IBS - Laboratorní cvičení

Pavel Frýz

10. května 2016

Cíl úlohy

- Seznámit se s nástrojom Scyther.
- Vyzkoušet si implementaci symetrických a asymetrických bezpečnostních protokolů.
- Vyzkoušet si verifikaci bezpečnostních protokolů v nástroji Scyther.

Studijní materiály

- V této úloze bude cílem implementovat protokoly *Needham-Schroeder Public Key*, *Otway Rees* a protokol *Denning-Sacco*.
- Studijní materiály
  - Needham-Schroeder Public Key
    `http://www.lsv.ens-cachan.fr/Software/spore/nspk.html`
    Roger Needham and Michael Schroeder. Using encryption for authentification in large networks of computers. *Communications of the ACM*, 21(12), December 1978.
  - Otway Rees
    `http://www.lsv.ens-cachan.fr/Software/spore/otwayRees.html`
    John Clark and Jeremy Jacob. A survey of authentication protocol literature : Version 1.0., November 1997.
  - Denning-Sacco
    `http://www.lsv.ens-cachan.fr/Software/spore/denningSacco.html`
    D. Denning and G. Sacco. Timestamps in key distributed protocols. *Communication of the ACM*, 24(8):533–535, 1981.

Příprava prostředí

- Stáhněte a nainstalujte knihovnu GraphViz-Graph Visualization Software:
  `http://www.graphviz.org/Download.php`.
- Stáhněte instalační soubor Pythonu 2.x a spusťte instalaci:
  `http://www.python.org/download/`
- Stáhněte a nainstalujte knihovnu wxPython, vyberte verzi odpovídající verzi Pythonu:
  `http://www.wxpython.org/download.php`
- Stáhněte a rozbalte archiv s nástrojem Scyther:
  `http://people.inf.ethz.ch/cremersc/scyther/install-generic.html`
- Spusťte soubor `scyther-gui.py` z adresáře aplikace.

1 Implementace protokolu Needham-Schroeder Public Key

V tomto příkladě si vyzkoušíte implementaci a verifikaci *Needham-Schroeder Public Key* protokolu, který slouží pro obousměrnou autentizaci pomocí důvěryhodného serveru.

1. Po spuštění nástroje zadávajte kód protokolu v hlavní editovací části okna.

2. Na úvod deklarujeme funkce pro veřejné a soukromé klíče. Deklarace pomocí klíčového slova `const`, učiní funkci `pk` veřejnou. Pomocí slova `secret` označíme pak funkci `sk` jako tajnou.

```
1 const pk: Function;
2 secret sk: Function;
```

3. Následně deklarujeme, že tyto funkce tvoří pár asymetrických klíčů.

```
3 inversekeys(pk,sk);
```

4. Dále deklarujeme protokol klíčovým slovom `protocol` nasledovaný jeho jménem a jmény účastníků.

```
4 protocol NeedhamSchroeder(A,B,S)
5 {
```

5. Následuje definice jednotlivých uživatelů. Definice účastníka začíná klíčovým slovem `role`. Prvně definujeme účastníka `A` a deklarujeme lokální proměnné pro nonce `Na`, který vytváří, a pro nonce `Nb`, který obdrží od účastníka `B`.

```
6   role A
7   {
8     const Na: Nonce;
9     var Nb: Nonce;
```

6. Dále definujem příchozí a odchozí zprávy účastníka `A`. Příchozí a odchozí zpráva začíná klíčovým slovem `recv`, respektive `send` následované označením zprávy. V kulatých závorkách poté následuje od koho zpráva pochází, příjemce zprávy a text zprávy.

```
10    send_1(A,S,(A,B));
11    recv_2(S,A,{pk(B),B}sk(S));
12    send_3(A,B,{Na,A}pk(B));
13    recv_6(B,A,{Na,Nb}pk(A));
14    send_7(A,B,{Nb}pk(B));
```

   kde `{x}pk(A)` značí zprávu `x` zašifrovanou veřejným klíčem `A`.

7. Nakonec přidáme bezpečnostní požadavky protokolu. V našem případě se jedná o utajení nonců, a neinjektivní synchronizaci

```
15    claim_A1(A,Secret,Na);
16    claim_A2(A,Secret,Nb);
17    claim_A3(A,Nisynch);
18  }
```

8. Obdobně přidáme i účastníka `B` a server `S`.

```
19  role B
20  {
21    const Nb: Nonce;
22    var Na: Nonce;
23    recv_3(A,B,{Na,A}pk(B));
24    send_4(B,S,(B,A));
25    recv_5(S,B,{pk(A),A}sk(S));
```

```
      send_6(B,A,{Na,Nb}pk(A));
      recv_7(A,B,{Nb}pk(B));
      claim_B1(B,Secret,Na);
      claim_B2(B,Secret,Nb);
      claim_B3(B,Nisynch);
    }
    role S
    {
      recv_1(A,S,(A,B));
      send_2(S,A,{pk(B),B}sk(S));
      recv_4(B,S,(B,A));
      send_5(S,B,{pk(A),A}sk(S));
    }
}
```

9. Nakonec přidáme nedůvěryhodného klienta, jehož soukromý klíč byl zpronevěřen útočníkem.

```
const C: Agent;
untrusted C;
compromised sk(C);
```

10. Spuťte verifikaci protokolu(v menu `Verify->Verify protocol`), a projděte výsledky verifikace. Nechte si zobrazit případný útok.

2 Implementace protokolu Otway Rees

V tomto příkladě si vyzkoušíte implementaci a verifikaci *Otway Rees* protokolu, který používá symetrickou kryptografii.

1. Po spuštění nástroje zadávajte kód protokolu v hlavní editovací části okna.

2. Na úvod deklarujeme funkci pro symetrický klíč. Protože se jedná o tajný klíč označíme funkci klíčovým slovem `secret`. Dále deklarujeme nový typ pro distribuovaný klíč `Key`.

```
secret k: Function;
usertype Key;
```

3. Dále deklarujeme protokol klíčovým slovom `protocol` nasledovaný jeho jménem a jmény účastníků.

```
protocol OtwayRees(A,B,S)
{
```

4. Následuje definice jednotlivých uživatelů. Definice účastníka začíná klíčovým slovem `role`. Prvně definujeme účastníka `A` a deklarujeme lokální proměnné pro nonce `Na` a `M`, a pro obdržený klíč `Kab`.

```
  role A
  {
    const Na: Nonce;
    const M: Nonce;
    var Kab: Key;
```

5. Dále definujem příchozí a odchozí zprávy účastníka `A`. Příchozí a odchozí zpráva začíná klíčovým slovem `recv`, respektive `send` následované označením zprávy. V kulatých závorkách poté následuje od koho zpráva pochází, příjemce zprávy a text zprávy.

```
    send_1(A,B,(M,A,B,{Na,M,A,B}k(A,S)));
    recv_4(B,A,(M,{Na,Kab}pk(A,S)));
```

kde `{x}pk(A,S)` značí zprávu `x` zašifrovanou sdíleným klíčem `A` a `S`.

6. Nakonec přidáme bezpečnostní požadavky protokolu. V našem případě se jedná o utajení distribuovaného klíče `Kab` a neinjektivní synchronizaci.

```
12      claim_A1(A,Secret,Kab);
13      claim_A2(A,Nisynch);
14    }
```

7. Obdobně přidáme i účastníka `B`. Protože účastníci musí být schopni dešifrovat příchozí zprávy, musíme u účasníka `B` navíc deklarovat proměnné `T1` a `T2` typu `Ticket`, do kterých můžeme uložit zašifrovanou zprávu.

```
15    role B
16    {
17      const Nb: Nonce;
18      var M: Nonce;
19      var Kab: Key;
20      var T1, T2: Ticket;
21      recv_1(A,B,(M,A,B,T1));
22      send_2(B,S,(M,A,B,T1,{Nb,M,A,B}k(B,S)));
23      recv_3(S,B,(M,T2,{Nb,Kab}k(B,S)));
24      send_4(B,A,(M,T2));
25      claim_B1(B,Secret,Kab);
26      claim_B2(B,Nisynch);
27    }
```

8. Dále přidáme server `S`.

```
28    role S
29    {
30      const Kab: Key;
31      var Na, Nb, M: Nonce;
32      recv_2(B,S,(M,A,B,{Na,M,A,B}k(A,S),{Nb,M,A,B}k(B,S)));
33      send_3(S,B,(M,{Na,Kab}k(A,S),{Nb,Kab}k(B,S)));
34    }
35  }
```

9. Nakonec přidáme nedůvěryhodného klienta, jehož sdílený klíč byl zpronevěřen útočníkem.

```
36  const C,D: Agent;
37  untrusted C;
38  compromised k(C,D);
39  compromised k(D,C);
```

10. Spuťte verifikaci protokolu(v menu `Verify->Verify protocol`), a projděte výsledky verifikace. Nechte si zobrazit případný útok.

3 Implementace protokolu Denning-Sacco

V tomto příkladě si vyzkoušíte implementaci a verifikaci protokolu *Denning-Sacco*.

1. Po spuštění nástroje zadávajte kód protokolu v hlavní editovací části okna.

2. Na úvod deklarujeme funkci pro symetrický klíč. Protože se jedná o tajný klíč označíme funkci klíčovým slovem `secret`. Dále deklarujeme nový typ pro distribuovaný klíč `Key` a typ `Timestamp` pro časovou značku.

```
1  secret k: Function;
2  usertype Key;
3  usertype Timestamp;
```

3. Dále deklarujeme protokol klíčovým slovom `protocol` nasledovaný jeho jménem a jmény účastníků.

```
4  protocol DenningSacco(A,B,S)
5  {
```

4. Následuje definice jednotlivých uživatelů. Definice účastníka začíná klíčovým slovem `role`. Prvně definujeme účastníka `A` a deklarujeme lokální proměnné pro časovou známku `T`, pro obdržený klíč `Kab` a ticket pro přeposílanou zprávu `M`.

```
6    role A
7    {
8      var T: Timestamp;
9      var M: Ticket;
10     var Kab: Key;
```

5. Dále definujem příchozí a odchozí zprávy účastníka `A`. Příchozí a odchozí zpráva začíná klíčovým slovem `recv`, respektive `send` následované označením zprávy. V kulatých závorkách poté následuje od koho zpráva pochází, příjemce zprávy a text zprávy.

```
11     send_1(A,S,(A,B));
12     recv_2(S,A,{B,Kab,T,M}k(A,S));
13     send_3(A,B,M);
```

   kde `{x}pk(A,S)` značí zprávu `x` zašifrovanou sdíleným klíčem `A` a `S`.

6. Nakonec přidáme bezpečnostní požadavky protokolu. V našem případě se jedná o utajení distribuovaného klíče `Kab` a neinjektivní synchronizaci a agreement.

```
14     claim_A1(A,Secret,Kab);
15     claim_A2(A,Nisynch);
16     claim_A3(A,Niagree);
17   }
```

7. Obdobně přidáme i účastníka `B`.

```
18   role B
19   {
20     var T: Timestamp;
21     var Kab: Key;
22     recv_3(A,B,{Kab,A,T}k(B,S));
23     claim_B1(B,Secret,Kab);
24     claim_B2(B,Nisynch);
25     claim_B3(B,Niagree);
26   }
```

8. Dále přidáme server `S`.

```
27   role S
28   {
29     const Kab: Key;
30     const T: Timestamp;
31     recv_1(A,S,(A,B));
32     send_2(S,A,{B,Kab,T,{Kab,A,T}k(B,S)}k(A,S));
33   }
34 }
```

9. Nakonec přidáme nedůvěryhodného klienta, jehož sdílený klíč byl zpronevěřen útočníkem.

```
35 const C,D: Agent;
36 untrusted C;
37 compromised k(C,D);
38 compromised k(D,C);
```

10. Spuťte verifikaci protokolu(v menu `Verify->Verify protocol`), a projděte výsledky verifikace. Nechte si zobrazit případný útok.