain each other's public encryption and signature keys.

Thus, KAS can verify the signature of C and C can decrypt the encryption with $\mathsf{pk}(C)$ (since C knows $\mathsf{inv}(\mathsf{pk}(C))$) and verify KAS' signature.

It is questionable to receive each other's public encryption/signature keys in the form $\mathsf{pk}(C)$ and $\mathsf{sigk}(C)$ etc. In general, when using a key-server, we shall assume that the public-keys are not necessarily known in advance by all participants, i.e. the functions $\mathsf{pk}(\cdot)$ and $\mathsf{sigk}(\cdot)$ are not known to all participants (but just by the server s); thus one cannot really check that a received key is of the "form" $\mathsf{pk}(C)$ etc. However, since an honest server s is certifying the keys, the intruder cannot manipulate this, so the restriction to keys of this "form" is indeed attack-preserving.

- $fv(C) = \{C, T_C, N_2\}$ and $fv(KAS) = \{KAS, K\}$. One may argue that also $KAS \in fv(C)$ would make sense as C should send the message to a particular recipient (and not accept replies signed by a different KAS') and we treat KAS as a free variable of C in the following.

- th_0 needs at least the following initial threads to give the attack:

$$\begin{aligned} tid_1 &\mapsto PKINIT(C)[C \mapsto c, T_c \mapsto t, N_2 \mapsto n, KAS \mapsto i] \\ tid_2 &\mapsto PKINIT(KAS)[KAS \mapsto b, K \mapsto k] \end{aligned}$$

  where c, t, n, b, k are some constants and i is the intruder. Note that we have explicitly instantiated KAS in role C for clarity.

  The necessary intruder knowledge so that i can act in the role of KAS:

$$IK_0 = \{i, c, kas, \mathsf{sigk}(s), \mathsf{pk}(s), \mathsf{pk}(i), \mathsf{inv}(\mathsf{pk}(i)), \mathsf{sigk}(i), \mathsf{inv}(\mathsf{sigk}(i)), Cert_i\}$$

  where $Cert_i = \{i, \mathsf{pk}(i), \mathsf{sigk}(i)\}_{\mathsf{inv}(\mathsf{sigk}(s))}$. Note that the intruder as a participant should also have a valid certificate.

- We just give the steps that are done and the resulting trace.
  (a) First, we have a step of thread tid_1:

$$(tid_1, \mathsf{snd}(Cert_c, \{t, n\}_{\mathsf{inv}(\mathsf{sigk}(c))}))$$

  (b) Now, the intruder can generate a similar message from his initial knowledge and the constants t and n from c's message, and kas can receive it:

$$(tid_2, \mathsf{rcv}(Cert_i, \{t, n\}_{\mathsf{inv}(\mathsf{sigk}(i))}))$$

  (c) kas will answer with the appropriate second message:

$$(tid_2, \mathsf{snd}(\{Cert_{kas}, \{k, n\}_{\mathsf{inv}(\mathsf{sigk}(kas))}\}_{\mathsf{pk}(i)}))$$

  (d) The intruder can decrypt this message and re-encrypt it for c:

$$(tid_1, \mathsf{rcv}(\{Cert_{kas}, \{k, n\}_{\mathsf{inv}(\mathsf{sigk}(kas))}\}_{\mathsf{pk}(c)}))$$