https://doi.org/10.33778/kcsa.2024.24.2.079

Spring Security와 Apache Shiro의 CSRF 공격 방어 기법 비교 분석 및 검증★

김 지 오*, 남 궁 다 연*, 전 상 훈**

요 약

본 논문은 웹 애플리케이션의 증가로 인해 소프트웨어 내 보안 취약점을 이용한 사이버 공격이 증가하고 있다. CSRF(Cross-Site Request Forgery) 공격은 특히 웹 사용자와 개발자에게 심각한 위협으로, 사전에 예방해야하는 공격이다. CSRF는 사용자의 동의 없이 비정상적인 요청을 통해 공격을 수행하는 기법으로, 이러한 공격으로부터 웹 애플리케이션을 보호하기 위한 방법은 매우 중요하다. 본 논문에서는 Spring Security와 Apache Shiro 두 프레임워크를 통해 CSRF 방어에 대한 성능을 비교 분석하고 검증하여, 효과적으로 적용 가능한 프레임워크를 제안한다. 실험 결과, 두 프레임워크 모두 CSRF 공격 방어에 성공하였으나, Spring Security는 평균 2.55초로 Apache Shiro의 5.1초보다 더 빠르게 요청을 처리하였다. 이러한 성능 차이는 내부 처리 방식과 최적화 수준의 차이에서 비롯되었으며, 시스템 자원 사용 측면에서는 두 프레임워크 간에 차이가 없었다. 따라서 높은 성능과 효율적인 요청 처리가 요구되는 환경에서는 Spring Security가 더 적합하며, Apache Shiro는 개선이 필요하다. 이 결과는 웹 애플리케이션의 보안 아키텍처 설계 시 중요한 참고 자료로 활용되기를 기대한다.

Comparative Analysis and Validation of CSRF Defense Mechanisms in Spring Security and Apache Shiro

Jj-oh Kim*, Da-yeon Namgoong*, Sanghoon Jeon**

ABSTRACT

This paper addresses the increasing cyber attacks exploiting security vulnerabilities in software due to the rise in web applications. CSRF (Cross-Site Request Forgery) attacks pose a serious threat to web users and developers and must be prevented in advance. CSRF involves performing malicious requests without the user's consent, making protection methods crucial for web applications. This study compares and verifies the CSRF defense performance of two frameworks, Spring Security and Apache Shiro, to propose an effectively applicable framework. The results show that both frameworks successfully defend against CSRF attacks; however, Spring Security processes requests faster, averaging 2.55 seconds compared to Apache Shiro's 5.1 seconds. This performance difference stems from variations in internal processing methods and optimization levels. Both frameworks showed no significant differences in resource usage. Therefore, Spring Security is more suitable for environments requiring high performance and efficient request processing, while Apache Shiro needs improvement. These findings are expected to serve as valuable references for designing web application security architectures

Key words : Web Application, Security, Cross-Site Request Forgery, Spring Security, Apache Shiro

접수일(2024년 05월 31일), 게재확정일(2024년 06월 03일)

* 수원대학교/정보보호학과 (공동주저자)
** 수원대학교/정보보호학과 (교신저자)

★ 본 연구는 2021년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. 2021R1I1A1A01040943l3).

1. 서 론

웹 애플리케이션의 증가와 함께 소프트웨어에 내재된 보안 취약점을 악용하는 사이버공격이 지속적으로 늘어나고 있다[1]. 사이버 침해 사고 신고 통계를 살펴보면 2021년 640건, 2022년 1,142건, 2023년 1,277건으로 꾸준히 증가하고 있다[2].

보안 취약점은 정보 시스템의 보안 침해에 있어 중요한 문제로 대두되고 있으며, 보안 취약점을 예방하고 효율적으로 대응하는 것에 관심이 높아지고 있다[3]. 한국인터넷진흥원 (KISA)는 소프트웨어 개발 생명 주기에서 고려되어야 하는 보안 위협을 최소화하기 위해 수행해야 하는 보안 활동을 제안하였다. 특히, 분석・설계단계 소프트웨어 보안 강화 부분에서는 XQuery 삽입, XSS(Cross-Site Scripting), 사이트 간 요청 위조(Cross-Site Request Forgery, CSRF), HTTP응답 분할, 버퍼오버플로우 등 여러 공격을 언급하며 보안 강화에 주목하고 있다[4]. 본 논문의 연구 범위는 다양한 공격 방법 중 웹 사용자와 개발자에게 심각한 공격인 CSRF를 다룬다.

CSRF는 서버와 사용자에게 들키지 않고 비정상적인 요청을 정상적인 요청으로 속여 사용자가 원하지 않는 행동을 수행하도록 한다[5]. 또한 해당 취약점을 통해 다른 악성 프로그램을 배포하고, 계정 관련 정보를 가로챌 수 있는 등 다양한 공격을 통해 취약점이 확장될 수 있어 보안 공격으로 인해 심각한 피해를 입을 수 있다[6].

CSRF 공격은 시스템 구성을 수정하고, 관리자 계정을 생성하고, 커뮤니티 데이터 게시/삭제, 또는 사용자 프로필 변경과 같은 무단 작업을 수행하는 데 사용된다[7]. 이러한 공격은 감지하기 어렵고 사고가 발생한 후에만 복구가 가능한 경향이 있다. 또한, 현재 사용되고 있는 웹 애플리케이션의 취약점은 수정을 통해 근본적으로 해결해야 한다. 그러나 취약점 제거 시 코딩 전체 부분을 수정해야 하므로 시간과 비용이 많이 들고, 지속적인 통합관리가 불가능하다[8].

이러한 사이버공격의 75% 이상이 애플리케이션 보안 취약점을 악용하고 있다는 점을 고려할 때, 소프트웨어 개발 단계부터 보안 취약점을 유발하는 보안 취약점을 진단하고 제거하는 보안 활동의 필요성 및 중요성이 강조되고 있다[9]. 또한 사고 발생 전까지 공격을 탐지하기 어렵기 때문에 사전에 예방하기 위한 정보보호대책이 필요하다.

본 논문은 CSRF공격을 막기 위해, 방어 프레임워크인 스프링 시큐리티 및 아파치 시로를 활용한 방어 기법을 제안하고 각 기법의 성능을 비교분석한다.

본 논문의 구성은 다음과 같다. 2장에서는 CSRF 공격에 대해 기술한다. 3장에서는 CSRF 방어 방법에 대해 기술한다. 4장에서는 스프링시큐리티와 아파치시로를 직접 적용하여 두 프레임워크의 차이점을 나타내고 비교하여 성능을 검증한다. 5장에서 두 프레임워크에 대한 분석으로 결론 맺는다.

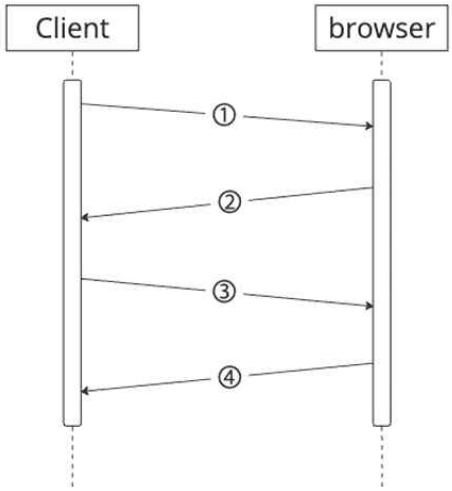
2. CSRF 공격 방법

2.1 일반적인 요청

웹 서버란 웹 브라우저 같은 클라이언트로부터 HTTP(Hypertext Transfer Protocol) 요청을 받아 정적인 웹 페이지를 클라이언트로 보내주는 서버이다. 웹 어플리케이션 서버는 HTTP를 통해 컴퓨터나 장치에 어플리케이션을 수행해주는 미들웨어이다. 본 논문에서 언급되는 서버는 웹 서버와 웹 어플리케이션 서버가 협력적인 구조로 사용되는 서버다. HTTP 요청은 사용자가 URL을 입력하는 순간부터 시작된다. URL을 입력하면 URL에 적힌 값을 파싱하고 HTTP 요청 메시지를 만든다. 브라우저는 네트워크에 요청 메시지를 직접 송출할 수 없기 때문에 만들어진 메시지를 서버로 전송한다. 브라우저는 사용자가 입력한 URL 주소에 해당하는 서버에 도착해서 데이터를 요청하고, 서버에서 응답한 데이터를 받아서 화면에 보여준다.

(그림 1)은 정상적인 웹서버 요청에 대한 핸드쉐이킹(HandShaking) 과정을 나타낸다. 첫째, 유저가 서버에 로그인을 요청하면 브라우저(서버)는 요청의 유효성을 확인하고 unique한 id를 sessionid라는 이름으로 저장한다. 둘째, 브라우저가 응답할 때 응답헤더에 서

버가 응답할 때 응답헤더에 set-cookie: sessionid: 를 추가하여 응답한다. 셋째, 유저는 이후 요청할 때 전달받은 'sessionid: ' 쿠키를 자동으로 요청 헤더에 추가하여 요청한다. 마지막으로 브라우저에서 요청 헤더의 sessionid값을 세션 저장소에서 찾아보고 유효한지 확인한 후 요청을 처리하고 응답한다. 즉, 사용자가 특정 페이지를 웹 서버에 요청하게 되면 웹 서버가 이를 처리한 후 결과를 사용자에게 응답을 하게 되는 구조이다.

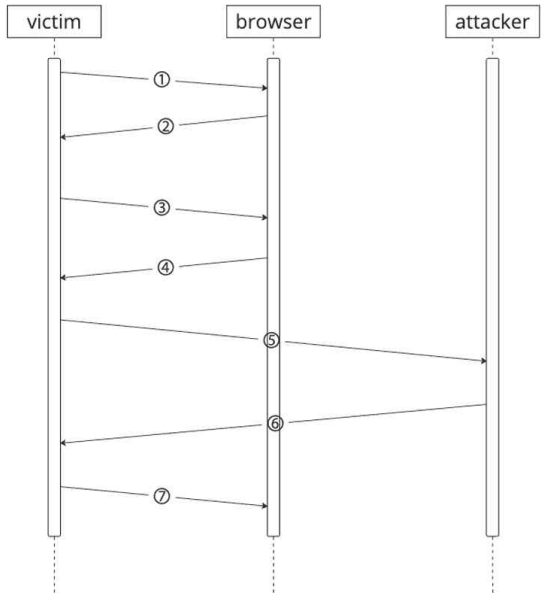


(그림 1. General request)

2.2 CSRF의 요청

Cross-Site Request Forgery (CSRF)는 웹 애플리케이션의 보안 취약점 중 하나로, 사용자가 자신의 의지와는 무관하게 공격자가 의도한 행동을 웹 애플리케이션에 수행하도록 하는 공격 기법이다[10]. 이러한 공격은 사용자가 공격자의 조작된 링크나 스크립트가 포함된 웹 페이지를 방문할 때 발생한다. (그림 2)는 CSRF 공격의 핸드쉐이킹 과정을 나타낸다. CSRF 시도 시, 공격자는 사용자가 로그인한 상태에서 악의적인 요청을 만들어낼 수 있는 웹 페이지나 이메일을 준비해둔다. 사용자가 브라우저에 로그인하여 세션 쿠키를 받고 이 악의적인 콘텐츠에 접근하게 되면, 브라우저는 자동으로 사용자의 세션 쿠키와 함께 공격자가 준비한 요청을 웹 애플리케이션에 전송한다. 브라우저는 이 요청을 사용자의 유효한 요청으로 인식하고, 공격자가 의도한 작업을 실행하게 되는 것이다.

즉, 사용자가 해당 사이트에 로그인한 상태에서 CSRF 공격이 이루어지면, 공격자는 사용자의 세션을 이용하여 웹 서버를 속이고 무단으로 데이터를 변경하거나 민감한 행동을 수행할 수 있게 되는 것이다.



(그림 2. CSRF request)

3. CSRF 방어 모델

3.1 개요

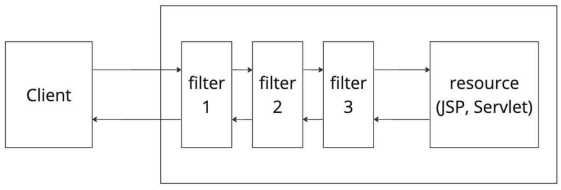
스프링 시큐리티와 아파치 시로를 사용한 방어 기법을 제안하고, 각 방어 기술에 대한 성능 평가를 수행한다. 스프링 시큐리티는 자바 기반의 웹 응용 프로그램을 보호하기 위한 강력한 보안 프레임워크로, 사용자 인증, 권한 부여, 공격 방어 등 다양한 보안 기능을 제공한다. 아파치 시로는 자바 애플리케이션의 인증 및 권한 부여를 처리하기 위한 라이브러리로, 세션 관리, 토큰 기반의 인증, 권한 검사 등을 지원한다. 두 기술의 차이는 주로 사용되는 시나리오와 기능 측면에서 나타난다. 스프링 시큐리티는 스프링 애플리케이션에 녹아들어 있는 반면, 아파치 시로는 독립적으로

사용될 수 있다. 또한, 스프링 시큐리티는 스프링의 다양한 기능과 통합되어 개발자가 보다 편리하게 보안을 구현할 수 있고, 아파치 시로는 다른 자바 애플리케이션과의 통합이 보다 유연하게 이루어질 수 있다는 차이가 있다.

3.2 스프링 시큐리티

스프링은 객체지향 프로그래밍을 도와주는 도구로, 자바 기반의 오픈소스 애플리케이션 프레임워크다[11]. 스프링 시큐리티는 스프링 프레임워크 기반 어플리케이션의 보안을 담당하는 프레임워크로 필터 체인 기반으로 동작한다[12]. 스프링 시큐리티는 인증과 인가에 대한 부분을 필터 흐름에 따라 처리하고 있다. 인증은 해당 사용자가 본인이 맞는지 확인하는 절차이며, 인가는 사용자가 요청한 자원에 접근 가능한지를 결정하는 절차이다. 스프링 시큐리티는 기본적으로 인증 절차를 거친 후에 인가 절차를 진행하게 되며, 인가 과정에서 해당 리소스에 대한 접근 권한이 있는지 확인을 하게 된다. 보안과 관련해서 체계적으로 많은 옵션을 제공해주기 때문에 보안 관련 로직을 일일이 구현하지 않아도 된다는 장점이 있다.

(그림 3)은 스프링 시큐리티 필터를 보여준다. 사용자는 요청을 보내고, 그 요청을 서블릿이나 JSP(Java Server Pages)등이 처리한다. 요청을 받기 전, 여러 필터들을 거칠 수 있는데 필터는 클라이언트와 자원 사이에서 요청과 응답 정보를 이용해 다양한 처리를 하는 것에 목적이 있다. 각 필터는 인증을 위한 로그인 폼 URL 감시, HTTP 기본 인증 헤더를 감시 및 처리, 인증된 사용자와 관련된 세션을 추적하는 등 다양한 역할을 수행한다.

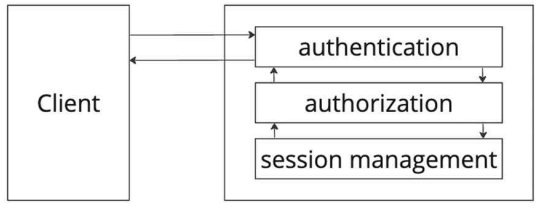


(그림 3. Spring security)

3.3 아파치 시로

아파치 시로는 자바로 개발된 오픈소스 보안 프레임워크이다. 이 프레임워크는 네 가지의 보안 기본 원칙인 사용자 인증, 권한 부여, 세션 관리, 암호화 및 해싱 기능을 수행하기 위한 보안 API를 제공한다. 언급된 기능 외에 유닛테스트, 멀티 스레딩 같은 보조 기능도 지원하지만, 이 모든 것은 네 가지 보안 기본 원칙을 강화하기 위함이다. 아파치 시로는 세션 관리를 통해 사용자의 상태를 추적하고 세션 유효성을 유지하거나 OAuth(Open Authorization), JWT(Json Web Token) 와 같은 외부 인증 시스템과의 협업도 가능하다. 또한 다양한 자바 프레임워크와의 통합을 지원하여 환경에 따라 쉽게 적용할 수 있다는 장점이 있다[13].

(그림 4)는 아파치 시로를 나타낸다. 사용자로부터 요청을 받으면 해당 요청은 아파치 시로로 전달된다. 아파치 시로는 전달받은 요청을 인증, 인가, 세션 관리 등의 모듈로 분리하여 처리한다. 인증 모듈은 사용자의 신원을 확인하고, 인가 모듈은 사용자의 권한을 확인하며, 세션 관리 모듈은 사용자의 세션을 추적하고 관리한다. 처리된 요청은 다시 전달되어 응답을 생성한다.



(그림 4. Apache Shiro)

4. 실험 및 평가

이 장에서는 두 프레임워크의 성능을 평가하고, 비교하기 위한 평가 환경과 방법을 설명한다.

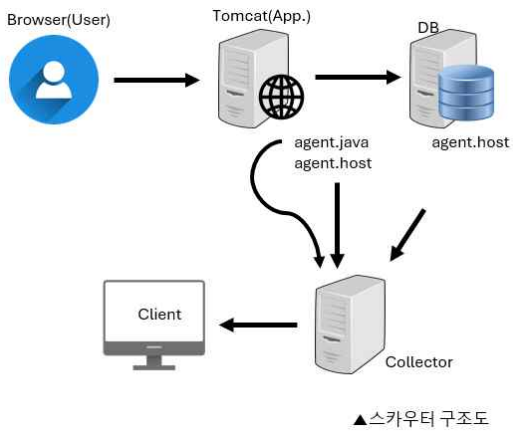
4.1 평가 환경

평가에 사용한 컴퓨터는 Apple이 직접 설계한 M1칩을 탑재하고 있으며, Mac OS 운영체제 14.3.1 버전이

다. 이는 8코어 CPU, 14코어 GPU 그리고 16코어 Neural Engine, 16GB 통합 메모리, 200GB/s 메모리 대역폭으로 구성되어 있다. 해당 컴퓨터로 서버 구성, 실험 및 성능 모니터링을 진행하였다.

4.2 평가 기구

평가 기구는 Scouter 프로그램을 사용한다. Scouter는 LG CNS에서 개발한 APM(Application Performance Monitoring) 도구이다. APM은 어플리케이션의 성능을 관리하고 통제하는 도구로 기록에 근거해 현재 서비스에서 수용 가능한 트래픽을 예상하거나 서비스 구간별로 성능을 기록해 병목을 파악할 수 있다. (그림 5)는 스카우터 구성도로 agent.java는 각종 성능 정보를 수집하고 스카우터 서버로 전달하는 역할이며, agent.host는 CPU, Memory, 디스크 성능 정보를 Collector로 전송한다. Collector는 agent로부터 실시간 모니터링 정보를 수집, 가공하여 통계정보관리 및 장애/에러 경보 등 관리에 필요한 각종 기능을 수행한다. 이로부터 수집된 정보를 확인하는 것은 Client 프로그램이 담당한다. 즉, Agent가 정보를 수집하여 Server에 전달하면 Server는 Client에 스트링하는 구조이다. 본 논문의 실험에선 Scouter를 사용해 성능과 트래픽 기록을 추출하여 각 프레임워크의 차이를 비교한다.



(그림 5. Scouter)

4.3 평가 방법

공격은 POST 요청이며, 해당 요청은 HTTP 메서드로 리소스를 생성/변경하기 위해 설계되어 HTTP 메시지의 Body에 담아서 전송한다. POST요청인 경우 취약한 웹 애플리케이션의 Endpoint로 요청을 전송하고, 요청에 사용되는 모든 매개변수와 값이 숨겨진 hidden 필드로 포함된 폼을 만들어야 한다. 스크립트를 사용하면 사용자의 폼 제출 등의 행동 없이도 폼이 자동으로 제출되게 할 수 있다.

공격자는 악의적인 웹사이트에서 특수한 스크립트를 이용하여 피해자의 브라우저를 조작하고 이 스크립트는 피해자의 브라우저를 통해 피해자가 로그인되어 있는 웹사이트로 요청을 보낸다. 해당 요청은 피해자가 인증된 상태 즉, 로그인된 상태로 보내지기 때문에 웹사이트는 유효한 요청으로 처리하게 된다.

(그림 6)은 공격 예시를 보여준다. 공격자는 비밀번호 설정 변수 value 값을 바꾸려는 값인 1234로 적는다. input type을 hidden으로 설정해두고, 사용자가 이 form 태그를 클릭하면 POST 방식으로 파라미터를 넘기게 된다. 결국 사용자가 form 클릭 시, 자신도 모르게 비밀번호를 바꾸게 되고 공격자는 해당 사용자의 비밀번호를 알게 되는 것이다.

```
<form action = "비밀번호를 변경하는 페이지의 URL" method = "POST"> <input type = "hidden" name = "password" value = "1234"/> <input type = "submit" value = "click" /> </form>
```

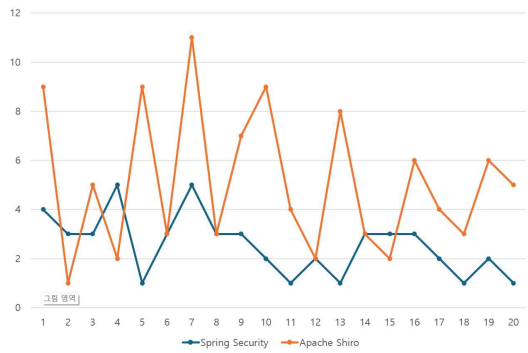
(그림 6. 공격 예시)

4.4 보안 성능 평가

정보시스템의 대표적인 성능 지표는 처리량(Throughput)과 소요 시간(Elapsed Time)이다. 처리량은 시스템 측면의 대상 시스템에 대한 성능을 평가하는 KPI이며, TPS단위를 사용하는데 시간당 대상 시스템에서 처리되고 있는 요청 건수를 의미한다. 소요 시간은 해당 배치작업을 완료할 때까지 소요된 시간으로 정의한다. 성능 평가는 각 프레임워크를 적용한 상태에서 CSRF 공격을 20번 시행하여 Elapsed Time과 TP

S, CPU, Memory 사용률을 측정하였다. (그림 7)은 성능 평가 결과를 그래프로 나타낸다. 그리고 <표 1>에서 20번의 성능 평가 결과를 종합하였다.

스프링 시큐리티를 적용한 상태에서 Elapsed Time은 평균 약 2.55초, TPS는 0.07초로 나타났다. 아파치 시로를 적용한 상태에서는 Elapsed Time은 평균 약 5.1초, TPS는 0.07초로 나타났다. CPU와 Memory 사용률 또한 TPS와 마찬가지로 동일 하였다.



(그림 7. Comparison of Elapsed Time)

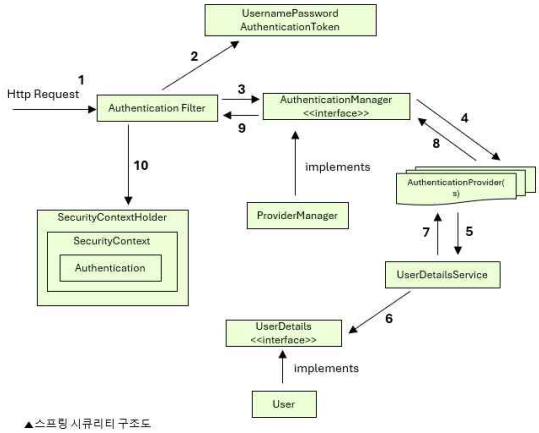
<표 1. Experimental Results>

Performance Metric	Sping Security	Apache Shiro
Average Elapsed Time [Second]	Approx. 2.55	Approx. 5.1
TPS [Second]	Approx. 0.07	Approx. 0.07

4.5 스프링 시큐리티 구조분석

(그림 8)은 스프링 시큐리티 구조를 나타낸다. 스프링시큐리티에서 어플리케이션 보안을 구성하는 두 가지 영역, 인증과 인가가 있다. 인증은 해당 사용자가 본인이 맞는지 확인하는 과정이며, 인가는 해당 사용자가 요청하는 자원을 실행할 수 있는 권한이 있는가를 확인하는 과정이다. 스프링 시큐리티는 기본적으로 인증 절차를 거친 후에 인가 절차를 진행하며, 인가 과정에서 해당 리소스에 접근 권한이 있는지 확인하게 된다.

사용자가 로그인 정보와 함께 Http 인증 요청을 한다. 필터를 통해 AuthenticationToken을 AuthenticationManager로 위임한다. AuthenticationManger는 등록된 AuthenticationProvider들을 조회하며 인증을 요구하며 실제 데이터베이스에서 사용자 인증정보를 가져오는 UserDetailsService에 사용자 정보를 넘겨준다. 넘겨받은 사용자 정보를 통해 데이터베이스에서 찾아낸 사용자 정보인 UserDeatails 객체를 만들고, User 객체의 정보들을 UserDetails가 UserDetailsService로 전달하여 사용자 정보를 비교한다. 인증이 완료되면 권한 등의 사용자 정보를 담은 Authentication 객체를 반환하여 다시 최초의 AuthenticationFilter에 Authentication 객체가 반환되며 인증이 끝나게 된다. 그 후 Authentication 객체를 Security Context에 저장한다. 최종적으로 SecurityContextHolder는 세션 영역에 있는 SecurityContext에 Authentication 객체를 저장하게 되는데 사용자 정보를 저장한다는 것은 스프링 시큐리티가 전통적인 세션-쿠키 기반의 인증 방식을 사용한다는 것을 의미한다.

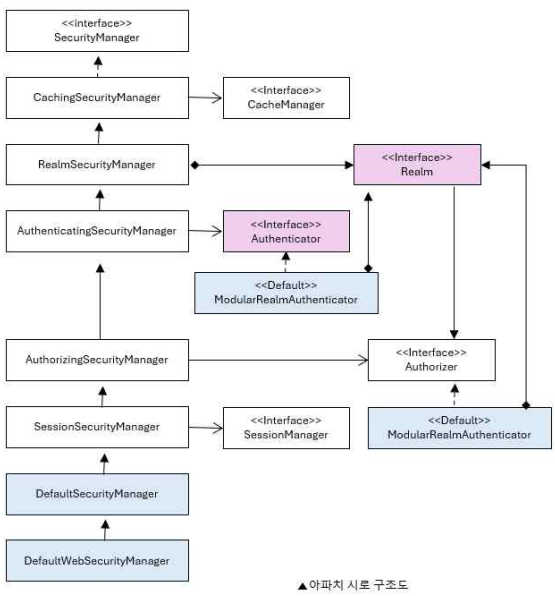


(그림 8. Sping Security Architecture)

4.6 아파치 시로 SecurityManager 구조분석

(그림 9)은 아파치 시로의 구조를 나타낸다. Security Manager 인터페이스는 인증, 권한 검사, 세션 관리와 관련된 모든 기능을 정의하고 있다. 하위 클래스들은 각각 특정 역할을 수행하며, 실제로 개발자가 사용하

는 구현 클래스는 DefaultSecurityManager와 DefaultWebSecurityManager이다. RealmSecurityManager는 Realm 목록을 관리해주고 AuthenticatingSecurityManager는 인증처리를 Authenticator에 위임하는 기능을 제공한다. AuthorizingSecruityMansger는 권한 검사처리를 Authorizer에 위임하는 기능을 제공하며 SessionSecurityManager에서 세션을 관리한다. DefaultSecruityManager는 기본 구현 클래스로, 기억하기 기능을 추가로 제공하며 DefaultWebSecurityManager는 웹 어플리케이션에서 사용되는 구현 클래스이다. DefaultSecurityManager클래스를 사용하면 인증 처리와 권환 검사 처리는 각각 Authenticator와 Authorizer에 위임하므로 사용에 맞게 구현클래스를 제공하면 된다. 또한 Authenticator와 Authorizer를 따로 지정하지 않으면 각각 ModularRealmAuthenticator와 ModularRealmAuthorizer를 구현 클래스로 사용하며, 이 두 ModularRealm 객체는 다시 Realm에게 인증 처리나 권한 처리를 위임한다.



(그림 9. Apache Shiro Architecture)

4.7 CSRF 방어

두 프레임워크의 CSRF 방어 원리는 비슷하다. 세팅해둔 실험 환경에 사용자가 처음 접근하거나 세션이 생성될 때, 서버는 난수 생성기를 사용하여 고유한 CSRF 토큰을 생성한다. 생성된 CSRF 토큰은 사용자 세션에 저장되고 이를 통해 서버는 각 사용자의 요청마다 고유한 CSRF 토큰을 유지할 수 있게 된다. CSRF 토큰은 웹 브라우저로 전달되고 이때, 서버가 렌더링하는 모든 HTML폼이 CSRF 토큰을 숨겨진 필드에 포함하게 된다. 서버는 해당 요청인 폼 필드에서 CSRF 토큰을 추출하고, 서버는 요청에서 추출한 CSRF 토큰과 세션에 저장된 CSRF 토큰을 비교한다. 두 토큰이 일치하면 요청을 정상적으로 처리하고 일치하지 않으면 요청을 거부하게 되는 원리인데, 본 실험 공격의 CSRF 토큰과 사용자의 세션에 저장된 정상적인 CSRF 토큰이 일치하지 않아 공격을 막게 되는 것이다. Apache Shiro는 앞서 설명한 기능이 기본적으로 제공되지 않기에 개발자가 직접 구현해야 한다. 그러나 Spring Security는 CSRF 토큰이 포함되어 있는지 확인하고, 추출한 토큰과 저장된 토큰이 일치하는지 검증하는 일을 'CsrfFilter'라는 특수한 필터를 통해 처리한다. CSRF 보호를 기본적으로 제공하여 자동 처리 필터를 사용하게 되는 것이다.

5. 결 론

본 연구에서는 Spring Security와 Apache Shiro의 CSRF 공격 방어 성능을 비교 분석하였다. 실험 결과, 두 프레임 워크 모두 CSRF 공격을 성공적으로 방어하는 데 있어 TPS, CPU, Memory는 동일했으나, 평균 소요시간에서 차이를 보였다. Spring Security는 약 2.55초로 Apache Shiro의 약 5.1초보다 더 짧은 시간이 소요되었다. 이러한 차이가 발생한 주요 원인은 두 프레임워크의 내부 처리 방식과 최적화 수준에 있다. Spring Security는 CSRF 토큰을 생성하고 검증하는 과정에서 보다 효율적인 알고리즘과 메커니즘을 사용하여 요청 처리 속도를 높인 반면, Apache Shiro는 상대적으로 거쳐 가야 할 필터가 많아 최적화에 적합하지 않았다.

결론적으로, Spring Security는 CSRF 공격 방어에 있어 보다 빠르고 효율적인 성능을 제공하며, 시스템 자원 사용 측면에서의 차이는 없었다. 따라서 높은 성능과 효율적인 요청 처리를 요구하는 환경에서는 Spr

ing Security를 사용하는 것이 더 적합할 것으로 판단된다. Apache Shiro는 특정 상황에서 유용할 수 있으나, Spring Security와의 추이를 보면 CSRF 방어 성능과 자원 사용 측면에서 개선이 필요할 것으로 보인다. 이러한 결과는 웹 애플리케이션의 보안 아키텍처를 설계할 때 중요한 참고 자료로 활용될 수 있을 것이다.

참고문헌

[1] Donghwi Lee, “A Study on Web Service Security Testing Methodology for Performance Evaluation“, Journal of information and security, Vol. 10, No. 4, pp. 31-37, 2010.

[2] “Cyber Threat Trends Report”, KISA, 2024.

[3] Joonseon Ahn, Byeong-Mo Chang, Eunyoung Lee, “Quantitative Scoring System on the Importance of Software Vulnerabilities,” Journal of The Korea Institute of Information Security & Cryptology, Vol. 25, No. 4, 2015.

[4] “2021 Software Development Security Guide”, KISA, 2021.

[5] Sung-Jin Kim and Kyung-Goo Doh, “Source-code-level Defense against CSRF Attacks,” Korean Institute of Information Scientists and Engineers, Vol. 37, No. 2, pp. 56-61, 2010,

[6] Kyung-ae Kim, “Is it possible to insert malicious code when writing the next cafe post on the portal?”, 2024.https://www.boannews.com/media/view.asp?idx=49918&page=1&kind=1

[7] Xiaoli Lin, Pavol Zavarsky, Ron Ruhl1, Dale Lindskog, “Threat Modeling for CSRF Attacks”, IEEE, Vol.3, pp. 486-491, 2009.

[8] Kihwan KimO, Dongil Lee, Hyunbin Lee, Yongtae Shin, “Web application security management integrated platform development study”, KSCI, Vol. 26, No. 1, pp. 85-86, Feb. 2018.

[9] “2023 National Cybersecurity White Paper”, KISA, 2023.

[10] Jin-hyeon Park, Im Y. Jung, and Sun-ja Kim, "Enhanced CSRF Defense Using a Secret Value Between Server and User," The Journal of Korean Institute of Communications and Information Sciences, Vol. 39, No. 3, pp. 162-168, 2014.

[11] Ill Deung Yang and Seong Ryeol Kim, "A Proposal Of The Horizontality Development Method On The Spring MVC," Journal of the Korea Institute of Information and Communication Engineering, Vol. 19, No. 10, pp. 2350-2358, 2015.

[12] Chan-Yeong Gwak, Ryeo-Won Kim, Ho-Yoon Kim, and Seung-Soo Shin, "A Study on Single Sign On using Spring Security", Proceedings of KIIT Conference, pp. 495-496, 2021.

[13] Ochoa, Javier. "Security for Java Web Applications Using Apache Shiro", Helsinki Metropolia University of Applied Sciences, 2014.

〔저자소개〕

김 지 오 (Ji-oh Kim)
2020년 3월~현재: 수원대학교 정보보호학과 학사과정
email : sjgio0515@naver.com

남 궁 다 연 (Da-yeon Namgoong)
2021년 3월~현재: 수원대학교 정보보호학과 학사과정
email : cookie_31@naver.com

전 상 훈 (Sanghoon Jeon)
2012년 2월: 경북대학교 IT대학 심화전자공학 공학사
2014년 2월: 대구경북과학기술원 정보통신융합공학전공 공학석사
2020년 8월: 대구경북과학기술원 정보통신융합전공 공학박사
2020년 3월~8월: 한양대학교 산학협력단 선임연구원
2020년 9월~2022 9월: 한양대학교 의과대학 응급의학과 포닥연구원
2022년 10월~2023 9월: 한양대학교 의과대학 응급의학과 연구조교수
2023년 10월~현재: 수원대학교 지능형SW융합대학 정보보호학과 조교수
email : shjeon@suwon.ac.kr