Failles

Failles

- Failles: on dit qu'un protocole contient une faille s'il ne remplit pas les propriétés de sécurité pour lesquelles il a été conçu.
- Par exemple, on dit qu'un protocole d'authentification contient une faille si un agent A arrive à prouver à un agent B qu'il est un autre agent C.
- Un protocole qui achemine des informations confidentielles est dit défaillant si un intrus est capable de connaître l'une des informations secrètes.
- La preuve d'existence d'une faille dans un protocole est généralement une trace valide de ce protocole montrant que l'un des objectifs visés n'est pas atteint et cette trace en construit un contre exemple.

Failles

- On peut regrouper les failles en classes de failles:
  - Les failles de fraîcheur
  - Les failles d’oracle
  - Les failles d’association
  - Les failles de type
  - Les failles d’implantation
  - Les failles de répudiation

Les failles de fraîcheur

- Une faille de fraîcheur se produit quand un intrus attaque un protocole en utilisant des informations récupérées des sessions précédentes

Message 1 :	A	$\longrightarrow$	S	:	$A.B.N_a$
Message 2 :	S	$\longrightarrow$	A	:	$\{N_a.B.k_{ab}.\{k_{ab}.A\}_{k_{bs}}\}_{k_{as}}$
Message 3 :	A	$\longrightarrow$	B	:	$\{k_{ab}.A\}_{k_{bs}}$
Message 4 :	B	$\longrightarrow$	A	:	$\{N_b\}_{k_{ab}}$
Message 5 :	A	$\longrightarrow$	B	:	$\{N_b + 1\}_{k_{ab}}$

Les failles de fraîcheur

- Si un intrus I intercepte le Message 3: $\{k'_{ab}.A\}_{k_{bs}}$ d'une session précédente, il peut procéder au scénario d'attaque suivant :

Message 3 :	$I(A)$	$\longrightarrow$	B	:	$\{k'_{ab}.A\}_{k_{bs}}$
Message 4 :	B	$\longrightarrow$	$I(A)$	:	$\{N_b\}_{k'_{ab}}$
Message 5 :	$I(A)$	$\longrightarrow$	B	:	$\{N_b+1\}_{k'_{ab}}$

A la fin de cette séquence, I réussit à faire croire à B que k'_{ab} est la clé de session partagée avec A.

Les failles d’oracle

- Une faille d’oracle se produit quand un intrus parvient à utiliser le protocole afin de connaître des informations secrètes ou des messages utiles pour attaquer le protocole. Nous distinguons deux types de failles d’oracles :
- Faille d’oracle à rôle simple : faille d’oracle associée à un protocole qui ne permet pas à un agent de changer de rôle d’une session à une autre.
- Faille d’oracle à rôle multiple : faille d’oracle associée à un protocole au cours duquel un agent peut changer de rôle d’une session à une autre.

Les failles d'oracle

Message 1 :	A	$\longrightarrow$	B	: $\{N_a.A\}_{k_b}$
Message 2 :	B	$\longrightarrow$	A	: $\{N_a.N_b\}_{k_a}$
Message 3 :	A	$\longrightarrow$	B	: $\{N_b\}_{k_b}$

1.1 $A \rightarrow I : \{ N_a , A \}_{ki}$

2.1 $I(A) \rightarrow B : \{ N_a , A \}_{kb}$

2.2 $B \rightarrow I(A) : \{ N_a , N_b \}_{ka}$

1.2 $I \rightarrow A : \{ N_a , N_b \}_{ka}$

1.3 $A \rightarrow I : \{ N_b \}_{ki}$

2.3 $I(A) \rightarrow B : \{ N_b \}_{kb}$

À la fin de ce scénario, I réussit à se faire passer pour A au regard de B en manipulant deux sessions du même protocole et sans changer de rôle.

Les failles d'oracle

- Considérons le protocole suivant :

Message 1 :	A	$\longrightarrow$	B	:	$\{N_a\}_{k_{ab}}$
Message 2 :	B	$\longrightarrow$	A	:	$\{N_a+1\}_{k_{ab}}$

Un scénario d'attaque peut être mené comme suit :

Message 1.1 :	A	$\longrightarrow$	$I(B)$	:	$\{N_a\}_{k_{ab}}$
Message 2.1 :	$I(B)$	$\longrightarrow$	A	:	$\{N_a\}_{k_{ab}}$
Message 2.2 :	A	$\longrightarrow$	$I(B)$	:	$\{N_a+1\}_{k_{ab}}$
Message 1.2 :	$I(B)$	$\longrightarrow$	A	:	$\{N_a+1\}_{k_{ab}}$

À la fin de ce scénario, I réussit à se faire passer pour B au regard de A en manipulant deux sessions du même protocole et en jouant tantôt le rôle de A, tantôt le rôle de B.

Les failles d'association

- Une faille d'association se produit quand un intrus parvient à corrompre la correspondance entre la clé publique d'un principal et son identité.

Message 1 : A $\rightarrow$ S : A, B, N_a
Message 2 : S $\rightarrow$ A : S, $\{S, A, N_a, K_b\}_{K^{-1}_s}$

Message 1.1 : A $\rightarrow$ I(S) : A, B, N_a
Message 2.1 : I(A) $\rightarrow$ S : A, I, N_a
Message 2.2 : S $\rightarrow$ I(A) : S, $\{S, A, N_a, K_i\}_{K^{-1}_s}$
Message 1.2 : I(S) $\rightarrow$ A : S, $\{S, A, N_a, K_i\}_{K^{-1}_s}$

À la fin de cette attaque, et en utilisant deux sessions, l'intrus réussit à faire croire à A que la clé publique de B est k_i.

Les failles de type

- Une faille de type se produit quand un intrus base son attaque sur la substitution d'un champ (ou une composante) d'un message par un champ de type différent.

Message 1 : A	$\rightarrow$	B : M, A, B, $\{N_a, M, A, B\}_{Kas}$
Message 2 : B	$\rightarrow$	S : M, A, B, $\{N_a, M, A, B\}_{Kas}$, $\{N_b, M, A, B\}_{Kbs}$
Message 3 : S	$\rightarrow$	B : M, $\{N_a, K_{ab}\}_{Kas}$, $\{N_b, K_{ab}\}_{Kbs}$
Message 5 : B	$\rightarrow$	A : M, $\{N_a, K_{ab}\}_{Kas}$

Message 1 : A	$\rightarrow$	I(B) : M, A, B, $\{N_a, M, A, B\}_{Kas}$
Message 5 : I(B)	$\rightarrow$	A : M, $\{N_a, M, A, B\}_{Kas}$

À la fin de cette attaque, l'intrus réussit à faire croire à A que la clé qu'il partage avec B est la chaine de bits M,A,B.

Les failles d'implantation

- Une faille d'implantation se produit quand un intrus profite des faiblesses dues à la combinaison de l'algorithme de chiffrement et du protocole cryptographique.

Message 1 :	A	$\longrightarrow$	B	:	$\{M\}_{k_a}$
Message 2 :	B	$\longrightarrow$	A	:	$\{\{M\}_{k_a}\}_{k_b}$
Message 3 :	A	$\longrightarrow$	B	:	$\{M\}_{k_b}$

Exemple I : Soit la fonction de chiffrement:

$$\left\{\begin{array}{l} encrypt(m,k) = m \oplus k \\ \\ 0 \oplus x = x \\ x \oplus x = 0 \\ x \oplus y = y \oplus x \\ x \oplus (y \oplus z) = (x \oplus y) \oplus z \end{array}\right\}$$

Les failles d'implantation

- Un scénario d'attaque peut être mené comme suit :
- L'intrus, en recevant les trois messages : Message 1, Message 2 et Message 3, il les additionne à l'aide de l'opérateur ou exclusif :

$$
\begin{aligned}
\text{Message 1} \oplus \text{Message 2} \oplus \text{Message 3} &= \\
\{M\}_{k_a} \oplus \{\{M\}_{k_a}\}_{k_b} \oplus \{M\}_{k_b} &= \\
(M \oplus k_a) \oplus (M \oplus k_a \oplus k_b) \oplus (M \oplus k_b) &= \\
(M \oplus M \oplus M) \oplus (k_a \oplus k_a) \oplus (k_b \oplus k_b) &= \\
M \oplus 0 \oplus 0 &= M
\end{aligned}
$$

Conclusion : L 'intrus a pu connaître la valeur de M.

Failles de répudiation

- Une faille de répudiation survient lorsqu'un principal peut nier sa participation à une session du protocole.

Remarque

- Une faille peut contenir un nombre important d'étapes de communication.

  - Difficile à la trouver manuellement.
  - Besoin d'un outil automatique (Scyther, Avispa…) qui nous aide à trouver des failles.

L'outil Scyther

- L'outil Scyther a été développé par Cas Cremers en 2007.
- Scyther est un contrôleur automatisé des protocoles de sécurité, qui s'est avéré être un outil efficace pour la vérification, l'analyse des protocoles de sécurité et la détection des attaques.
- Il peut vérifier les aspects de sécurité de la plupart des protocoles avec un nombre illimité de sessions et dans un temps très rapide.
- Scyther est le seul outil existant actuellement capable de vérifier la synchronisation, cette dernière se traduit par le fait que les messages sont transmis exactement comme ils sont décrits dans le protocole. En d'autres termes chaque fois qu'un initiateur I termine une exécution du protocole avec un répondeur R, et R a été en cours d'exécution avec le protocole I. c'est à dire, tous les messages sont reçus tels qu'ils ont été envoyés, dans l'ordre dans lesquels ils sont décrits par le protocole.

- L'outil fournit une interface utilisateur graphique qui complète la ligne de commande et les interfaces de script Python. L'interface graphique s'adresse aux utilisateurs intéressés par la vérification ou la compréhension d'un protocole. Les interfaces de ligne de commande et de script facilitent l'utilisation de Scyther pour les tests de vérification de protocole à grande échelle.

```
protocol Test(U, V){
  role U { };
  role V { };
};
```

```
role U{
  fresh Ru: Nonce; # Freshly generated nonce
  var Rv: Nonce; # Variable for receiving a nonce

  send_1(U, V, Ru); # Send message to V containing Ru
  recv_2(V, U, Ru, Rv); # Receive message from V containing
    Ru and Rv. The received Rv value is stored as the
    variable Rv.
};
```

```
role V{
  [...]

  recv_1(U, V, Ru); # Receive message sent from U containing
    Ru. The received Ru nonce is stored as the variable Ru.
  send_2(V, U, Ru, Rv); # Send message to U containing the
    received nonce Ru and the freshly generated Rv.
};
```

```
role U{
  [...]

  # Claims:
  claim_F1(U, Secret, Ru);
  claim_F2(U, Secret, Rv);
};
```

L'agent jouant le rôle U sait que l'intrus ne pourra jamais avoir connaissance de Ru,Rv.

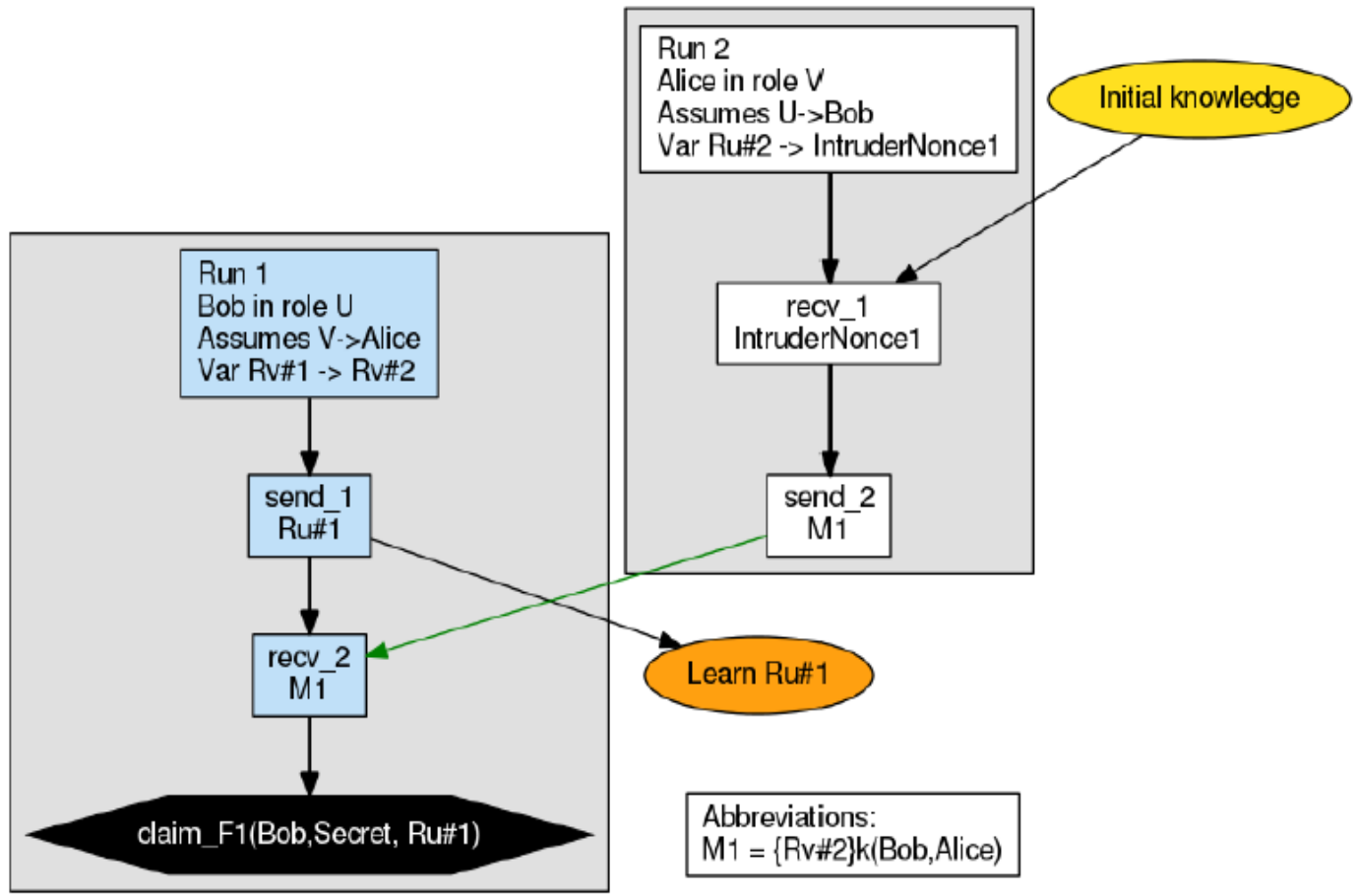
```
role U {
  [...]

  # Claims:
  [...]

  claim_U4(U, Nisynch);
}
```

Claim				Status		Comments
TEST	U	TEST,U1	Alive	Ok	Verified	No attacks.
		TEST,U2	Weakagree	Ok	Verified	No attacks.
		TEST,U3	Niagree	Ok	Verified	No attacks.
		TEST,U4	Nisynch	Ok	Verified	No attacks.
		TEST,U5	Commit V,Ru	Ok	Verified	No attacks.
	V	TEST,V1	Alive	Ok	Verified	No attacks.
		TEST,V2	Weakagree	Ok	Verified	No attacks.
		TEST,V3	Niagree	Ok	Verified	No attacks.
		TEST,V4	Nisynch	Ok	Verified	No attacks.

Claim				Status		Comments	Patterns
TEST	U	TEST,F1	Secret Ru	**Fail**	Falsified	Exactly 1 attack.	1 attack
		TEST,F2	Secret Rv	**Fail**	Falsified	Exactly 1 attack.	1 attack


Run 2
Alice in role V
Assumes U->Bob
Var Ru#2 -> IntruderNonce1
Initial knowledge
recv_1
IntruderNonce1
send_2
M1
Run 1
Bob in role U
Assumes V->Alice
Var Rv#1 -> Rv#2
send_1
Ru#1
recv_2
M1
Learn Ru#1
claim_F1(Bob,Secret, Ru#1)
Abbreviations:
M1 = {Rv#2}k(Bob,Alice)