정형 명세를 통한 보안 프로토콜 코드 생성

전철욱[0*], 김일곤[*], 최진영[*], 김상호[**], 노병규[**]
[*]고려대학교 컴퓨터학과
{cwjeon, igkim, choi}@formal.korea.ac.kr
[**]한국정보보호진흥원
{shkim, nono}@kisa.or.kr

Automatic Implementation of Security Protocol Code from Formal Specification

Chul-wuk Jeon[0*], Il-gon Kim[*], Jin-Young Choi[*]
[*]Dept of Computer Science & Engineering, Korea University

Sang-Ho Kim[**], Byung-Gyu Nho[**]
[**]Korea Information Security Agency

요약

컴퓨터 통신이 확대되면서 심각하게 대두된 문제 중 하나는 보안 프로토콜의 설계와 구현이라 할 수 있다. 현재 안전한 보안 프로토콜을 설계하기 위해 정형 기법을 적용하여 검증하는 연구가 많이 진행되고 있다. 하지만 프로토콜을 설계 할 때 나타날 수 있는 보안적 취약 사항들을 정형 기법을 이용하여 제거한다 하더라도 구현된 프로토콜상에서 프로그래머의 코딩 실수나 프로그램 언어의 특성상 보안 취약점이 존재할 수 있다. 따라서 보안 프로토콜 구현시 나타날 수 있는 문제를 해결하기 위해 정형 검증된 프로토콜을 실제 구현 코드를 생성할 수 있는 도구의 필요성이 높아지고 있다. 본 논문에서는 Casper에서 보안 프로토콜을 검증한 후 검증된 프로토콜을 AISP-C2에 입력하여 C#으로 구현 코드를 자동 생성하도록 하고 정형 검증에서 검증할 수 없는 실제 컴퓨팅 환경에서 발생할 수 있는 보안성 취약점을 제거하기 위한 기능을 추가하였다.

1. 서론

컴퓨터 통신이 확대되면서 심각하게 대두된 문제 중 하나는 보안 프로토콜의 설계와 구현이라 할 수 있다. 보안 프로토콜의 설계는 통신의 부하, 암호화를 위한 계산, 수학적 검증등과 같은 시스템과 시스템의 환경에 맞는 설계를 해야 하기 때문에 많은 어려움이 존재한다. 특히 보안 프로토콜의 보안성 검증은 중요한 문제로서 Needham-Schroeder Protocol의 경우 보안적 취약점을 존재하고 있었으나 설계시 이를 알지 못했다. 현재 이러한 난점을 극복하기 위해 정형 기법을 적용한 연구가 많이 진행되고 있다. 보안 프로토콜을 검증할 수 있는 정형 기법으로는 귀납적 증명, 모델 체킹, 유한 상태 검색등이 있다[7]. 하지만 프로토콜을 설계 할 때 나타날 수 있는 보안적 취약 사항들을 정형 기법을 이용하여 제거한다 하더라도 구현된 프로토콜상에서 프로그래머의 코딩 실수나 프로그램 언어의 특성상 보안 취약점이 존재할 수 있다. 실제로 2001년 SSL 프로토콜(version1)에서 발생한 버퍼 오버플로우는 설계상으로는 안전한 보안 프로토콜이 구현시 발생할 수 있는 문제점을 보여준 대표적인 예이다. 이러한 문제를 극복하기 위해 정형 기법으로 검증된 프로토콜을 실제 코드로 자동으로 생성해 주는 연구가 현재 진행되고 있다[4-6]. 안전한 코드를 자동으로 생성해 줌으로 구현시 나타날 수 있는 문제점을 제거할 뿐만 아니라 시간적, 경제적으로도 많은 이점이 있다. 본 본문은 2장에서 Casper를 소개하며 3장에서 AISP-C2에 관해 설명하고 마지막 4장에서는 결론 및 향후 연구에 대해 논하도록 하겠다.

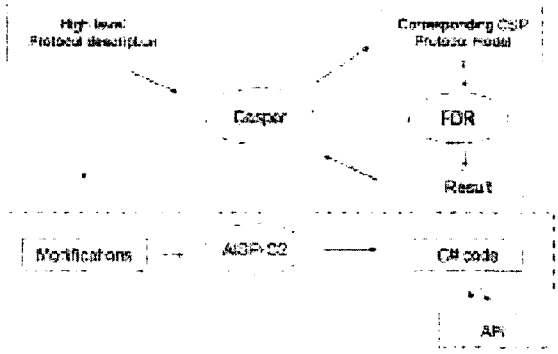
2. Casper 소개 (A Compiler for the Analysis of Security Protocols)

보안 프로토콜을 검증하기 위해 현재 널리 사용되고 있는 방법으로 프로세스 알제브라 CSP[1]로 모델을 명세하고 모델 체커인 FDR[2]로 모델을 검증하는 방법이 있다. 하지만 CSP로 보안 프로토콜을 명세하기 위해서는 많은 시간이 걸리며 사소한 실수로 인해 전체 프로토콜의 명세가 잘못되는 일이 발생할 뿐만 아니라 CSP를 잘 알아야 명세할 수 있다는 단점이 있다. 이러한 단점을 극복하기 위해 검증자가 추상적인 단계에서 쉽게 명세하면 CSP 코드를 생성해 주는 컴파일러가 Casper[3]이다. 즉, Casper를 이용하여 보안 프로토콜의 행위를 보다 쉽게 명세하고 자동으로 CSP 코드를 생성한 후 FDR 도구를 이용하여 보안 프로토콜의 안전성을 검증한다. 본 논문의 페이지 사정상 CSP, Casper 및 FDR 도구에 대한 상세한 설명은 [1-3] 논문을 참조하길 바란다.

3. AISP-C2(Automatic Implementation of Security Protocol Code from Casper)

3.1 개요

Casper를 이용하여 명세한 보안 프로토콜을 실제 구현 코드로 생성하는 과정은 [그림 1]과 같이 나타낼 수 있다. 검증하고자 하는 보안 프로토콜을 Casper에 입력언어로 명세하여 Casper에 입력하면 프로세스 알제브라 CSP로 모델화 된 코드를 출력한다. 이 코드를 모델 체커인 FDR에 입력하면 명세된 프로토콜이 보안적 취약적이 있는가를 출력한다.



[그림 1] 전체적인 프레임

만약 취약점이 있다면 명세된 프로토콜을 수정한 후 다시 Casper로 통하여 같은 방법으로 FDR로 검증한다. [그림 1]의 상단 부분이 이에 해당한다. 만약 취약점이 없다면 AISP-C2에 입력할 수 있도록 Casper 입력 언어를 약간의 변경하여 AISP-C2에 입력한다. AISP-C2은 보안 프로토콜을 C#코드로 생성해 준다. [그림 1]의 하단 부분이 이에 해당한다.

3.2 구조

AISP-C2 구조는 다음과 같이 크게 두 부분으로 나눌 수 있다.

- Protocol Notation - Casper에 입력 언어로 이해하기 쉽게 메시지 순서와 메시지 종류등을 추상화한 부분을 실제 C# 코드로 생성하는 부분이다.

- AISP-C2 API - 통신과 암호화에 관련한 메쏘드를 정의하고 구현한 부분으로 생성된 C#코드에서 호출하는 사용할 수 있도록 되어있다. [그림 1]의 아래 API가 이에 해당한다.

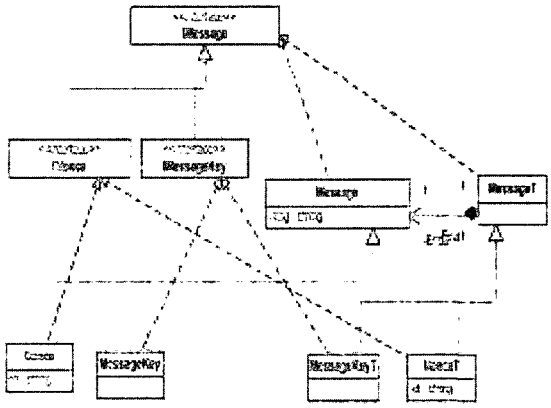
Haskell로 만들어진 컴파일러가 수정된 Casper 입력 언어를 받아 들여 C#코드 생성하고 이 코드는 AISP-C2 API을 호출하여 통신과 암호화 수행한다.

3.3 C#과 AISP-C2 API 구조

현재 관련된 연구의 도구들이 모두 JAVA로 된 코드를 생성하고 있다. 하지만 실제 보안 프로토콜을 사용하는 시스템은 JAVA뿐만 아니라 여러 가지 프로그램 언어로 만들어 지고 있다. 따라서 JAVA로 된 코드의 생성은 사용할 수 있는 한계를 가질 수 있고 이를 다른 프로그램 언어로 확장해야 할 필요성이 존재한다. C#은 자바가 가지고 있는 보안적 특징을 가지고 있을 뿐만 아니라 코드 액세스 보안, 역할 기반 보안등을 여러 가지로 강력한 보안을 지원하고 있다. 또한 마이크로소프트사의 .NET기반의 환경은 CLS(Common Language Specification)와 CTS(Common Type System)를 이용하여 .NET 기반에서 서로 다른 프로그램 언어로 만들어진 코드를 재사용 할 수 있도록 되어있다. 다시 말해 .NET에서 C#으로 만들어진 클래스는 .NET에서 C++, Visual Basic, JScript에서 사용될 수 있기 때문에 가용성이 높다 할 수 있다. 따라서 C#으로 구현된 API는 C++, Visual Basic에서도 사용할 수 있다는 장점이 있다. AISP-C2의 API의 구조는 [그림 2]와 같다. 그림에서 볼 수 있듯이 Nonce, Key, Message는 IMessage라는 인터페이스를 구현한다. IMessage는 Message가 가지고 있는 암호화, 복호화등 기반행위를 수행한다.

3.4 키생성기(KeyGenerator)

보안 프로토콜에서 키의 생성 및 관리는 가장 중요한 부분 중에 하나이다. 키를 생성할 때 일정한 규칙이나 패턴이 존재하면 아무리 암호화를 잘한 메시지라고 해도 공격 당하기가 쉽다. 또한 키를 아무리 잘 생성하였다 하더라도 관리를 잘 못할 경우도 마찬가지로 공격 당하기 쉽다. 그래서 AISP-C2에서는 키생성기를 만들어 키를 생성할 때 생성할 때마다 랜덤하게 생성하였으며 xml파일로 저장하도록 하였다 AISP-C2에서 생성된 코드는 키생성기에서 생성한 키를 이용하여 실행하도록 하였다.



[그림2] API 를 구성하는 클래스 다이어그램

3.5 도구의 장점

Casper에서 명세하는 보안 프로토콜은 실제 프로토콜과는 여러 가지 차이점이 있다. Casper에서는 Protocol Notation에 대하여 탁월한 검증을 하는 반면 Casper의 기능 한계상 다중 프로토콜 공격등 몇 가지 실제 컴퓨팅 환경에서 나타날 수 있는 공격에 대하여는 검증하지 못하는 경우가 있다. 본 논문에서 소개하고 있는 AISP-C2을 이용해서 자동으로 생성된 보안 프로토콜 구현 코드는 다음과 같은

장점을 가지고 있다.

- 버퍼오버플로우 방지 - C#언어의 특성상 버퍼의 크기를 자동으로 체크하여 버퍼오버플로우로 인해 발생할 수 있는 보안상의 취약점을 사전에 방지할 수 있다.

- 키 추측공격(Key Guessing Attack) - 8바이트 이상의 키를 랜덤하게 생성하기 때문에 공격자의 키 추측공격을 방지할 수 있다.

- 랜덤 포트 생성 - 명세된 보안 프로토콜을 컴파일 할 때 이 기능 체크함으로써 생성된 코드에서는 랜덤한 포트를 통하여 통신이 가능하도록 하였다. 통신을 할 때 포트를 변경하므로 기존의 고정된 포트보다는 안전성을 좀더 높일 수 있다.

- 수신 메시지 지우기 - 컴파일 시 이 기능을 체크할 경우 통신을 하면서 수신 메시지를 그대로 가지고 있을 경우 Relay 공격을 할 수 있기 때문에 수신 메시지를 지우도록 하였다.

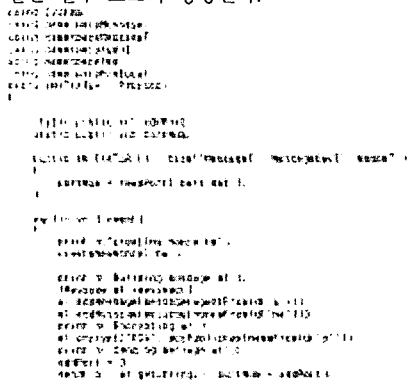
3.6 Needham-Schroeder Protocol 예제

Casper로 명세하고 검증한 Needham-Schroeder Protocol을 AISP-C2에 입력으로 하여 코드를 생성한 후 실제 시스템에서 실행하도록 하였다. Needham-Schroeder Protocol은 통신하는 두 에이전트가 서로를 인증하는 프로토콜이다.

Message 1. a -> b : {a, na}PK(b)
Message 2. b -> a : {na, nb}PK(a)
Message 3. a -> b : {nb}PK(b)

a, b는 에이전트고 PK(x)는 에이전트 x의 public Key로 암호화한 것이다. na, nb는 각각 에이전트의 Nonce가 된다. 이 프로토콜에서는 다음과 같은 요구 사항을 가지고 있다; a는 b로부터 인증을 받아야 한다. b는 a로부터 인증을 받아야 한다. na, nb를 a,b만 알고 있어야 한다.

이러한 요구 사항을 가지고 있는 보안 프로토콜을 Casper로 검증하고 AISP-C2으로 컴파일하면 [그림3]과 같은 일부 코드가 생성된다.



[그림3] 생성된 일부 코드

생성된 코드와 키생성기로 생성된 키를 이용하여 두 에이전트로 구성된 프로토콜을 실행시키면 [그림4]와 같은 결과를 볼 수 있다.

[그림4] 실행 결과

4. 결론 및 향후 연구

정형 검증된 보안 프로토콜에 대한 코드 생성은 실제 구현 과정에서 생길 수 있는 보안적 취약점을 막아줄 뿐만 아니라 경제적, 시간적으로 많은 이점이 있다. 본 논문에서는 Casper와 FDR을 이용하여 검증된 안전한 보안 프로토콜로부터 자동으로 C# 구현 코드를 생성함으로써 구현상에서 생길 수 있는 프로그래머의 실수나 프로그램 언어의 보안 취약점을 방지할 수 있다. 향후 연구과제로는 구현한 코드가 안전한가를 검증해야 하며, 보다 풍부한 명세가 가능하도록 확장해야 할 것이다.

참고 문헌

[1] C.A.R. Hoare, Communicating Sequential Processes, Prentice-Hall, 1985.
[2] Formal Systems(Europe) Ltd. Failure Divergence Refinement-FDR2 User Manual, Aug. 1999.
[3] Gavin Lowe, "Casper A Compiler for the Analysis of Security Protocols", 1993.
[4] Benjamin Tobler, "Generating Network Security Protocol Implementations from Formal Specifications", 2004.
[5] Dawn Song, "AGVI- Automatic Generation, Verifications, and Implementation of Security Protocols", 2001.
[6] Davide Pozza, "Spi2Java: Automatic Cryptographic Protocol Java Code Generation from spi calculus", 2004.
[7] Jonathan Millen, "Cryptographic Protocol Generation From CAPSL", 2001.