Verificação Automática de Protocolos de Segurança com a ferramenta Scyther

Tadeu Jenuario[1,3], João Otávio Chervinski[1], Giulliano Paz[2,3], Diego Kreutz[1,2,3]

[1]Laboratório de Estudos Avançados (LEA)
[2] Mestrado Profissional em Engenharia de Software (MPES)
[3] Universidade Federal do Pampa (UNIPAMPA)
{gnoario,joaootaviors,giulliano94}@gmail.com, kreutz@acm.org

Resumo. *Protocolos de segurança (e.g., Needham-Schroeder, Kerberos, SSH, TLS) representam o alicerce das comunicações do mundo atual. Um dos desafios de projeto destes protocolos é garantir a sua própria segurança. Atualmente, existem ferramentas desenvolvidas especificamente para a verificação formal e automática de protocolos de segurança, como a Scyther, CryptoVerif e AVISPA. Entretanto, estas ferramentas são ainda pouco conhecidas e utilizadas na prática por projetistas de protocolos e estudantes de computação. Este trabalho tem por objetivo contribuir para diminuir esta lacuna através da introdução e demonstração prática de utilização da ferramenta Scyther, desenvolvida para auxiliar na verificação automática de protocolos de segurança.*

1. Introdução

Com a popularização da Internet, a segurança das comunicações tornou-se crítica para proteger os dados das entidades comunicantes. Atualmente, os mecanismos (e.g., geração e troca de chaves) e as propriedades essenciais de segurança da informação (e.g., confidencialidade, integridade e autenticidade) são asseguradas por conjuntos de protocolos específicos, como IKE, Needham-Schroeder, Kerberos, IPSec, SSH e TLS.

Um dos maiores desafios no desenvolvimento de protocolos de segurança é garantir a corretude do próprio protocolo, desde o projeto até a implementação. Devido a isso, a verificação automática e formal de protocolos, utilizando métodos formais e matemáticos, vem tornando-se cada vez mais frequente e imprescindível na comunidade de segurança [Chudnov et al. 2018, Li et al. 2018, Kreutz et al. 2019]. Pesquisas relatam, por exemplo, que o processo de verificação formal já contribuiu também para a correção de protocolos que estavam em utilização [Dalal et al. 2010, Cremers et al. 2016, Affeldt and Marti 2013].

Atualmente, existem ferramentas desenvolvidas especificamente para auxiliar o projeto e a verificação automática de protocolos, como Scyther [Cremers 2006], AVISPA [Armando et al. 2005] e ProVerif [Blanchet et al. 2018]. No entanto, estas ferramentas ainda são pouco conhecidas e utilizadas pela comunidade. Por exemplo, analisando alguns dos principais cursos de computação do Rio Grande do Sul e as últimas três edições do WRSeg e do SBSeg, dois dos principais eventos de segurança do Brasil, não encontramos nenhum trabalho sobre verificação formal de protocolos de segurança utilizando essas ferramentas.

O objetivo deste trabalho é contribuir para a difusão do conhecimento e da importância de utilizar ferramentas de verificação automática e formal de pro-

tocolos de segurança. Neste sentido, as principais contribuições do trabalho podem ser resumidas em: (a) introdução às semânticas operacionais básicas da ferramenta Scyther; (b) descrição didática do protocolo Needham-Schroeder de chaves assimétricas [Needham and Schroeder 1978] utilizando a semântica da ferramenta; e (c) demonstração e discussão do processo de análise e correção de falhas do protocolo.

2. A Ferramenta Scyther: Semânticas Operacionais

Assim como ocorre com as demais ferramentas, a verificação automática de protocolos com a Scyther requer a utilização de uma linguagem própria [Cremers 2006]. A seguir são apresentadas algumas das principais semânticas operacionais da ferramenta, que servirão como base para o projeto e análise formal do protocolo de Needham-Schroeder (ver Seções 3 e 4).

Termos atômicos: a ferramenta Scyther manipula termos de modo que o protocolo apresente o comportamento desejado. Os termos atômicos, ou tipos específicos de dados, servem para definir os dados que serão utilizados no protocolo. Entre eles estão: o *var*, que define variáveis para armazenar os dados recebidos de uma comunicação; o *const*, que define constantes que são geradas para cada instanciação de uma função e são consideradas variáveis locais; e o *fresh*, que representa variáveis que serão iniciadas com valores pseudo-aleatórios.

Chaves assimétricas: a estrutura de chave públicas ou assimétricas é predefinida pela ferramenta, sendo $sk(X)$ correspondente a uma chave privada de X e $pk(X)$ a chave pública correspondente. Um dado ni cifrado utilizando uma chave pública (e.g., $\{ni\}pk(X)$) só poderá ser decifrado com a chave privada correspondente ($sk(X)$).

Tipos predefinidos: determinam o comportamento de uma função ou termo. Por exemplo, o tipo **Agent** é utilizado para criar um agente que irá interagir nas comunicações. **Function** é um tipo especial que define um termo como sendo uma função que pode receber uma lista de termos como parâmetro. O tipo **Nonce** é considerado o tipo padrão para termos e é utilizado para armazenar valores utilizados durante a troca de mensagens.

Tipos básicos de eventos, como **send** (enviar) e **recv** (receber), são utilizados para a comunicação entre os agentes de um protocolo (e.g., Alice e Bob). É comum que cada evento de envio possua um evento de recebimento correspondente (e.g., **send_1** e **recv_1**). Para enviar uma mensagem de Alice para Bob, contendo o dado ni cifrado com a chave pública de Bob, basta utilizar a sintaxe $send\_1(Alice, Bob, \{ni\}pk(Bob))$. Para receber a mensagem enviada de Alice para Bob, basta utilizar a sintaxe $recv\_1(Alice, Bob, \{ni\}pk(Bob))$. Bob irá receber a mensagem de Alice e utilizar a sua chave privada correspondente para decifrar o dado contido na variável ni.

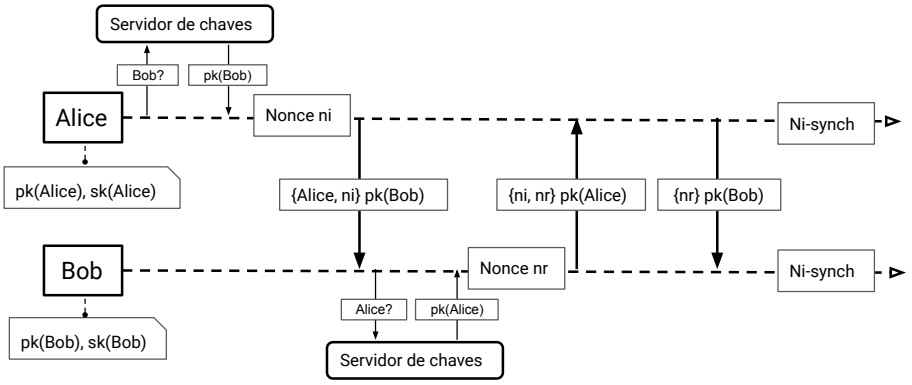
Eventos de afirmação (*claim*) são utilizados em especificações de funções para modelar propriedades de segurança. Na prática, após afirmar que uma variável é secreta, a ferramenta irá verificar se a afirmação procede durante a execução do protocolo, i.e., verificar se somente as partes comunicantes possuem acesso ao dado secreto. Por exemplo, para afirmar que uma variável ni é secreta, utiliza-se o termo **Secret** dentro de um evento de afirmação, i.e., $claim(Bob, Secret, ni)$. Também pode-se utilizar o termo **Nisynch** dentro de um evento de afirmação, i.e., $claim(Bob, Nisynch)$ para verificar se todas as mensagens recebidas pelo receptor foram de fato enviadas pelo parceiro de comunicação e não por um agente desconhecido.

3. O Protocolo Needham-Schroeder

O protocolo Needham-Schroeder fornece um método de autenticação para dois participantes em uma rede insegura utilizando criptografia assimétrica [Needham and Schroeder 1978]. Entretanto, o protocolo original apresenta falhas de segurança [Lowe 1995]. O objetivo é demonstrar e corrigir as falhas de segurança do protocolo utilizando a ferramenta Scyther.

A Figura 1 apresenta um diagrama da comunicação entre Alice e Bob utilizando o protocolo Needham-Schroeder. É assumido que cada usuário possui a chave privada e a chave pública do seu par. Alice deseja comunicar-se com Bob e, para tanto, requisita ao servidor de chaves a chave pública do Bob. Na sequência, Alice gera e envia para Bob um *nonce* ni juntamente com sua identificação. A mensagem é cifrada com a chave pública do Bob. Ao receber a mensagem, Bob decodifica-a e recupera o *nonce* ni e a identificação da remetente. Bob lê a identificação de Alice e requisita a respectiva chave pública para o servidor de chaves. No próximo passo, Bob gera e envia para Alice um novo *nonce* nr, criptografado com a chave pública da receptora. Ao receber a mensagem, Alice verifica que o *nonce* ni recebido é o mesmo que foi enviado para Bob anteriormente. Finalmente, Alice então envia para Bob o *nonce* nr. Ao receber e verificar os respectivos *nonces*, utilizando as respectivas chaves privadas, Alice e Bob são capazes de confirmar a autenticidade das mensagens.

Figura 1. Alice e Bob utilizando o protocolo Needham-Schroeder



O Algoritmo 1 apresenta o protocolo Needham-Schroeder na semântica do Scyther. As chaves públicas pk e privadas sk são declaradas globalmente (linhas 1 e 2), embora sejam atribuídas autonomamente a Alice e Bob pela ferramenta. O comando *inversekeys* (linha 3) determina que as chaves pk e sk são inversas, ou seja, ao cifrar com a chave pública só é possível decifrar com a chave secreta e vice-versa. Para a execução de ataques ao protocolo, é definido um agente não-confiável *Eve* (linhas 5 e 6).

A chamada à função **protocol** (linha 6) marca o início da especificação do protocolo ns, com três agentes (Alice, Bob e Eve), sendo que Alice e Bob possuem papéis (**role**) explícitos. A variável ni (linha 8), do tipo **Nonce** e antecedida por **fresh**, irá conter um valor pseudo-aleatório (e.g., linha 10). Já a variável nr (linha 9), do mesmo tipo, mas **var** ao invés de **fresh**, é utilizada para armazenar valores recebidos (e.g., linha 11).

Cada evento de envio (**send_1**) possui um evento de recebimento (**recv_1**) cor-

Algoritmo 1: Algoritmo Needham-Schroeder na semântica da Scyther

```
1  const pk: Function;
2  secret sk: Function;
3  inversekeys (pk,sk);
4  const Eve: Agent;
5  untrusted Eve;
6  protocol ns(Alice,Bob,Eve){
7     role Alice{
8        fresh ni: Nonce;
9        var nr: Nonce;
10       send_1(Alice,Bob,{Alice,ni}pk(Bob));
11       recv_2(Bob,Alice,{ni,nr}pk(Alice));
12       send_3(Alice,Bob,{nr}pk(Bob));
13       claim(Alice,Secret,ni);
14       claim(Alice,Secret,nr);
15       claim(Alice,Nisynch);
16    }
17    role Bob{
18       var ni: Nonce;
19       fresh nr: Nonce;
20       recv_1(Alice,Bob,{Alice,ni}pk(Bob));
21       send_2(Bob,Alice,{ni,nr}pk(Alice));
22       recv_3(Alice,Bob,{nr}pk(Bob));
23       claim(Bob,Secret,ni);
24       claim(Bob,Secret,nr);
25       claim(Bob,Nisynch);
26    }
27 }
```

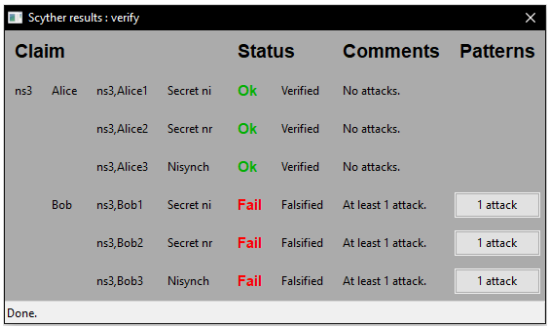
respondente (e.g., linhas 10 e 20). A sintaxe do evento **send_1** (linha 10) indica que a transmissão é de Alice para Bob, os dados enviados, cifrados com a chave pública do Bob $pk(Bob)$, são "*Alice*"e ni. No agente Bob (linha 20), há um evento com sintaxe idêntica, cuja única diferença é o tipo, i.e., **recv** ao invés de **send**. Na sequência, Bob envia os valores ni e nr para Alice (linha 21). Alice recebe a resposta (linha 11), verificar o valor do ni e envia nr para Bob (linha 12), que recebe e verifica o valor (linha 22).

Finalmente, as afirmações **claim** (linhas 13, 14, 15 23, 24, 25) definem os requisitos de segurança do protocolo. No caso, as afirmações criadas verificam se os *nonces* gerados por Alice (ni) e Bob (nr) permanecem secretos durante as comunicações (claim **Secret**) e se as mensagens são de fato trocadas entre eles apenas (claim **Nisynch**).

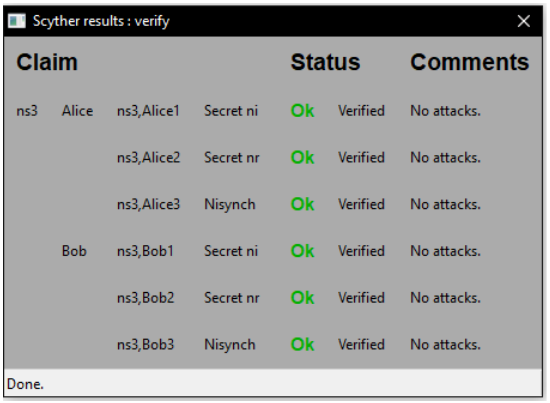
4. Análise do Protocolo com a Ferramenta Scyther

Ao analisar o código do Algoritmo 1 na ferramenta Scyther, é gerado um relatório que aponta a existência de falhas no protocolo, como pode ser visto na Figura 2a. Na coluna **Claim** são apresentadas quatro informações, o protocolo testado (*ns*), os agentes analisados (Alice e Bob), um par de valores que servem como um identificador único para cada evento (e.g., $ns, Alice1$) e o evento de afirmação (e.g., $Secret\ ni$). Nas colunas **Status**, **Comments** e **Patterns** são reportados os eventuais ataques que o protocolo é suscetível. Como pode ser observado, há pelo menos um ataque no protocolo Needham-Schroeder, que afeta os eventos $ns, Bob1$ e $ns, Bob2$. Nestes casos, a ferramenta gera também um grafo com os passos de execução do(s) ataque(s), como ilustrado na Figura 3.

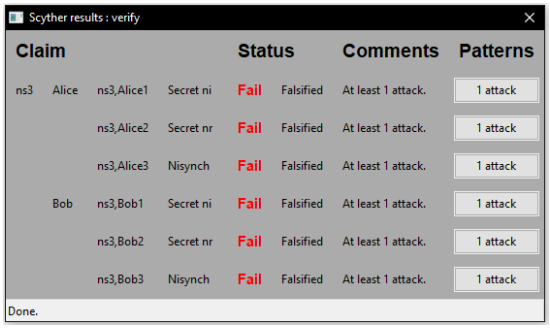
Para comprometer a comunicação entre Alice e Bob, Eve (agente malicioso) precisa apenas convencer Alice de que é Bob. Alice envia a Eve uma mensagem $\{Alice, ni\}$ cifrada com a chave pública $pkEve$. Como Eve possui a chave privada correspondente, ele consegue ler o conteúdo da mensagem. Eve então cifra e envia a mensagem para Bob. Bob, sem desconfiar, pois recebeu uma mensagem cifrada com sua chave pública, responde ao atacante com a mensagem $\{ni, nr\}pkAlice$. O agente malicioso simplesmente encaminha a mensagem para Alice. Para confirmar o recebimento do $nonce\ nr$, Alice responde ao atacante com $\{nr\}pkEve$. O atacante decifra a mensagem, descobre o valor de nr, cifra e envia a mensagem para Bob. Bob confirma o valor de nr (autenticação)



(a) **Verificação do protocolo Needham-Schroeder**



(b) **Verificação do protocolo corrigido**



(c) **Comprometimento da chave secreta**

Figura 2. Resultados gerados pela ferramenta Scyther.

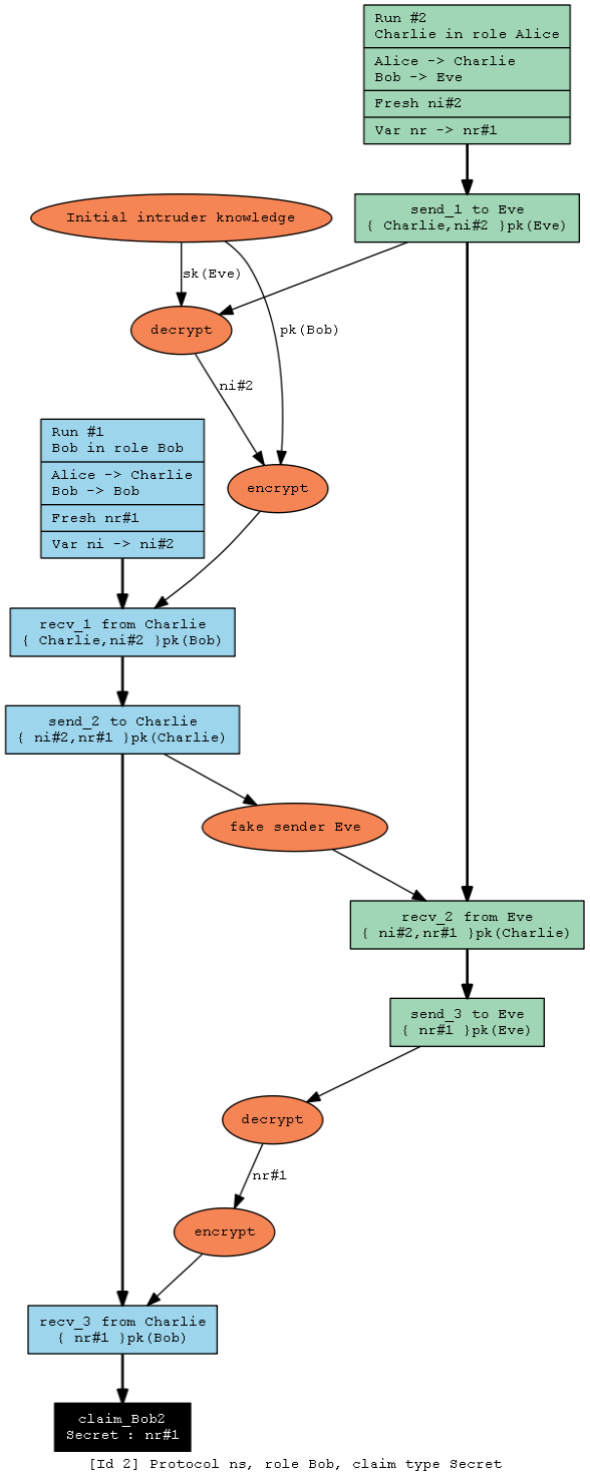


Figura 3. Diagrama do ataque ao protocolo gerado automaticamente pela ferramenta.

e acredita que está comunicando-se com Alice. A partir deste ponto, todas as mensagens trocadas entre Alice e Bob passam primeiro pelo atacante. Observem que o ataque é

simples. Eve precisa apenas estar entre Alice e Bob e conhecer a chave pública de Bob.

Para corrigir esta falha do protocolo, Bob deve adicionar a sua identidade na resposta à primeira mensagem de Alice (ni, nr, Bob na linha 20 do Algoritmo 1). Com isto, Alice consegue descobrir que a identidade contida na mensagem é diferente da identidade de Eve, para quem está enviando as suas mensagens. Neste caso, Alice simplesmente encerra a troca de mensagens. Com o algoritmo corrigido (linha 20), o resultado da análise da Scyther pode ser visto na Figura 2b. Como pode ser observado, um simples detalhe de especificação pode comprometer toda a segurança do protocolo. Isto demonstra o quão importante é a utilização de ferramentas de verificação automática de protocolos.

Para observar o que ocorre no protocolo, é possível definir um agente não-confiável Eve ("$compromised\ sk(Eve);$"), capaz de comprometer as chaves privadas dos demais agentes. Tomando como exemplo o Algoritmo 1, a ferramenta considera a existência de um agente malicioso que conhece as chaves secretas de Alice e Bob. Isto permite que o atacante comprometa as comunicações, conforme a Figura 2c.

5. Conclusão

Este trabalho apresentou uma introdução à ferramenta de verificação automática de protocolos Scyther, utilizando como exemplo o protocolo Needham-Schroeder. Este protocolo possui uma falha de segurança grave, que pode ser facilmente detectada e corrigida com a utilização da Scyther. Este exemplo prático real, destaca a importância da disseminação do conhecimento e da utilização de ferramentas de verificação formal de protocolos.

Referências

Affeldt, R. and Marti, N. (2013). Towards Formal Verification of TLS Network Packet Processing Written in C. In *7th PLPV*, pages 35–46. ACM.

Armando, A., Basin, D., Boichut, Y., Chevalier, Y., Compagna, L., Cuéllar, J., Drielsma, P. H., Héam, P.-C., Kouchnarenko, O., Mantovani, J., et al. (2005). The AVISPA tool for the automated validation of internet security protocols and applications. In *International conference on computer aided verification*, pages 281–285. Springer.

Blanchet, B., Smyth, B., Cheval, V., and Sylvestre, M. (2018). ProVerif 2.00: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial. `https://bit.ly/2OUlH7f`.

Chudnov, A., Collins, N., Cook, B., Dodds, J., Huffman, B., MacCárthaigh, C., Magill, S., Mertens, E., Mullen, E., Tasiran, S., Tomb, A., and Westbrook, E. (2018). Continuous Formal Verification of Amazon s2n. In *Computer Aided Verification*, pages 430–446.

Cremers, C., Horvat, M., Scott, S., and v. d. Merwe, T. (2016). Automated Analysis and Verification of TLS 1.3: 0-RTT, Resumption and Delayed Authentication. In *IEEE SP*.

Cremers, C. J. F. (2006). *Scyther: Semantics and verification of security protocols*. Eindhoven University of Technology Eindhoven.

Dalal, N., Shah, J., Hisaria, K., and Jinwala, D. (2010). A comparative analysis of tools for verification of security protocols. *Int. J. of Comm., Network and System Sciences*, 3(10):779.

Kreutz, D., Yu, J., Ramos, F. M. V., and Esteves-Verissimo, P. (2019). ANCHOR: Logically centralized security for software-defined networks. *ACM Trans. Priv. Secur.*, 22(2):8:1–8:36.

Li, L., Sun, J., Liu, Y., Sun, M., and Dong, J. (2018). A Formal Specification and Verification Framework for Timed Security Protocols. *IEEE Trans. on Soft. Engineering*, 44(8):725–746.

Lowe, G. (1995). An Attack on the Needham- Schroeder Public- Key Authentication Protocol. *Information processing letters*, 56(3).

Needham, R. M. and Schroeder, M. D. (1978). Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999.