# 패킷 필터링 기능 테스트를 위한 테스트 도구 개발

## (Development of Test Tool for Testing Packet Filtering Functions)

김 현 수 † 박 영 대 †† 국 승 학 †††
(Hyeon Soo Kim) (Young Dae Park) (Seung Hak Kuk)

**요 약** 패킷 필터링이란 악의적인 네트워크 패킷들을 거르는 작업을 말한다. 패킷 필터링의 기능을 테스트하기 위해서는 구축된 보안 정책이 의도했던 대로 정확하게 동작하는가를 검증하여야 한다. 그러나 이런 기능을 테스트하기 위한 도구들은 많지 않으며, 테스트 과정에서 많은 사용자의 노력을 요구한다. 대부분의 보안 관리자들은 보안 정책을 새로 수립하거나 기존의 보안 정책을 수정할 때에 새로운 보안 정책을 체계적으로 테스트하는 것에 부담을 느낀다. 이런 부담을 경감해주기 위해 우리는 사용자의 참여를 최소화하는 새로운 테스트 방법을 제안한다. 제안하는 방법은 테스트 케이스와 테스트 오라클의 생성을 자동화한다. 자동으로 생성된 테스트 케이스는 테스팅 과정의 입력 요소를 선택해야 하는 고민을 덜어주며, 자동으로 생성된 테스트 오라클은 사용자의 도움 없이 테스트 결과에 대한 판단을 가능하게 한다. 우리의 테스트 방법을 구현한 테스트 도구는 테스트 수행의 전체 4단계 중 테스트 준비, 테스트 실행, 테스트 평가의 3단계에 걸쳐 테스트 자동화를 실현하고 있다. 이런 테스트 도구 위에서 테스팅을 수행하게 된다면 결과적으로 테스트 활동의 신뢰도를 보다 높게 향상시킬 수 있다. 이 논문은 우리의 테스트 방법과 테스트 도구의 설계 및 구현에 관한 내용을 기술한다.

**키워드** : 보안 시스템 기능 테스트, 패킷 필터링, 보안 정책, 테스트 오라클, 방화벽

***Abstract*** Packet filtering is to filter out potentially malicious network packets. In order to test a packet filtering function we should verify whether security policies are performed correctly as intended. However there are few existing tools to test the function. Besides, they need user participation when generating test cases or deciding test results. Many security administrators have a burden to test systematically new security policies when they establish new policies or modify the existing ones. To mitigate the burdens we suggest a new test method with minimal user participation. Our tool automates generation steps of the test cases and the test oracles, respectively. By using the test oracles generated automatically, deciding test results is possible without user intervention. Our method realizes an automatic testing in three phases; test preparation phase, test execution, and test evaluation. As a result it may enhance confidence of test activities more highly. This paper describes the design and implementation of our test method and tool.

**Key words** : Security systems' function test, Packet filtering, Security policy, Test oracle, Firewall

## 1. 서 론

네트워크가 개방되고 인터넷이 급속히 성장함에 따라 정보 보안 시스템은 갈수록 중요하게 되었다[1]. 외부 침입에 대비한 효율적인 정보 보안 시스템을 구축 운영하려면 우선적으로 선행되어야 할 것이 조직에 맞는 정보 보안 정책 수립과 정당한 정책 수행이다. 대부분의 조직은 라우터(router)나 방화벽(firewall)과 같은 보안 솔루션을 이용하여 조직에 맞는 보안 정책을 수립한다[2].

라우터나 방화벽은 대부분 패킷 필터링 기능을 기본으로 가지고 있으며 이 기능을 이용하여 접근 제어를 실현하고 있다. 접근 제어 기술은 라우터나 방화벽에서

† 종신회원 : 충남대학교 컴퓨터공학과 교수
hskim401@cnu.ac.kr
†† 정 회 원 : 삼성전자주식회사 S/W Lab 연구원
yd7.park@samsung.com
††† 정 회 원 : 충남대학교 컴퓨터공학과
triple888@cnu.ac.kr
논문접수 : 2006년 1월 25일
심사완료 : 2007년 3월 13일

통과시켜야 할 트래픽과 차단해야 할 트래픽을 통제하기 위한 방법으로 접근 제어 리스트(Access Control List)를 이용한다. 이 리스트에는 IP 헤더의 근원지 주소/포트, 목적지 주소/포트, 프로토콜에 대한 정보가 있어 이를 이용하여 트래픽을 통과(accept) 또는 차단(drop) 시키는 기능을 수행한다. 이러한 보안 솔루션에서 보안 정책이 올바르게 수립되었는지 혹은 보안 시스템이 정상적으로 작동하는지를 테스트하는 것은 보안 시스템을 이용하고자 하는 조직뿐만 아니라 보안 시스템 개발자들에게도 매우 중요하다[2-4].

이 논문에서는 라우터나 방화벽과 같은 보안 솔루션에서 수립된 보안 정책뿐만 아니라 라우터 등과 같은 네트워크 장비에 탑재될 보안 시스템을 모두 테스트 할 수 있는 테스트 방법과 테스트 도구를 제안한다. 이는 기존의 완성된 보안 시스템을 위한 테스트 방법 및 도구로도 사용될 수 있지만 보안 시스템을 새롭게 개발하거나 기존의 보안 정책을 새롭게 변경할 때에도 그 기능이 정확하게 동작하는지 테스트하기 위해 사용할 수 있다.

실제로 조직의 보안 담당자들은 접근 제어 리스트 등을 이용하여 다양한 보안 정책을 수립할 수 있는데 이렇게 수립한 보안 정책이 제대로 동작하는지 확인할 수 있는 테스트 도구는 흔하지 않았다. 또한 존재하는 도구들도 그 수행 과정에서 보안 솔루션 사용자의 개입이 많이 요구되었다. 즉, 테스트 케이스의 선정 과정에서 사용자가 테스트 케이스를 추출하거나 테스트 결과를 판단하기 위한 과정에서 네트워크를 통해 보내진 패킷과 받아진 패킷의 로그를 사용자가 개입하여 분석하는 방법을 취하였다. 따라서 사용자의 선택과 판단에 따라 테스트 결과의 신뢰성에 기복이 심하게 나고 또한 사용자가 테스트 도구뿐만 아니라 테스트 방법에 대해서도 많은 지식을 갖고 있어야 한다는 단점이 있었다.

본 논문에서는 네트워크 보안 관리자의 의도대로 패킷 필터링의 기능이 정확하게 동작하는지 테스트하기 위한 기능 테스트 방법을 제안한다. 특히 앞서 언급한 문제점들을 개선하기 위해 보안 정책을 기술하는 규칙을 분석하여 테스트 케이스와 테스트 오라클을 자동으로 생성하는 방법을 제안하고, 테스트 준비, 실행, 평가 과정을 자동화하는 시스템을 제안하고 구현한다. 본 시스템을 통해 테스트 수행 과정에서 보안 시스템 사용자들의 개입을 최소화할 수 있으며, 결국 보안 담당자들이 테스트에 대한 많은 지식이 없어도 짧은 시간 내에 쉽고 편리하게 수립된 보안 정책이나 보안 시스템의 기능 테스트를 수행할 수 있다. 또한 그 수행의 결과도 체계적인 분석 과정으로 인해 훨씬 높은 신뢰도를 확보할 수 있게 되었다.

본 논문의 구성은 다음과 같다. 2장에서는 방화벽의 보안 기술, 패킷 필터링의 개념, 기존의 패킷 필터링 테스트 방법 등을 살펴보고, 3장에서는 제안하는 테스트 도구의 구성을 기술하며, 4장에서는 제안하는 체계적인 테스트 방법에 대하여 자세히 기술한다. 즉 테스트 대상과 각 테스트 대상에 대하여 수행할 테스트 방법, 그 때에 적용될 테스트 케이스 자동 생성 방법을 기술한다. 또한 테스트 수행 후 테스트 결과를 평가하기 위한 테스트 오라클의 자동 생성에 대해서도 기술한다. 5장에서는 제안한 방법을 실제 시스템에 적용시켜 본 실험 결과에 대해 기술하고, 6장에서 결론 및 향후 연구 방향에 대해 기술한다.

## 2. 관련 연구

### 2.1 패킷 필터링

패킷 필터링은 네트워크에서 지나가는 패킷의 헤더 정보를 분석해 그 전체 패킷의 진입/진출을 제어한다. 이것은 해당하는 패킷의 헤더 정보를 보고 'DROP'(즉, 패킷 진입/진출 거부) 하던가, 'ACCEPT'(즉, 패킷 진입/진출 허락) 하던지 또는 다른 행동을 할 것인가를 결정하는 방법이다[2]. 이를 이용하면 내부 네트워크에서 다른 네트워크로 인터넷을 이용하여 접속을 하고자 할 때 어떤 형태의 전송은 가능하게 하고 다른 것은 불가능하게 할 수 있다. 예를 들어, 패킷 헤더는 목적지의 주소를 포함하고 있는데 이 정보를 바탕으로 이 패킷을 외부 네트워크의 다른 곳으로 전달하지 않을 수 있다. 또한 보안 관점에서 악의적인 정보나 불확실한 의도를 지닌 패킷들을 내 컴퓨터에 들어오기 전에 미리 차단할 수도 있다.

이러한 패킷 필터링의 가장 기본적인 형태는 IP헤더와 Protocol 헤더의 정보를 이용해 미리 설정된 보안정책에 따라 패킷을 통과시킬지 말지를 결정하는 상태없는(stateless) 패킷 필터링이다. 상태없는 패킷 필터링은 정적인(static) 패킷 필터링이라고도 하며 현재 출시되는 대부분의 라우터에는 기본 기능으로 내장되어 있으며, 상용 방화벽 제품들도 어떤 형태로든 패킷 필터링 기능을 기본적으로 제공하고 있다. 이러한 정적인 패킷 필터링은 최소한의 액세스 규칙만을 적용시켜 프로세싱을 하므로 가장 빠른 속도를 낼 수 있다. 그리고 구현이 간단하며, 사용자에게는 투명하게 동작한다는 장점이 있다. 그러나 복잡한 네트워크나 서비스에 대한 정교한 액세스 규칙을 구현하기가 매우 어렵다. 또한, 제한된 정보만을 사용하므로 간단한 액세스 규칙을 갖는 접속 제어 기능만을 제공한다. 따라서 일단 패킷 필터를 통과한 위험한 패킷으로부터 내부 자원을 효과적으로 보호할 수 없다는 취약점을 가진다.

상태 있는 검사(stateful inspection)는 정적 패킷 필터링의 발전된 형태로 동적 패킷 필터링이라고도 하며 네트워크 계층뿐만 아니라 응용 계층까지 모든 계층에서 패킷을 검사해 패킷을 필터링할 수 있다. 이는 정적인 패킷 필터링의 기능뿐만 아니라 상태 정보를 지속적으로 유지하여 이를 바탕으로 현재 패킷의 통과 여부를 결정할 수 있다. 예를 들어, 해당 연결의 첫 패킷만을 액세스 규칙과 비교하여 통과 여부를 결정하고, 이 사실을 상태 정보에 추가하여, 그 연결의 후속 패킷들은 상태 정보에 따라 통과시키거나 거절하고, 연결이 끝나면 자동적으로 해당 상태 정보를 삭제한다. 이러한 상태 있는 검사는 정적인 패킷 필터링에 비해 정교한 접근 제어가 가능하다. 하지만 정적인 패킷 필터링에 비해 구현하기가 어렵다. 또한 모든 연결에 대한 상태 정보를 유지해야 하며, 모든 패킷에 대한 상태 정보 검사를 수행하기 때문에 성능이 떨어진다는 단점이 있다.

### 2.2 패킷 필터링 시스템 구현 방법

#### 2.2.1 Libpcap 기반 방법

Libpcap은 패킷의 길이, 내용, 헤더 정보를 이용해 패킷을 검사할 수 있는 라이브러리이다. 이 라이브러리의 기능은 패킷을 필터링하거나, 분류하는데 사용될 수 있다. 일반적인 소프트웨어 보안 애플리케이션들은 Libpcap과 같은 필터링 라이브러리를 이용한다. Snort는 Libpcap을 이용해 네트워크 침입 탐지 시스템을 구현한 대표적인 예이다. Snort는 IP 네트워크와 프로토콜 분석, 패킷의 데이타 검색을 통해 추적 분석과 패킷 로그 분석의 기능을 제공하며, 비정상적인 패킷의 탐색 기능을 제공한다. 패킷의 데이타를 검색하는 것은 비정상적인 패킷을 탐색하는데 중요한 메커니즘이다. Libpcap을 이용한 또 다른 프로그램으로 Hogwash가 있다. Hogwash는 두 네트워크 사이에서 변경된 패킷이나 손실된 패킷을 탐색하는 기능을 수행한다[5].

#### 2.2.2 규칙 기반 방법

패킷 필터링을 위해 일련의 규칙을 이용하는 방법이 있다. 이 방법은 일반적으로 발생하는 공격 패턴이나 위험한 패턴을 분석해 각각의 패턴에 맞는 필터링 규칙을 설정하여 패킷을 필터링한다. 간단한 규칙을 설정함으로써 보안 시스템에 융통성과 편리성을 허락할 수 있다. 대표적인 예로 리눅스 기반의 라우터에서 사용되는 IPtables가 있다[6,7]. IPtables에 필터링 규칙을 설정함으로써 비정상적인 패킷을 필터링할 수 있다. Snort에서도 필터링 규칙을 설정함으로써 일반적으로 발생할 수 있는 비정상적인 패킷을 추적 할 수 있다. Snort는 모듈식의 플러그인 형태의 탐색 엔진을 이용한다. 이를 통해 규칙의 생성과 변경을 손쉽게 할 수 있다. 마지막으로 ACL(Access Control List)이 있다. 이는 라우터와 같은 통신 보안 장치에서 호스트의 서비스 이용을 부분적으로 허용 또는 거절을 통해 조정하는 데 쓰인다. IPtables, Snort 그리고 ACL에서는 일련의 규칙 리스트를 통해 각각의 패킷을 검사하고 다양한 방법으로 패킷을 제어한다.

### 2.3 패킷 필터링 테스트

패킷 필터링 테스트는 방화벽이나 라우터의 보안 정책, 패킷 필터링 규칙을 테스트하기 위한 방법이다. 이는 지정된 규칙에 맞게 패킷이 정확하게 필터링 되는지, 즉 패킷 필터와 이를 이용한 보안 정책에 의해 시스템이 안전하게 보호될 수 있는가를 테스트하기 위한 것이다. 기존의 많은 연구들의 결과로 다양한 유형의 방화벽 테스트 도구들이 존재한다. 그들 중 일부는 SPAK, ipsend, 또는 Ballista 등과 같이 네트워크 트래픽 생성기로 분류된다. 이 도구들은 다양한 종류의 트래픽을 생성함으로써 성능 측면에서 방화벽의 능력을 테스트 할 수 있다[8]. Network Monitor는 네트워크를 모니터링하기 위한 도구이다. 그것은 서버나 네트워크 연결, 또는 다른 장치들의 과부하 또는 고장으로 인해 발생하는 문제 측면에서 네트워크를 감시한다[8]. Nmap은 타겟 호스트의 포트를 검사하고, 어떤 서비스와 관계가 있는지 알아보는 포트 스캐너이다. 이 도구는 종종 네트워크 운영자에 의해 주어진 호스트 상에서 공격에 취약한 포트 번호를 스캔하는 방법으로 네트워크 보안을 검사하는데 사용된다[6]. Nessus 역시 취약성 검출 도구이다. 이 도구는 수행중인 서비스가 알려진 공격에 대하여 취약한지 여부를 평가한다. 만일 새로운 취약성이 공개된다면, 네트워크를 공격하는 어떤 것을 검사하고 보고하기 위해 새로운 버전의 Nessus가 요구된다[6]. 이런 도구들 각각은 네트워크 보안이라는 그 나름대로의 목적에 맞게 중요한 역할을 수행하였다. 그러나 상대적으로 라우터나 방화벽 같은 보안 시스템의 기본 기능인 패킷 필터링에 대한 테스트 노력들은 간과되어 대부분 수동으로 이루어졌거나 몇몇 기능이 빈약한 테스트 도구들만 존재할 뿐이다.

자동화된 패킷 필터링 테스트 방법은 기존에 알려진 침투 방법이나 비정상적인 패킷을 자동으로 생성하여 테스트하는 방법이다. 즉 도구를 이용해 패킷을 생성한 다음 패킷 필터를 통과시키고 통과된 결과를 분석하여 보안 정책이나 필터링 규칙이 정확하게 동작하는지 판단한다[9]. 자동화된 패킷 필터링 테스트 도구는 대부분 패킷을 생성하는 패킷 생성기와 패킷을 받는 패킷 탐지기(sniffer), 보내진 패킷과 받아진 패킷의 정보를 분석하는 분석 도구로 구성된다. 기존에 사용되던 테스트 도구의 구조는 그림 1과 같다. Ftester[10]는 방화벽의 보안정책을 테스트하기 위해 사용되는 대표적인 도구이다.

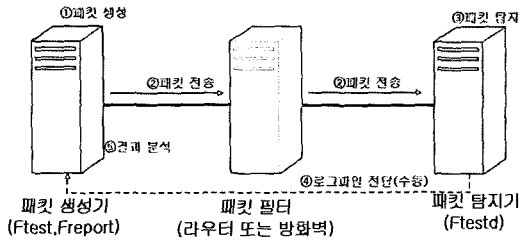


그림 1 기존의 테스트베드

Ftester는 패킷 생성기 역할을 하는 Ftest와 패킷 탐색기 역할을 하는 Ftested, 기본적인 환경을 설정하는 'ftest.conf' 파일, Ftest와 Ftested의 로그를 비교하는데 필요한 스크립트를 갖고 있는 Freport로 구성되어 있다. 패킷 생성기는 테스트 목적에 맞게 패킷을 생성한 후 타겟에 보내는 역할을 하는데 이때 보내는 패킷에 대해 확인할 수 있는 일련의 ID를 부여하고, 보낸 패킷에 대해 로그를 남긴다. 패킷 탐지기는 타겟에 도착한 모든 패킷 중 패킷 생성기에서 생성한 ID를 가진 패킷만을 읽어 들여 로그를 남긴다. 마지막으로 분석 도구는 생성기의 로그와 탐지기의 로그를 비교하여 필터링 규칙에 맞게 패킷이 전달되었는지 확인한다. 그러나 Ftester는 새로운 테스트 케이스에 대해 매번 인위적인 설정과 스크립트를 작성해 주어야 하며 분석 도구 또한 종단간의 로그에 대한 단순 비교만을 제공한다. 특히, '상태 있는 검사'를 테스트하기 위해서는 패킷을 주고받는 종단 간의 동기화가 필수적인데 Ftester의 경우에는 이러한 동기화를 정확하게 보장하지 못한다.

그러나 본 논문에서 제안하는 테스트 도구는 테스트 패킷의 체계적인 생성 방법, 테스트 오라클의 자동 생성을 통한 테스트 평가 방법을 통해 번거로운 수작업 없이 테스트 케이스 설정에서부터 결과 분석까지 모든 부분을 자동화 할 수 있고, 또한 상태 있는 검사와 같은 동적인 패킷 필터링 테스팅을 위해 종단간의 동기화를 지원하는 메커니즘을 제공함으로써 정적 패킷 필터링뿐만 아니라 동적 패킷 필터링까지 패킷 필터링의 모든 영역을 테스트할 수 있다는 강점이 있다.

## 3. 시스템 개요

### 3.1 테스트베드 구축

기존의 테스트 도구들은 패킷 필터(라우터 또는 방화벽)의 기능을 테스트하기 위해 그림 1과 같이 패킷 생성기와 패킷 탐지기를 가지고 테스트를 수행하였다. 기존 방법의 단점은 테스트 수행 결과를 평가하기 위하여 테스트 로그를 사용자가 직접 확인, 분석, 판단해야 한다는 것과 또한, 상태 기반 테스트를 수행할 때 종단간의 동기화가 어렵다는 것이다. 상태 있는 검사의 기능을 테스트하기 위해서는 테스트베드의 모든 구성 요소들이 패킷 필터의 상태에 관해 알아야 한다. 즉, 가상의 3-way handshaking을 위해 패킷 필터의 상태에 따라 패킷 전송이 양쪽 종단에서 번갈아 이루어져야 하는데 이렇게 하기 위해서는 종단간의 동기화가 필요하다. 그렇지만 기존 도구들은 상태 정보를 실시간으로 전달하여 종단간의 동기화를 이루게 하는 방법이 없었다. 이런 문제점을 해결하기 위해서 본 논문에서는 그림 2와 같은 테스트베드를 제안한다.

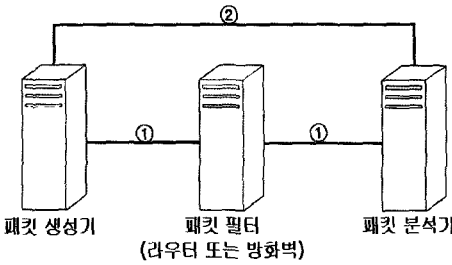


그림 2 테스트베드

그림 2의 테스트베드에서는 ②번 선을 새롭게 도입하였고, 그림 1의 패킷 탐지기 대신 패킷 분석기로 구성을 변화시켰다. 그림 2에서 패킷 생성기와 패킷 분석기는 ②번 선을 통해 종단간의 동기화를 유지할 수 있을 뿐만 아니라 테스트에 필요한 정보를 주고받는다.

### 3.2 테스트베드 내부 구조

그림 3은 테스트베드의 내부 구조이다. 여기서 패킷 생성기와 패킷 분석기의 내부 구성 요소들의 역할은 다음과 같다.

• 규칙 해석기

일련의 스크립트 형태로 작성된 패킷 필터의 필터링 규칙들을 파싱하여 규칙의 의미를 해석한다. 규칙은 ACL이나 IPtables에서 정의하여 사용하는 형태 모두 가능하다. 규칙 해석기는 테스트 대상에 따라 필요한 정보를 파악하고 이를 테스트 케이스 생성기에 전달한다. 여기서 테스트 대상이란 패킷 필터의 보안 정책을 의미한다. 패킷 필터의 보안 정책은 크게 기본적인 정책(송신자 주소, 목적지 주소, 송신자 포트, 목적지 포트, 프로토콜), Flooding 제어 정책, 상태 기반 보안 정책으로 나누어진다.

• 테스트 케이스 생성기

본 논문에서 제안하는 테스트 케이스 생성 방법에 따라 테스트 케이스들을 생성한다.

• 패킷 전송기(packet sender)

테스트 케이스에 따라 해당 패킷들을 생성하여 라우터로 전송하고, 전송한 패킷들의 로그를 기록한다.

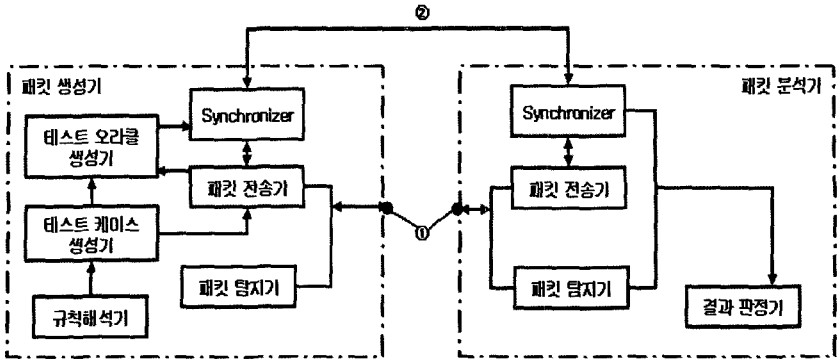


그림 3 테스트베드 내부 구조

• 패킷 탐지기(packet sniffer)

라우터를 거쳐 도착한 패킷을 탐지하여 로그를 남긴다.

• synchronizer

테스트 수행 중에 패킷 생성기와 패킷 분석기 사이에서 전송할 패킷을 제외한 다른 정보들을 주고받는다. 또한 상태 기반 테스트의 경우, 패킷 생성기와 패킷 분석기의 상태 전이를 위해 동기화를 유지시켜 준다.

• 테스트 오라클 생성기

생성된 테스트 케이스와 패킷 전송기에서 전송한 패킷 로그를 입력으로 받아들여 테스트 오라클을 생성한다. 테스트 오라클은 테스트 수행 시 예상되는 올바른 결과를 의미한다. 이러한 테스트 오라클을 사용하여 테스트 실행 결과의 옳고 그름을 판단할 수 있다. 즉 테스트 실행 결과와 테스트 오라클을 비교하여 차이가 있으면 테스트 대상에 오류가 존재한다고 생각한다.

• 결과 판정기

패킷 탐지기가 수신한 패킷 로그와 패킷 생성기에서 넘어온 테스트 오라클을 비교하여 테스트 결과를 판정한다. 경우에 따라서 결과 분석기는 수신한 패킷 로그를 분석한다. 즉 각각의 테스트 케이스에 따른 로그에 대해 패킷의 개수를 파악하며, Flooding 정책에 대한 테스트의 경우 시간간격을 기반으로 패킷 로그 정보를 분석한다.

테스트베드는 다음과 같이 동작한다. 먼저, 패킷 생성기에서는 패킷 필터에서 세운 보안 규칙을 해석(규칙해석기)하여 테스트 케이스를 생성(테스트 케이스 생성기)하고 이 테스트 케이스에 따라 패킷을 생성한 다음 타겟으로 전송(패킷 전송기)한다. 또한 계속해서 전송한 패킷을 로그로 남긴다. 테스트 오라클 생성기는 생성된 테스트 케이스와 전송된 패킷의 로그를 입력으로 하여 테스트 오라클을 생성한 다음 그것을 synchronizer와 ②번 선을 통해 패킷 분석기에 보낸다. 다음으로, 패킷 분석기는 패킷 생성기에서 전송한 패킷들 중 수신 패킷(패킷 탐지기)에 대한 로그를 남긴다. 다음으로 ②번 선을 통해 전달된 테스트 오라클을 결과 분석기에 보내어 테스트 결과를 판정한다.

상태 기반 테스트의 경우 생성된 테스트 케이스에 따라 패킷 생성기와 패킷 분석기의 synchronizer를 통해 가상으로 TCP의 3-Way Handshaking을 하면서 테스트를 수행한다. 이를 위해 테스트베드는 양쪽의 패킷 전송기들과 패킷 탐지기들을 적절하게 제어한다.

## 4. 패킷 필터링 테스트

기존의 패킷 필터링 테스트는 대부분 테스터에 의한 수동 테스트 방법을 이용한다. 이는 테스트의 모든 과정에서 테스터의 개입이 요구되며, 이는 테스트 대상 시스템, 네트워크, 테스트 방법에 대한 지식 없이는 불가능하다. 또한 수동 테스트 방법에서 결과의 판단은 테스터에 의해 수행되기 때문에 테스터의 역량에 따라 그 신뢰성의 기복이 심하다. 이러한 문제는 대표적인 패킷 필터링 테스트 도구인 Ftester에서도 나타난다. Ftester는 테스트 수행 과정과 일부 결과 판단 과정을 자동화 하지만 테스트 케이스의 선정과 결과 판단에서 테스터의 개입으로 인한 신뢰성 문제를 갖고 있다. 본 논문에서는 이러한 문제를 해결하기 위해 보안 규칙에 따라 테스트 대상을 나누고, 각 대상별 자동화된 테스트 케이스 생성 방법, 테스트 오라클 생성 방법을 제안한다.

### 4.1 테스트 대상 및 방법

일반적으로 패킷 필터링 기능을 제공하는 라우터나 방화벽에서의 보안은 송신자 IP 주소, 목적지 IP 주소, 송신자 포트, 목적지 포트 및 프로토콜의 5가지 튜플을 조합하여 여러 개의 보안 정책을 세울 수 있다. 그 보안 정책은 보안의 목적과 환경에 따라 달라질 수 있으며 또한 이 5가지 튜플들과 함께 추가적인 제약사항을 기술함으로써 특별한 보안 정책을 세울 수 있는데, 이러한 특별한 보안 정책으로는 Flooding 제어 보안 정책과 상태 기반 보안 정책이 있다. 패킷 필터는 이러한 보안 정

책을 가지고 필터링 작업을 수행하므로 이러한 보안 정책이 테스트 대상이 된다. 다음은 테스트 대상을 분류하고 그 각각에 대한 테스트 방법을 살펴본다. 여기서 필터링 규칙은 IPtables의 형태로 기술된다.

4.1.1 기본 정책(주소, 포트, 프로토콜 기반 정책)

기본 정책은 네트워크상에서 가장 간단한 보안 정책으로써 송신자 IP 주소, 목적지 IP 주소, 송신자 포트, 목적지 포트 및 프로토콜의 5가지 튜플을 조합하여 보안 정책을 세운다. 예를 들어, 특정 송신자 IP 주소나 포트 번호를 갖는 패킷을 허락(accept)하거나 거를(drop) 수 있다.

표 1 기본 보안 정책

| 규칙 번호 | IPtables 규칙 |
|---|---|
| 1 | -s 168.188.46.0/24 -j ACCEPT |
| 2 | --dport 21 -j ACCEPT |
| 3 | -p tcp -j ACCEPT |
| 4 | -p tcp -s 168.188.46.0/24 --dport 21 -j DROP |

표 1의 규칙 1은 송신자 IP 주소에 대한 보안 정책의 사용 예로 168.188.46.0의 IP에 대해 Prefix(넷 마스크에서 1로 설정될 비트수)가 24로 168.188.46.0~168.188.46.255의 범위에 대해 패킷을 허락한다(accept)는 의미이다. 규칙 2는 목적지 포트 21번에 대해 패킷을 허락한다는 의미이며 규칙 3은 TCP프로토콜을 갖는 패킷에 대해 허락한다는 의미이다. 규칙 4는 5가지 튜플 중 규칙 1, 2, 3에 사용된 3가지 튜플(송신자 주소, 목적지 포트, 프로토콜)을 조합하여 하나의 규칙으로 정의한 거절(drop) 정책의 사용 예이다.

이런 보안 정책을 테스트하기 위해서는 설정된 규칙에 맞는 패킷을 전송하거나 규칙에 맞지 않는 패킷을 전송한 다음 수신된 패킷을 분석하여 규칙에 맞게 패킷이 허락되거나 거절되었는지를 파악한다. 규칙 1을 예로 들면, 송신자 주소 168.188.46.0부터 168.188.46.255까지 패킷을 전송하고 수신 측에서는 이들 패킷 모두가 수신되었다면 규칙 1의 보안 정책을 만족한다고 판단한다. 또한 범위 밖의 패킷들을 전송하였을 때 수신 측에 도착한 패킷이 없다면 위의 정책을 만족한다고 판단한다.

4.1.2 Flooding 제어 정책

서비스 공격의 유형은 SYN flooding, ICMP flooding, UDP flooding이 대부분이다[11]. 그림 4는 flooding 공격 중 하나인 TCP flooding을 보여준다. TCP flooding은 TCP의 연결 과정인 3-way handshaking을 이용하여 서비스를 제공하지 못하게 하는 방법이다. 공격자(attacker)가 대상시스템(victim)에 송신자 IP 주소를 spoofing하여 SYN 패킷을 특정 포트로 전송하게 되면

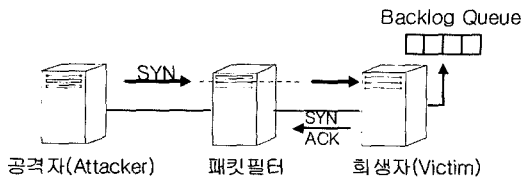


그림 4 TCP Flooding

이 포트의 대기 큐(backlog queue)가 가득 차게 되어 이 포트로 들어오는 연결 요청은 큐가 빌 때까지 무시됨으로써 결국 서비스를 받지 못한다. 이러한 flooding은 패킷 필터에 의해 제어될 수 있다. 패킷 필터는 통과하는 패킷이 악의 없는 정상 패킷이 아니라 공격 패킷이라는 것을 감지해야 한다. 하지만 이런 공격을 감지하기란 쉽지 않다. 따라서 패킷 필터는 flooding으로 여겨지는 패킷들을 최대 허락 패킷 수(burst)와 단위 시간당 허용 가능한 패킷 수(rate)를 가지고 제한함으로써 자원의 과도한 할당을 피하여 DoS(Denial of Service)나 DDoS(Distributed DoS)와 같은 flooding 공격을 막을 수 있다[11]. 표 2의 규칙은 rate가 '5/초'이고 burst가 10인 flooding 제어 보안 정책이다. 이 규칙은 flooding 공격과 같이 많은 개수의 패킷이 도착하였을 때 10개의 SYN 패킷을 수락하도록 경계값(threshold)을 지정함으로써 도착한 패킷을 10개까지는 허락하고 그 이후(지정된 경계값을 넘게 된 시점)부터는 초당 5개의 SYN 패킷만을 허용하고, 그 이상의 패킷들은 거부(drop)하라는 의미이다.

표 2 Flooding 제어 보안 정책

| 규칙 번호 | IPtables 규칙 |
|---|---|
| 1 | -m limit --limit 5/second --limit-burst 10 -j DROP |

이런 보안 정책이 제대로 수행되는지 테스트하기 위해서는 flooding으로 여겨지도록 많은 개수의 패킷들을 전송한 다음, 수신 측이 제한된 개수의 패킷만을 처리하는지 확인하면 된다. 예를 들어, 표 2의 보안 정책에 대해 테스트할 경우 많은 개수의 패킷을 전송한 다음 설정된 burst와 rate(5/초)에 따라 처음에 최대 10개, 그 이후로는 평균적으로 초당 5개의 패킷이 수신 측에 전달되는지를 확인하면 된다.

4.1.3 상태 기반 정책

상태 있는 검사(stateful inspection)는 패킷의 헤더 정보를 기반으로 하는 정적인 패킷 필터링과는 달리 특정 기간에 걸쳐 패킷들의 통신을 추적하는 강화된 보안

정책이다. 즉, 정적인 패킷 필터링처럼 정의된 규칙에만 의존하는 것이 아니라 연결(connection) 추적 메커니즘에 의해 방화벽이나 라우터를 통과했던 이전의 패킷들에 의해 설정된 환경을 바탕으로 패킷 필터링 결정이 이루어진다. 이런 예로써 TCP 프로토콜을 이용한 연결에서는 3-way handshaking에 대한 패킷 전송을 추적하여 유효한 패킷과 그렇지 않은 패킷들을 결정하게 된다.

그림 5를 보면 Client에서 Server로 SYN 패킷이 전달되면서 연결의 상태는 초기(Initial) 상태에서 NEW로 전이되고 Server에서 Client로 SYN+ACK패킷이 전송되면서 상태는 NEW에서 ESTABLISHED상태로 전이된다. 다시 ESTABLISHED 상태는 FIN 패킷 또는 Time-out에 의해 연결이 종료되면서 초기 상태로 전이된다.

표 3 상태 보안 정책

| 규칙 번호 | IPtables 규칙 |
|---|---|
| 1 | -p TCP -d 168.188.46.1 --dport 21 -m state --state NEW,ESTABLISHED -j ACCEPT |
| 2 | -p TCP -s 168.188.46.1 --sport 21 -m state --state ESTABLISHED -j ACCEPT |

그림 5에서 Server의 IP주소가 168.188.46.1이라고 할 때, 표 3에서 규칙 1은 그림 5의 Client에서 Server로의 패킷 전송을 제어하기 위한 규칙이고, 규칙 2는 Server에서 Client로의 패킷 전송을 제어하기 위한 규칙이다. 즉, 규칙1은 NEW와 ESTABLISHED 상태를 허용할 수 있으므로 SYN 패킷에 의해 NEW 상태로 전이할 수 있고 SYN+ACK 패킷에 의해 ESTABLISHED 상태로 전이할 수 있다. 이는 Client에서 Server로의 연결(connection) 시도가 가능함을 의미한다. 규칙 2는 ESTABLISHED 상태만을 허용하므로 Client와의 연결이 이루어졌을 때만 패킷 전송이 가능함을 의미한다.

상태 기반 정책에 대한 테스트는 유효한 패킷을 보내 정상적인 상태 전이와 정상적인 통신이 이루어지는지 확인하고, 유효하지 않은 패킷에 대해서는 상태 전이가 일어나지 않고 또한, 통신이 이루어지지 않는지 확인한다. 유효하지 않은 패킷에 대해서 상태 전이가 일어나고 그로 인해 통신이 이루어진다면 이 패킷 필터는 상태 있는 검사의 기능을 제대로 수행하지 못한다는 것을 의미한다.

### 4.2 테스트 케이스 설계

(1) 기본 정책에 대한 테스트 케이스

보안 정책이 특정 IP 혹은 특정 범위 내의 IP 주소를 갖는 패킷을 허락(accept)하는 정책이라면 입력 도메인은 그림 6과 같이 세 개의 동치 클래스(equivalent class)로 나눠진다. 이 때 생성되는 테스트 케이스는 각각의 동치 클래스로부터 추출된 IP 주소를 갖는 패킷들로 구성된다. 이는 거부(drop) 정책에 대해서도 동일하게 적용된다.

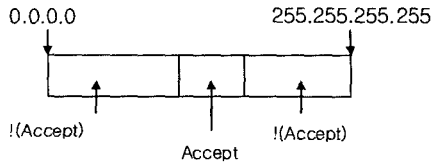


그림 6 IP 주소 입력 도메인

IP 주소 기반 테스트에서 보안 규칙은 IP 주소와 Prefix에 의해 결정된다. 표 1의 규칙 1의 경우, 허락될 IP 주소의 범위는 Prefix가 24bit이므로 168.188.46.0~168.188.46.255가 된다. 물론 이 때 허락되지 않아야 할 IP 주소의 범위는 0.0.0.0~168.188.45.255와 168.188.47.0~255.255.255.255가 된다.

IP 주소와 Prefix에 의해 허락될 패킷의 범위가 설정되면 테스트 케이스 생성기가 IP 주소 기반 테스트 케이스들을 생성한다. 이 때 생성되는 테스트 케이스는 허락 범위 내의 클래스와 범위 아래의 클래스 및 범위 위의 클래스에서 고루 선택된다. 이렇게 테스트 케이스를 추출하는 이유는 허락되지 않는 범위에서 패킷을 선택했을 때 그 패킷이 패킷 필터에 의해 제대로 거부(drop)되는지와 허락되는 범위의 패킷은 제대로 허락(accept)

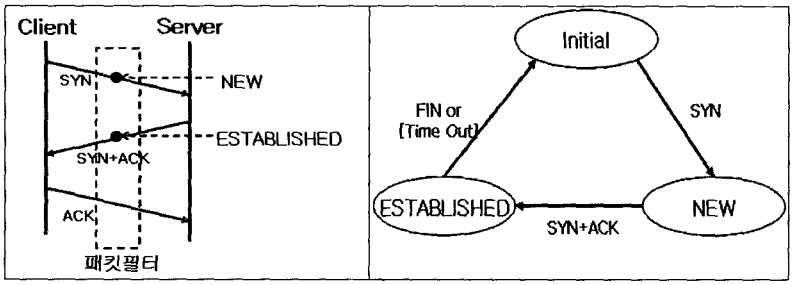


그림 5 상태전이

되는지를 파악하기 위함이다. 또한 허락 범위의 경계값 부근에서 충분한 테스트 케이스를 추출한다. 즉 경계값과 경계값보다 하나 작은 값, 경계값보다 하나 큰 값의 형태로 테스트 케이스를 추출한다. 이렇게 테스트 케이스를 추출하는 이유는 많은 오류들이 경계값 부근에서 발생하기 때문이다[12]. 예를 들어, 표 1의 규칙 1의 경우 다음 표 4와 같은 테스트 케이스를 생성할 수 있다.

표 4 IP 주소기반 보안 정책에 대한 테스트 케이스 사례

| TC# | 선택 조건 | 선택 영역 | IP주소기반 패킷 |
|---|---|---|---|
| 1 | 동치클래스 | OUT | 168.188.44.74 |
| 2 | | IN | 168.188.46.100 |
| 3 | | OUT | 168.188.55.32 |
| 4 | 경계값 분석 | OUT | 168.188.45.255 |
| 5 | | IN | 168.188.46.0 |
| 6 | | IN | 168.188.46.255 |
| 7 | | OUT | 168.188.47.0 |

표에서 보는 바와 같이 이 예제의 경우 경계값 부근에서 추출한 테스트 케이스가 크게 의미를 갖지 않을 수 있지만 만일 허락될 IP 주소의 범위가 168.188.46.0~168.188.46.127과 같이 주어졌을 때는 경계값으로 168.188.46.127과 168.188.46.128을 선택하여 테스트 해 보는 것이 중요한 의미를 갖는다.

포트에 대한 테스트 케이스도 IP 주소 테스트 케이스처럼 유효한 포트 번호 1~65,535 사이의 값에 대해 허락되는 포트 번호에 대한 동치 클래스와 허락되지 않는 포트 번호에 대한 동치 클래스로 나누고 각각에서 테스트 케이스를 추출한다. 앞서 설명했듯이 대부분의 네트워크 공격은 TCP, UDP, ICMP를 이용하여 이루어진다. 따라서 프로토콜에 대한 테스트 케이스를 위해 이런 프로토콜에 대한 선택을 제공한다.

기본 정책에 대한 테스트를 수행하기 위해서는 5가지 튜플에 대해서 각각 규칙을 설정하고 테스트를 할 수 있다. 5가지 튜플에 대해 테스트 케이스를 생성하는 방법은 두 가지가 있다. 하나는 각 튜플에 대해 개별적인 테스트 케이스를 생성하기 위해 다른 튜플의 값을 고정하는 방법이고, 다른 하나는 최소한의 테스트 케이스를 생성하기 위해 모든 튜플을 포함할 수 있는 테스트 케이스를 생성하는 방법이다[12]. 우리는 첫 번째 방법을 선택했다. 왜냐하면 패킷 필터링 테스트는 네트워크상에서 이루어지므로 여러 가지 가변적인 요소가 있다. 따라서 최소한의 테스트 수행 결과보다는 각 튜플에 대해 여러 가지 테스트 케이스를 가지고 다양한 테스트를 수행함으로써 보다 정확한 결과를 얻을 수 있을 것이라 생각했기 때문이다.

(2) Flooding 제어 정책에 대한 테스트 케이스

Flooding 제어 정책에 대해서는 rate와 burst에 대한 규칙을 기반으로 flooding으로 인식될 수 있는 상황과 그렇지 않은 상황에 대한 테스트 케이스를 생성한다. Burst는 허락되는 최대 패킷 수를 나타내고 burst를 채운 다음부터 rate는 단위 시간당 허용되는 패킷 수를 나타내므로 flooding 상황은 단위 시간 내에 지정된 burst와 rate 중 큰 값보다 많은 패킷을 생성하면 된다. Flooding이 아닌 상황은 단위 시간 내에 지정된 burst 또는 rate중 작은 값과 같거나 적은 패킷을 생성하면 된다. 이를 간결하게 표현하면 단위 시간 당 생성되는 패킷 수를 m이라 할 때 다음과 같이 나타낼 수 있다.

| 조건 | 상황 |
|---|---|
| m > MAX(rate, burst) | flooding |
| m ≤ MIN(rate, burst) | non-flooding |

예를 들어, 표 2의 보안 정책에 대해서 다음 표 5와 같은 테스트 케이스를 생성할 수 있다.

표 5 Flooding 보안 정책에 대한 테스트 케이스 사례

| TC# | interval | 생성패킷수/단위시간 | flooding 상태 |
|---|---|---|---|
| 1 | 10초 | 100/초 | flooding |
| 2 | 10초 | 5/초 | non-flooding |

(3) 상태 기반 정책에 대한 테스트 케이스

상태 기반 정책에 대한 테스트 케이스는 정상적인 상태전이를 유발하는 테스트 케이스와 비정상적인 상태전이를 유도하는 테스트 케이스로 이루어진다. 예를 들어, 표 3의 규칙을 테스트하기 위해서는 그림 7의 왼쪽과 오른쪽 상황을 테스트 하여야 한다. 정상적인 상황에 대해서는 왼쪽 그림과 같이 가상의 3-way handshaking으로 패킷을 전송하고 ESTABLISHED 상태일 때

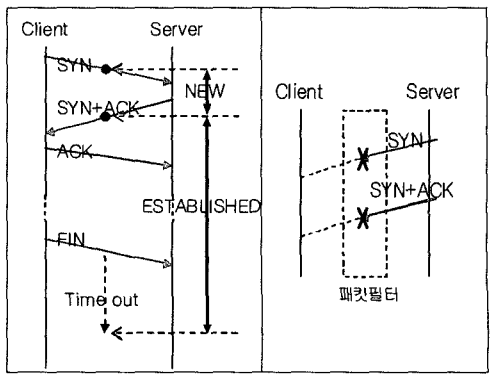


그림 7 상태 기반 정책에 대한 테스트

ACK 패킷들을 전송하여 전송이 올바르게 이루어지는지 확인한다. 표 3의 규칙 1의 경우 client에서 server로의 패킷 전송에 대해서 패킷 필터는 NEW와 ESTABLISHED 상태로 전이가 가능하기 때문에 client로부터 SYN 패킷이 들어오면 패킷 필터는 NEW 상태로 전이를 수행한다. 계속해서 server로부터 SYN+ACK 패킷이 들어오면 ESTABLISHED 상태로 전이가 일어나고 그 상태에서 다양한 데이타 패킷들을 통과시킨다. 한편 비정상적인 상황에 대해서는 그림 7의 오른쪽처럼 상태 전이가 일어나지 않는 패킷을 전송해봄으로써 패킷 필터가 패킷을 올바르게 거부(drop)하는지 확인한다. 표 3의 규칙 2의 경우 server로부터의 패킷 전송에 대해서 패킷 필터는 ESTABLISHED 상태로만 전이가 가능하기 때문에 패킷 필터가 초기 상태일 때는 server로부터 SYN 패킷이 들어오더라도 패킷 필터는 NEW 상태로의 전이가 가능하지 않다. 따라서 상태 전이를 수행할 수 없기 때문에 계속해서 들어오는 패킷들을 통과시키지 않는다. 그러나 그림 7의 왼쪽처럼 이미 client와의 연결이 이루어졌을 때에는, 즉 ESTABLISHED 상태일 때는 server로부터 어떤 패킷이 들어오더라도 패킷 필터는 패킷들을 통과시킨다. 생성된 테스트 케이스는 다음과 같다. 여기서 S+A로 표현한 패킷은 SYN+ACK 패킷을 의미한다. 그리고 전이 유효성은 패킷이 정상적인(valid) 상태전이를 유발하는지 아니면 비정상적인(invalid) 상태전이를 유발하는지를 나타낸다.

표 6 상태 기반 정책에 대한 테스트 케이스 사례

| TC# | 패킷전송자 | 패킷필터 상태 | 패킷 | 패킷 유효성 |
|---|---|---|---|---|
| 1 | Client | ESTABLISHED 아님 | SYN | V |
| 2 | | | ACK | I |
| 3 | | ESTABLISHED | SYN | V |
| 4 | | | ACK | V |
| 5 | Server | Initial | SYN | I |
| 6 | | | ACK | I |
| 7 | | NEW | S+A | V |
| 8 | | | ACK | I |
| 9 | | ESTABLISHED | SYN | V |
| 10 | | | ACK | V |

### 4.3 테스트 오라클 설계

본 논문에서 제안하는 시스템은 테스트 케이스에 따라 테스트 오라클을 자동으로 생성한다. 테스트 오라클은 테스트 수행 시 예상되는 올바른 결과를 의미한다[12]. 이러한 테스트 오라클을 사용하여 테스트 실행 결과의 옳고 그름을 판단할 수 있다. 테스트 오라클은 테스트 케이스 생성기에서 생성된 테스트 케이스와 패킷 전송기에서 전송한 패킷 로그를 입력으로 받아들여 테스트 오라클 생성기에 의해 자동 생성된다.

(1) 기본 정책에 대한 테스트 오라클

기본 정책에 대한 테스트 오라클은 다음과 같다. 여기서 선택 영역은 테스트 케이스를 선택하게 되는 입력 도메인 영역으로서 IN 영역은 규칙에서 지정된 범위, OUT은 지정된 범위 밖의 영역을 의미한다. 전송 패킷 로그($Log_S$)는 패킷 생성기 내의 패킷 전송기에서 전송한 패킷들의 로그를 의미하고, 수신 패킷 로그($Log_R$)는 패킷 분석기 내의 패킷 탐지기에서 수신한 패킷들의 로그를 의미한다.

표 7 기본 정책에 대한 테스트 오라클

| 선택 영역 | 규칙 | 전송패킷 로그 | 수신패킷 로그 | 판단 | 테스트 결과 |
|---|---|---|---|---|---|
| IN | Accept | $Log_S$ | $Log_R$ | $Log_S = Log_R$ | 성공 |
| | | | | $Log_S \neq Log_R$ | 실패 |
| OUT | Accept | $Log_S$ | $Log_R$ | $Log_R = \emptyset$ | 성공 |
| | | | | $Log_R \neq \emptyset$ | 실패 |
| IN | Drop | $Log_S$ | $Log_R$ | $Log_R = \emptyset$ | 성공 |
| | | | | $Log_R \neq \emptyset$ | 실패 |
| OUT | Drop | $Log_S$ | $Log_R$ | $Log_S = Log_R$ | 성공 |
| | | | | $Log_S \neq Log_R$ | 실패 |

(2) Flooding 제어 정책에 대한 테스트 오라클

Flooding 제어 정책에 대한 테스트 오라클은 다음과 같다. 여기서 Interval은 한 번의 테스트를 수행한 시간으로 n(n>1) 단위시간으로 나타낸다. 따라서 전송패킷 로그/Interval은 n 단위 시간 내에 전송한 패킷들의 로그를 의미한다. 수신 로그의 경우도 마찬가지이다. 그리고 |Log*|는 기록된 패킷의 개수를 의미한다.

표 8 Flooding 제어 정책에 대한 테스트 오라클

| 전송패킷로그 /Interval | 수신패킷로그 /Interval | 상황 | 판단 | 결과 |
|---|---|---|---|---|
| $Log_S$ | $Log_R$ | flooding | $\lvert Log_R \rvert \leq burst+(n-1)*rate$ | 성공 |
| | | | $\lvert Log_R \rvert > burst+(n-1)*rate$ | 실패 |
| | | non-flooding | $Log_S = Log_R$ | 성공 |
| | | | $Log_S \neq Log_R$ | 실패 |

(3) 상태 기반 정책에 대한 테스트 오라클

상태 기반 정책에 대한 테스트 오라클은 다음과 같다. 여기서 $Log_C$는 client 역할을 수행하는 패킷 생성기 내의 패킷 전송기가 전송한 패킷과 패킷 탐지기가 수신한 패킷들에 대한 로그이며, $Log_S$는 server 역할을 수행하는 패킷 분석기 내의 패킷 전송기가 전송한 패킷과 패킷 탐지기가 수신한 패킷들에 대한 로그이다.

표 9 상태 기반 정책에 대한 오라클

| 전송자 | 필터상태 | 패킷 | 판단 | 결과 |
|---|---|---|---|---|
| Client | Initial | SYN | $Log_C = Log_S$ | 성공 |
| | | | $Log_C \neq Log_S$ | 실패 |
| | | ACK | $Log_S = \emptyset$ | 성공 |
| | | | $Log_S \neq \emptyset$ | 실패 |
| | NEW | SYN | $Log_C = Log_S$ | 성공 |
| | | | $Log_C \neq Log_S$ | 실패 |
| | | ACK | $Log_C - \{ACK\} = Log_S$ | 성공 |
| | | | $Log_C - \{ACK\} \neq Log_S$ | 실패 |
| | ESTAB LISHED | SYN | $Log_C = Log_S$ | 성공 |
| | | | $Log_C \neq Log_S$ | 실패 |
| | | ACK | $Log_C = Log_S$ | 성공 |
| | | | $Log_C \neq Log_S$ | 실패 |
| Server | Initial | SYN | $Log_C = \emptyset$ | 성공 |
| | | | $Log_C \neq \emptyset$ | 실패 |
| | | ACK | $Log_C = \emptyset$ | 성공 |
| | | | $Log_C \neq \emptyset$ | 실패 |
| | NEW | S+A | $Log_C = Log_S$ | 성공 |
| | | | $Log_C \neq Log_S$ | 실패 |
| | | ACK | $Log_C = Log_S - \{ACK\}$ | 성공 |
| | | | $Log_C \neq Log_S - \{ACK\}$ | 실패 |
| | ESTAB LISHED | SYN | $Log_C = Log_S$ | 성공 |
| | | | $Log_C \neq Log_S$ | 실패 |
| | | ACK | $Log_C = Log_S$ | 성공 |
| | | | $Log_C \neq Log_S$ | 실패 |

## 5. 테스트 수행 및 결과

본 논문에서 제안하는 방법을 사용하여 IPtables 버전 1.2.7로 기술된 보안 정책을 갖는 라우터를 테스트하여 보았다. 이 장에서는 테스트 대상 필터링 규칙이 주어졌을 때, 4장에서 제시한 방법에 따라 자동으로 테스트 케이스가 생성되고, 생성된 테스트 케이스를 기반으로 패킷이 생성되어 전송되고, 그 결과를 바탕으로 테스트 오라클이 생성되어 각각의 보안 정책이 정확하게 동작하는지 자동으로 평가하는 과정을 보여준다.

### 5.1 기본 정책에 대한 테스트

5 튜플로 표현되는 기본 정책에 대하여 테스트를 수행하였다. 여기서 프로토콜은 TCP로 고정하고, 송신자 IP 주소의 변화에 대해서 테스트를 수행하였고 다른 한편으로 목적지 포트 번호를 변경시켜 가면서 테스트를 수행하였다. IP 주소 기반 테스트는 라우터의 보안 정책으로 "-p tcp -s 168.188.46.0/24 -j ACCEPT"라는 규칙을 세우고 표 10의 테스트 케이스 그룹 1번부터 4번까지의 조건으로 테스트를 실시하였다.

4.2 절의 표 4에서 보는 바와 같이 최소한의 테스트 케이스는 입력 도메인의 IN/OUT 영역에서 각각 하나씩을 선택하면 된다. 그런데 우리는 생성할 패킷을 1개, n개 또는 무한대 개수(사실상 허락되는 최대 개수)를 선택할 수 있도록 하는 기능을 제공하기로 하였다. 이렇게 결정한 이유는 보안 시스템은 네트워크상에서 동작하므로 여러 가지 가변적인 요소가 있을 수 있고, 따라서 한 개의 테스트 케이스를 수행시킨 결과만으로 그 테스트 케이스에 대한 정확한 테스트 결과를 얻었다고 단정 지을 수 없기 때문이다. 이 테스트에서는 패킷을 10개, 100개 등으로 다양하게 생성하였다. 패킷을 n개 생성할 경우 입력 도메인에서 고른 분포를 갖기 위해 지정된 IP 영역을 n등분하여 각 등분에서 대표 값을 뽑아 패킷을 생성한다. 위의 표 11은 테스트케이스 그룹1에 대해 실제 생성된 패킷을 보여준다. 테스트케이스 그룹1은 "-p tcp -s 168.188.46.0/24 -j ACCEPT"의 규칙에서 IN영역에서 10개의 패킷을 생성하기 때문에 실제 전송될 패킷은 경계값 분석과 동치 클래스 분할을 통해 송신자 IP를 변경하며 패킷을 전송한다. 테스트케이스 그룹 2,3,4에 대해서도 같은 방법으로 IN/OUT 영역에 대해 각각의 테스트 데이타를 생성해 테스트를 수행한다.

포트 기반 테스트는 표 10의 테스트 케이스 그룹 5번부터 8번까지의 조건으로 테스트를 실시하였다. 이때, 라우터의 보안 정책으로 "-p tcp --dport 1:1024 -j ACCEPT"라는 규칙을 사용하였다. 이 규칙은 포트 1번부터 1024번까지 만을 Accept하라는 의미이다.

표 10 기본 정책 테스트 케이스 그룹

| 테스트케이스 그룹 | 패킷 수 | IN/OUT 영역 |
|---|---|---|
| 1 | 10 | IN |
| 2 | 100 | IN |
| 3 | 10 | OUT |
| 4 | 100 | OUT |
| 5 | 10 | IN |
| 6 | 100 | IN |
| 7 | 10 | OUT |
| 8 | 100 | OUT |

표 11 그룹1에 대한 전송 패킷

| 전송 패킷 | 송신자 IP | 목적지 IP | 프로토콜 | S Port | D Port |
|---|---|---|---|---|---|
| 1 | 168.188.46.0 | 168.188.45.1 | TCP | 20 | 21 |
| 2 | 168.188.46.25 | 168.188.45.1 | TCP | 20 | 21 |
| 3 | 168.188.46.50 | 168.188.45.1 | TCP | 20 | 21 |
| 4 | 168.188.46.75 | 168.188.45.1 | TCP | 20 | 21 |
| 5 | 168.188.46.100 | 168.188.45.1 | TCP | 20 | 21 |
| 6 | 168.188.46.125 | 168.188.45.1 | TCP | 20 | 21 |
| 7 | 168.188.46.150 | 168.188.45.1 | TCP | 20 | 21 |
| 8 | 168.188.46.175 | 168.188.45.1 | TCP | 20 | 21 |
| 9 | 168.188.46.200 | 168.188.45.1 | TCP | 20 | 21 |
| 10 | 168.188.46.255 | 168.188.45.1 | TCP | 20 | 21 |

아래 그림 8은 표 10의 조건대로 테스트 케이스 그룹 1번부터 4번까지는 IP 주소 기반 테스트를, 5번부터 8번까지는 포트 기반 테스트를 수행한 후의 결과 화면이다. 테스트케이스 그룹1에 대한 테스트 케이스가 IN영역이므로 테스트 오라클은 전송된 패킷 전체의 집합이다. 따라서 테스트 수행 결과, 전송된 패킷 10개가 모두 수신되어 결과는 SUCCESS이다. 나머지 테스트 케이스에 대해서도 동일한 방법으로 테스트를 수행하였고, 그 결과가 모두 SUCCESS이다. 아래의 테스트 결과 패킷 필터링이 정확하게 동작함을 알 수 있었으며 결과적으로 송신자 IP 주소와 목적지 포트에 대하여 라우터의 보안 정책이 제대로 동작함을 알 수 있었다.

Log Analysis

| Acpt/Dny | Sent | Received | Result |
|---|---|---|---|
| Accept | 10 | 10 | SUCCE... |
| Accept | 100 | 100 | SUCCE... |
| Accept | 10 | 0 | SUCCE... |
| Accept | 100 | 0 | SUCCE... |
| Accept | 10 | 10 | SUCCE... |
| Accept | 100 | 100 | SUCCE... |
| Accept | 10 | 0 | SUCCE... |
| Accept | 100 | 0 | SUCCE... |

그림 8 기본 정책 테스트 수행 결과

### 5.2 Flooding 제어 정책에 대한 테스트

Flooding을 제어하기 위해 라우터의 보안 정책으로 rate는 5/sec, burst는 10으로 설정하고 테스트를 수행하였다. 이에 대한 규칙은 표 2와 같다. 이 때 사용한 테스트 케이스는 표 12와 같고, 생성되는 패킷은 표 13과 같다. 이때 생성되는 패킷은 168.188.46.20의 송신자에서 168.188.45.1의 목적지로 초당 3개의 패킷을 5초 동안 전송하므로 총 15개의 패킷이 전송된다.

그림 9는 테스트 케이스 1번의 수행 결과이다. 그림 9의 오른쪽의 flooding 분석표에서 보듯이 초당 3개의 패킷이 라우터를 통과하였다. 이러한 결과는 라우터 보안 정책의 rate나 burst보다도 낮은 비율로 패킷을 생성해서 보냈기 때문이다. 즉, 매초마다 flooding이 아닌 상황

Log Analysis

| Acpt/Dny | Sent | Received | Result |
|---|---|---|---|
| | 15 | 15 | SUCCE... |
| | 15 | 15 | SUCCE... |
| | 15 | 15 | SUCCE... |
| | | | |
| | | | |
| | | | |

Flood Analysis

| Interval | Received |
|---|---|
| 1 | 3 |
| 2 | 3 |
| 3 | 3 |
| 4 | 3 |
| 5 | 3 |
| | |

그림 9 Flooding 테스트 결과 1

이 되도록 패킷을 전송하였기 때문에 패킷이 모두 통과할 수 있었다.

테스트 케이스 2번은 라우터의 보안 정책으로 세운 rate나 burst보다 높은 50패킷/sec의 비율로 패킷을 생성한다. 즉 flooding 상황이 되도록 패킷을 전송한다. 초당 50개의 패킷을 생성해서 보내지만 라우터를 통과하는 패킷은 그림 10의 flooding 분석표에서 보듯이 burst에 의해 많아야 10개이고 rate에 의해 보통 5개로 제한되는 결과를 나타내었다. 테스트 결과 flooding으로 간주되는 테스트 케이스의 경우 보안 정책으로 세운 burst와 rate에 따라 패킷 필터링이 제대로 동작함을 확인할 수 있었다.

Log Analysis

| Acpt/Dny | Sent | Received | Result |
|---|---|---|---|
| | 250 | 27 | SUCCE... |
| | 250 | 27 | SUCCE... |
| | 250 | 29 | SUCCE... |
| | | | |
| | | | |
| | | | |

Flood Analysis

| Interval | Received |
|---|---|
| 1 | 10 |
| 2 | 4 |
| 3 | 5 |
| 4 | 5 |
| 5 | 5 |
| | |

그림 10 Flooding 테스트 결과 2

본 논문에서는 SYN flooding의 결과만을 보였지만 실험결과 UDP flooding, Ping flooding에 대해서도 같은 결과를 확인하였다.

### 5.3 상태 기반 정책에 대한 테스트

상태 기반 정책에 대한 테스트는 먼저 가상의 3-way handshaking을 이루도록 패킷 전송을 하여 ESTAB-

표 12 Flooding 정책에 대한 테스트 케이스

| TC# | 생성한 패킷 수/초 | interval |
|---|---|---|
| 1 | 3/초 | 5초 |
| 2 | 50/초 | 5초 |

표 13 테스트 케이스1에 대한 전송 패킷

| 전송 패킷 | 송신자 IP | 목적지 IP | FLAG |
|---|---|---|---|
| 1 | 168.188.46.20 | 168.188.45.1 | SYN |
| 2 | 168.188.46.20 | 168.188.45.1 | SYN |
| 3 | 168.188.46.20 | 168.188.45.1 | SYN |
| ..... | | | |
| 13 | 168.188.46.20 | 168.188.45.1 | SYN |
| 14 | 168.188.46.20 | 168.188.45.1 | SYN |
| 15 | 168.188.46.20 | 168.188.45.1 | SYN |

LISHED 상태에서 패킷 생성기(client 역할)와 패킷 분석기(server 역할)가 ACK 패킷을 주고받으면서 정상적인 패킷 전송이 이루어지는지 확인하였다. 계속해서 패킷 생성기에서 패킷 분석기로 FIN 패킷을 보내고 그 이후에도 계속해서 ACK 패킷들을 보내면서 정상적인 패킷 전송이 이루어지는지 확인하였다. 이는 표 6의 테스트 케이스 중 유효한 패킷들만 가지고 테스트를 수행한 경우이다. 테스트를 위해 표 14와 같은 패킷을 생성해 전송하였으며, 패킷 1과 2에 의해 패킷 필터는 ESTABLISHED상태에 도달하게 된다. 그 이후에는 ESTABLISHED 상태에서 패킷을 주고받는다. 그 결과는 그림 11에서 보는 바와 같이 유효한 패킷들에 대해서 패킷 전송이 정상적으로 이루어짐을 확인할 수 있다. 그런데 한 가지 주목할 점은 FIN 패킷을 전송한 이후에도 몇 번의 ACK 패킷들이 정상적으로 전송된다는 것이다. 이런 현상이 발생하는 이유는 FIN 패킷을 보낸다고 해서 곧바로 연결을 해지하지 않고 어느 정도의 시간이 흐를 때까지 ESTABLISHED 상태를 유지하기 때문이다(그림 7의 왼쪽 그림 참조).

비정상적인 상태전이 가능성에 대한 테스트는 Server에서 Client방향으로 SYN 패킷과 ACK 패킷을 보내 Drop됨을 확인하여 패킷 필터의 올바른 동작을 확인하였다. 테스트 결과는 그림 12와 같다. 이 경우에는 가상의 3-way handshaking을 통해 패킷 필터가 ESTABLISHED 상태에 도달하지 않은 상황에서 패킷 전송이 시도되었고, 그 결과로 패킷들이 패킷 필터에 의해 거부(drop)된 것이다.

본 논문에서는 상태기반 테스트를 위의 두 가지 경우만을 보여주지만 ESTABLISHED 상태가 되기 전 상황에서 Client의 접근등의 다양한 방법으로 테스트를 수행하여 결과를 확인하였다.

### 5.4 기존 방법과의 비교

기존에 트래픽 생성기, 포트 스캐너, 취약성 탐지기와 같은 많은 네트워크 보안 테스트 도구들이 존재한다. 각 도구는 그 나름대로의 목적에 따라 중요한 역할을 수행한다. 그러나 본 논문에서 제안한 시스템은 패킷 필터링의 보안 정책에 대한 기능 테스트를 목적으로 하고 있기 때문에 이러한 도구들과는 성격이 다르다. 이에 본 논문의 시스템을 기존의 패킷 필터링 기능 테스트 방법인 수동 테스트와 Ftester를 이용한 테스트 방법과의 비교를 통해 본 논문의 시스템의 우수성을 입증하고자한다. 다음의 표 16은 본 논문에서 제안하는 시스템과 기존 방법들을 비교한 표이다.

위의 표에서 비교한 것처럼 본 논문의 시스템은 패킷 필터링 테스트에 있어 전 과정을 자동화하여 테스트의 효율을 높일 수 있다. 또한 테스트 케이스의 생성 및 테스트 오라클의 생성 과정에 체계적인 소프트웨어 테스팅 기법을 적용함으로써 테스트의 신뢰성을 향상시켰다.

## 6. 결론 및 향후 연구방향

본 논문에서 제안한 테스트 도구는 패킷 필터링 기능을 제공하는 라우터나 방화벽과 같은 보안 솔루션의 보안 정책을 정책별로 기능 테스트할 수 있다. 기존의 기능 테스트 방법은 수동적인 테스트 케이스 생성으로 인

State Analysis

| Client | | Server | Pass/Drop | Result |
|---|---|---|---|---|
| SYN | -> | SYN | pass | SUCCESS |
| SYN ACK | <- | SYN ACK | pass | SUCCESS |
| ACK | -> | ACK | pass | SUCCESS |
| ACK | <- | ACK | pass | SUCCESS |
| ACK | -> | ACK | pass | SUCCESS |
| ACK | <- | ACK | pass | SUCCESS |
| FIN | -> | FIN | pass | SUCCESS |
| ACK | <- | ACK | pass | SUCCESS |
| ACK | -> | ACK | pass | SUCCESS |
| | | | | |

그림 11 상태 기반 테스트 결과

표 14 전송 패킷

| 전송 패킷 | 송신자 IP | 목적지 IP | FLAG |
|---|---|---|---|
| 1 | 168.188.46.20 | 168.188.45.1 | SYN |
| 2 | 168.188.45.1 | 168.188.46.20 | SYN+ACK |
| 3 | 168.188.46.20 | 168.188.45.1 | ACK |
| 4 | 168.188.45.1 | 168.188.46.20 | ACK |
| 5 | 168.188.46.20 | 168.188.45.1 | ACK |
| 6 | 168.188.45.1 | 168.188.46.20 | ACK |
| 7 | 168.188.46.20 | 168.188.45.1 | FIN |
| 8 | 168.188.45.1 | 168.188.46.20 | ACK |
| 9 | 168.188.46.20 | 168.188.45.1 | ACK |

State Analysis

| Client | | Server | Pass/Drop | Result |
|---|---|---|---|---|
| | <- | SYN | drop | SUCCESS |
| | <- | ACK | drop | SUCCESS |
| | | | | |

그림 12 상태 기반 테스트 결과

표 15 전송 패킷

| 전송 패킷 | 송신자 IP | 목적지 IP | FLAG |
|---|---|---|---|
| 1 | 168.188.45.1 | 168.188.46.20 | SYN |
| 2 | 168.188.45.1 | 168.188.46.20 | ACK |

표 16 기존 테스트 방법과의 비교

| | 수동 테스트 | Ftester | 본 논문의 시스템 |
|---|---|---|---|
| 테스트 케이스 생성 | 수동 | 수동 | 자동 |
| 테스트 수행 | 반자동 | 자동 | 자동 |
| 테스트 오라클 생성 | 없음 | 없음 | 생성 |
| 테스트 결과 판단 | 수동 | 일부 자동 | 자동 |
| 테스트 결과의 신뢰성 | 테스터의 역량에 따라 달라짐 | 테스터의 역량에 따라 달라짐 | - 테스터의 역량과 무관<br>- 자동화 방법에 의해 일관된 신뢰성 제공 |

해 테스트의 정확성이 떨어지고 테스트 결과를 판단하기 위하여 로그 분석을 수행해야 하는 어려움과 번거로움이 있었으며 보안 솔루션 사용자들에게 보안과 테스트에 대해 많은 전문 지식을 요구하였다. 하지만 제안한 시스템은 테스트 케이스의 생성과 테스트 오라클의 생성을 자동화하였고, 또한 다양한 능력을 지닌 패킷 생성기와 패킷 분석기를 기반으로 테스트베드를 구축하였다. 게다가 이 생성기와 분석기를 동기화할 수 있는 환경과 방법을 제공함으로써 사용자의 개입 없이 테스트 결과를 판단할 수 있도록 하였다.

본 논문에서 제안한 테스트 방법 및 도구는 테스트 케이스 생성 측면에서 소프트웨어 테스팅의 테스트 케이스 추출 이론 및 방법을 적용하여 테스트 케이스 생성을 체계화하고, 자동화하였으며, 테스트 수행 후 결과 판단 및 테스트 평가 측면에서 테스트 오라클을 자동으로 생성하고 이를 기반으로 테스트 수행 후 결과 판단의 자동화를 실현하였다. 따라서 보안 솔루션의 보안 정책에 대한 테스트 비용과 시간을 크게 줄여주는 혁신적인 방법이라 할 수 있다.

지금까지는 동시에 여러 규칙을 테스트하기보다는 각각의 규칙을 독립적으로 테스트하는 방법을 사용하고 있다. 그렇지만 대부분의 보안 정책은 여러 개의 규칙들로 기술된다. 현재 이와 같이 여러 규칙이 복합적으로 설정된 보안 정책을 테스트 할 수 있도록 규칙 분석기를 확장하고 있다. 또한 여러 규칙을 적용했을 경우, 규칙들 사이의 순서에 의해 동일한 패킷 필터링의 효과에 대해서도 서로 다른 시스템 성능이 나타날 수 있다. 향후 패킷 필터링 시스템의 성능을 향상시키기 위해 규칙의 순서를 최적화하기 위한 방안에 대해서 연구 할 것이다. 이러한 과정과 동반하여 성능 테스팅에 대한 연구도 진행할 계획이다.

## 참 고 문 헌

[1] B. Potter & G. McGraw, "Software Security Testing," *IEEE SECURITY AND PRIVACY MAGAZINE, 2*(5), 2004, 81-85.

[2] E. E. Schultz, "How to Perform Effective Firewall Testing," *COMPUTER SECURITY JOURNAL, 12*(1), 1996, 47-54.

[3] E. E. Schultz, "When Firewalls Fail: Lessons Learned From Firewall Testing," *NETWORK SECURITY*, 1997, 8-11.

[4] J. Wack, "Firewall Testing and Rating," *Proc.of NATIONAL INFORMATION SYSTEMS SECURITY CONFERENCE*, 1996.

[5] Y. H. Cho, S. Navab, & W. H. Mangione-Smith, "Specialized Hardware for Deep Network Packet Filtering," *LNCS2438*, 2002, 452-461.

[6] B. McCarty, *Red Hat Linux Firewalls*(Indianapolis, IN: Wiley Publishing, 2003).

[7] www.0kr.net/files/iptables-tutorial.html

[8] S. Northcutt, and et. al., *Inside Network Perimeter Security* (USA: New Riders Publishing, 2003).

[9] M. R. Lyu & L. K. Y. Lau, "Firewall Security: Policies, Testing and Performance Evaluation," *Proc.of Computer Software and Applications Conference: COMSAC2000*, 2000, 116-121.

[10] www.infis.univ.trieste.it/~lcars/ftest

[11] K. J. Houle & G. M. Weaver, "Trends in Denial of Service Attack Technology," CERT/CC, 2001.

[12] M. L. Hutcheson, *Software Testing Fundamentals: Methods and Metrics* (Wiley & Sons, 2003).

김 현 수

1988년 서울대학교 계산통계학과 학사 1991년 한국과학기술원 전산학과 공학석사. 1995년 한국과학기술원 전산학과 공학박사. 1995년~1995년 한국전자통신연구원 Post Doc. 1996년~2001년 금오공과대학교 조교수. 1999년~2000년 Colorado State University 방문교수. 2001년~현재 충남대학교 전기정보통신공학부 부교수. 관심분야는 소프트웨어 테스팅, 소프트웨어 재공학, 컴포넌트 마이닝, 컴포넌트 테스팅, SOA

박 영 대

2003년 충남대학교 컴퓨터공학교육과 학사. 2005년 충남대학교 컴퓨터공학과 공학석사. 2005년~현재 삼성전자 정보통신총괄 무선사업부 수출 S/W lab. 관심분야는 소프트웨어 테스팅

국 승 학

2004년 충남대학교 컴퓨터과학과 학사 2006년 충남대학교 컴퓨터공학과 공학석사. 2006년~현재 충남대학교 컴퓨터공학과 박사재학중. 관심분야는 소프트웨어 테스팅, 소프트웨어 품질관리, SOA, 웹서비스