웹 애플리케이션 보안에 관한 고찰

정진미°, 김영국
한국지질자원연구원°, 충남대학교 컴퓨터공학과
jmjung@kigam.re.kr°, ykim@cnu.ac.kr

Consideration for the Web Application Security

Jinmi Jung°, Young-Kuk Kim
Korea Institute of Geoscience and Mineral Resources°, Chungnam National University

요 약

최근 들어 웹 애플리케이션의 사용이 많아짐에 따라 웹의 취약성을 이용한 공격이 증가하고 있다. 공격의 양상 또한 풍부한 자료와 쉽게 사용할 수 있는 해킹 도구 등을 통해 점점 지능화되고 있어 기존 보안 솔루션의 방어 기능을 무력하게 만들고 있다. 이에 본 논문에서는 웹 애플리케이션의 취약점과 그에 대한 대응책을 살펴보고자 한다.

1. 서 론

애플리케이션 보안은 조직이 그들의 비즈니스를 행하는 데 있어 요구되는 소프트웨어를 신뢰할 수 있도록 기술 수준에서 소프트웨어에 대한 위협, 공격, 취약점, 그리고 대책에 초점을 둔 공학적 규율이다[1]. 중요 애플리케이션에 대한 공격의 성공은 그 애플리케이션이 인터넷 또는 인트라넷 상에 있든지 그것이 웹 애플리케이션, 웹 서비스, 그리고 리치클라이언트 등인지에 상관 없이 막대한 손실을 초래하는데 지난 10여 년 동안 네트워크 보안에 대한 향상으로 인하여 공격자의 관심이 점차 커스텀 애플리케이션으로 옮겨가고 있다.

특히, 근래에 와서 대부분의 기업이나 기관들이 다양한 서비스 제공 요구와 웹 서비스의 유용성 및 용이성 때문에 고객, 직원, 협력사 등과의 거래를 행하는 데 있어 웹 애플리케이션을 사용하는 경우가 많아지면서 웹의 취약성을 이용한 공격이 증가하고 있는 추세다.

Gartner Group은 오늘날 사이버 공격의 75%가 애플리케이션 레벨에서 행해진다고 추정한다[2]. 또한 300개 이상의 웹 사이트를 조사한 결과 약 97% 정도가 웹 애플리케이션 공격에 취약한 것으로 보고 있다. US Federal Bureau of Investigation 역시 기업의 95%가 웹 애플리케이션 공격을 당했고 그 중 5%만이 그 사실을 인지했다고 밝혔다.

현재 VISA/MasterCard/AmEx PCI, FISMA/NIST 800-53, OWASP, Sarbanes-Oxley, HIPAA 그리고 GLBA 등을 비롯한 많은 산업계에서 애플리케이션 수준의 보안에 관한 표준과 규정을 개발하고 있다. 이 중 OWASP(Open Web Application Security Project)는 아래의 [표1]과 같은 가장 심각한 10대 웹 애플리케이션 보안 취약점을 제시하고 그에 대한 대응책을 제공하고 있다[3].

[표1. 가장 심각한 10대 웹 애플리케이션 보안 취약점]

2007년 OWASP Top 10
A1. 크로스 사이트 스크립팅(XSS)
A2. 인젝션 취약점
A3. 악성 파일 실행
A4. 불안전한 직접 객체 참조
A5. 크로스 사이트 요청 변조(CSRF)
A6. 정보 유출 및 부적절한 오류 처리
A7. 취약한 인증 및 세션 관리
A8. 불안전한 암호화 저장
A9. 불안전한 통신
A10. URL 접속 제한 실패

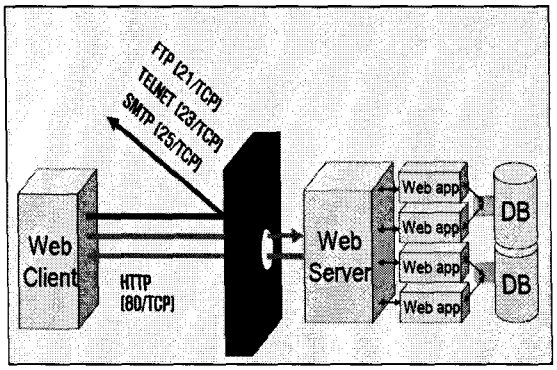
본 논문에서는 OWASP에서 제시한 취약점과 대응책을 바탕으로 안전한 웹 애플리케이션 구축을 위해 고려해야 할 사항들을 살펴본다. 본 논문의 구성은 2장에서 근래의 웹 애플리케이션 공격 동향을 짚어보고 3장에서는 웹 애플리케이션 보안에 요구되는 사항을 정리하며 4장에서 결론을 맺는다.

2. 웹 애플리케이션 공격 동향

최근 웹 애플리케이션 공격의 양상은 풍부한 웹 해킹 자료와 손쉽게 사용할 수 있는 해킹 도구 등으로 인하여 점점 지능화돼 가고 있다. 웹 해킹의 초기 단계에는 공격 기술에 대한 이해 없이 NIKto, N-Stealth와 같은 단순한 CGI 스캐너를 통해 공격을 했고,

이러한 공격은 IDS, 방화벽, L7 스위치 등의 보안 솔루션에 존재하는 기능으로 방어가 가능했다. 하지만 요즘은 웹 프로그래밍에 대한 지식과 다양한 분석 도구를 사용하여 난이도 높은 공격을 시도하고 있다. 이는 공격 탐지가 어렵고 기존 보안 솔루션(방화벽, IDS, IPS)의 방어 기능을 넘어선다.

웹은 서비스를 하기 위해서 80포트(HTTP)나 443포트(HTTPS) 같은 통로를 열어야 하는 근본적인 구조상의 취약점을 안고 있다. 때문에 [그림 1]과 같이 컨텐츠 필터링 기능이 추가된 방화벽이라 할지라도 허용된 포트를 통한 공격을 막을 수 없다[4].



[그림 1] 웹 방화벽의 한계

IDS(Intrusion Detection System)는 제어가 아닌 감시 기능만 수행하며 알려지지 않은 공격은 차단이 불가능하다. IPS(Intrusion Prevention System) 역시 웹 보안을 위한 검사 기능이 미약하다. 또한 완벽한 보안 프로그래밍을 구성했다 하더라도 수정으로 인하여 변경될 경우 취약점이 발생하게 되며, 최근의 웹 해킹의 유형은 이미 알려진 공격에 대한 패턴이 아니라 새롭게 시도되는 해킹 수법이라 이에 대한 대처 방안이 필요한 실정이다.

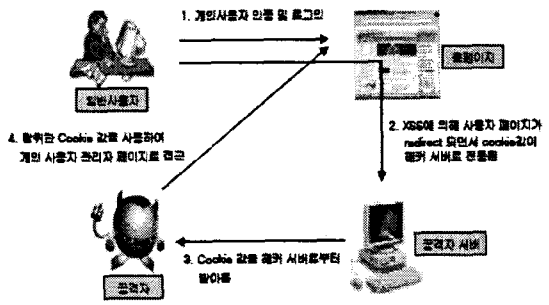
3. 웹 애플리케이션 보안

웹 애플리케이션은 단순히 정보를 검색하는 것에서부터 은행 계좌의 돈을 거래하는 인터넷 뱅킹까지 다양한 형태로 존재하지만, 일반적으로 안전한 웹 프로그래밍을 위해서는 공통적으로 유념해야 할 몇 가지 사항이 있다. 첫째, 클라이언트 측 데이터를 믿지 말라는 것이다. 웹 서버 상에서 운영되는 소프트웨어는 언제 어디서나 누구든지 접근 가능하기 때문에 인증된 사용자와 공격자를 구별하기 어렵다. 둘째, 믿을 수 있는 암호 알고리즘을 사용해야 한다. 자체적으로 만든 알고리즘이 아무리 안전하다고 생각될 지라도 사용을 지양하고 DES, Triple-DES, Blowfish, MD5, SHA1 등과 같이 널리 이용되고 있는 알고리즘을 사용한다. 셋째, 자바스크립트 또는 ActiveX를 이용한 인증 메커니즘은 피한다. 넷째, 새로운 암호 또는 위험도가 높은 일을 행하기 전에는 반드시 인증을 다시 한다. 그리고, 입력 데이터는 제한을 두고 sanity 체크를 하도록 한다.

OWASP는 위와 같은 사항을 좀 더 체계적으로 분류하여 앞서 기술한 바와 같이 10가지의 웹 애플리케이션 취약점과 그에 대한 대응책을 제공하고 있다.

3.1 크로스 사이트 스크립팅(XSS)

XSS는 HTML 인젝션의 한 부분으로 가장 치명적인 것으로 알려진 보안 문제이다. 이는 콘텐츠를 암호화나 검증하는 절차 없이 사용자가 제공하는 데이터를 애플리케이션에서 받아들이거나, 웹 브라우저로 보낼 때 발생한다. 자바스크립트 등 클라이언트 측에서 실행되는 언어로 작성된 악성 스크립트 코드를 웹 페이지, 웹 게시판 또는 이메일에 포함시켜 사용자에게 전달하면, 해당 웹 페이지나 이메일을 사용자가 클릭하거나 읽을 경우 악성 스크립트 코드가 실행되는데 공격자는 이를 통해 웹 사이트 변조, 악의적인 컨텐츠 삽입, 피싱 공격 등의 행위를 할 수 있다. [그림 2]는 이러한 취약점을 이용하여 공격자가 제3자의 쿠키 정보를 획득한 후 관리자 페이지로 접근하는 과정을 보여주고 있다[5].



[그림 2] 제 3자의 쿠키 정보 탈취

XSS를 위한 최선의 방어책은 모든 입력 데이터의 검증과 출력 데이터를 암호화하는 것이다. 데이터가 보여지거나 저장되기 전에 표준 입력 값 검증 방식을 통해 데이터의 길이, 형태, 문법과 비즈니스 규칙 등에 대해 검증한다. 널을 비롯한 특수 문자가 입력되는 경우 HTML 문자로 변경하거나 제거하도록 한다. 출력 값은 사용자에게 제공되기 전에 모든 문자를 적절하게 암호화한다. 또한 ISO 8859-1, UTF 8 과 같은 출력 값 암호화를 명시함으로써 공격자가 암호화를 선택하도록 허용하지 않는다.

3.2 인젝션 취약점

인젝션 취약점, 특히 SQL 인젝션은 웹 애플리케이션에는 일반적이다. 인젝션의 종류에는 SQL, LDAP, XPath, XSLT, XML, OS 명령어 인젝션 등과 같은 것들이 있다. 인젝션은 사용자가 입력한 데이터가 명령어나 질의문의 일부분으로 인터프리터에 보내질 때 발생한다. 공격자는 이 취약점을 이용함으로써 애플리케이션에서 이용 가능한 임의의 데이터를 읽고, 생성하고 갱신하며 삭제할 수 있게 된다. 최악의 경우 애플리케이션을 완전히 손상시켜 시스템의 동작을 멈추게 하거나 방화벽 환경을 우회하도록 허용한다.

이 취약점에 대한 대응 방법으로는 매개 변수화된 질의어와 객체 관계 매핑(ORM) 라이브러리와 같은 API를 사용하는 것이다. 이러한 인터페이스들은 모든 데이터의 벗어난 동작을 다루거나 또는 벗어나도록 요청하지 않는다. [그림 3]은 hibernate의 API를 이용하여 데이터베이스로부터 조회된 항목의 총 개수를 반환하는 SQL문을 작성한 예이다.

```
public int getTotalCount(int ORGANIZEID, String DELFLAG) throws
        ServiceException
{
    Session session = Initialize.currentSession();
    int totCount = 0;
    try
    {
        totCount = session.createCriteria(ENT_CUSTOMERINFO.class)
                  .add(Expression.eq("ORGANIZEID", ORGANIZEID))
                  .add(Expression.eq("STATE", DELFLAG))
                  .list().size();
    }
    catch (Exception e)
    {
        throw new ServiceException(e);
    }
    finally
    {
        Initialize.closeSession();
    }
    return totCount;
}
```

[그림 3] ORM을 이용한 SQL문 작성 예

3.3 악성 파일 실행

원격 파일 인젝션(RFI)에 취약한 코드는 공격자가 악의적인 코드와 데이터를 삽입하는 것을 허용함으로써 전체 서버 훼손과 같은 파괴적인 공격을 가할 수 있다. 악성 파일 실행 공격은 PHP, XML, 그리고 사용자로부터 파일명이나 파일을 받아들이는 프레임워크에 영향을 준다.

PHP를 예로써 [그림 4]와 같은 코드는 단순히 원격 공격 스크립트가 실행되도록 허용할 뿐만 아니라, PHP가 윈도우 환경에서 실행된다면 PHP 파일시스템 래퍼의 SMB 지원을 토대로 로컬 파일 서버에 접근하는 데 사용될 수 있다. 이 외에 공격자는 세션 파일, 이미지, 그리고 로그 데이터의 업로드를 통해 악의적인 데이터를 업로드할 수 있고, 압축이나 오디오 스트림을 사용하여 원격 자원에 접근할 수도 있다.

```
include $_REQUEST['filename'];
```

[그림 4] RFI에 취약한 PHP 코드 예

원격 파일 실행 취약점에 대응하기 위해서는 구조 설계와 디자인 전반에 걸쳐 철저한 검사를 하고, 특히 다음과 같은 사항을 고려해야 한다.

- 간접 객체 맵을 사용한다
- 명시적인 값 변조 확인 기법을 사용한다
- 사용자 입력 값을 검증한다
- 매우 가치가 높은 시스템에 대해 연결된 VLAN이나 독립망으로부터 웹 서버를 격리한다
- 사용자가 제공하는 파일이나 파일명을 확인한다

3.4 불안전한 직접 객체 참조

직접 객체 참조는 개발자가 파일, 디렉토리, 데이터베이스 기록 혹은 키 같은 내부 구현 객체에 대한 참조를 URL 혹은 폼 매개변수를 통해 내부에 구현된 객체를 노출시킬 때 발생하는데, 접근 권한 확인이 적절히 이뤄지지 않으면 공격자는 이러한 참조를 공격함으로써 인증 없이도 다른 객체에 접근할 수 있다. [그림 5]에서 보여지는 PHP 코드처럼 사용자의 입력이 특정 파일 이름이나 경로를 명시할 수 있도록 되어 있다면, 공격자는 애플리케이션의 다른 디렉토리로 이동하여 다른 리소스에 접근할 수 있다.

```
<select name="language">
    <option value="fr">Franqais</option>
</select>
...
requere_once ($_REQUEST('language')."lang.php");
```

[그림 5] 불안전한 직접 객체 참조의 코드 예

이러한 코드는 웹 서버의 파일 시스템 내에 존재하는 임의의 파일에 접근하기 위해서 "../../../../etc/passwd%00" 과 문자열을 이용한 null byte 인젝션 공격을 받을 수 있다. 데이터베이스 키에 대한 참조 역시 자주 드러나는데 공격자는 다른 유효한 키를 찾거나 추측하는 것으로 이러한 매개변수를 공격할 수 있다. 그리고 매개변수는 대개 연속적인 값을 갖는 속성이 있어 공격자가 원하는 어떤 값으로도 변경할 수 있다.

이 취약점에 대한 가장 좋은 보호 방안은 인덱스, 간접 참조 맵 또는 검증이 쉬운 다른 간접적인 방법을 사용하여 사용자에게 직접 객체 참조가 노출되지

않도록 하는 것이다. 직접 객체 참조를 반드시 사용해야 한다면, 사용자가 그것을 사용하기 전에 필히 인증 과정을 거쳐야 한다. 매개변수 변조 공격을 막기 위해서는 http://www.test.com/application?file=1 과 같이 인덱스 값이나 참조 맵을 사용한다. 또한 데이터베이스 구조에 대한 직접 참조를 노출해야 한다면, [그림 6]과 같이 권한이 부여된 레코드만 허용하도록 해야 한다.

```
int cartID = Integer.parseInt(request.getParameter("cartID"));
User user = (User)request.getSession().getAttribute("user");
String query = "SELECT * FROM table WHERE cartID="+cartID
              +"AND userID="+user.getID();
```

[그림 6] 데이터베이스 직접참조 노출 시 보호 코드 예

3.5 크로스 사이트 요청 변조(CSRF)

CSRF 공격은 로그온한 사용자의 브라우저가 사전 승인된 요청을 취약한 웹 애플리케이션에 보내도록 함으로써 발생한다.

이에 대한 대응책으로는 애플리케이션이 사용자를 검증하는 데 있어 브라우저에 의해 자동으로 제출된 증명이나 토큰에 의존하지 않고 브라우저가 기억할 수 없는 커스텀 토큰을 사용하는 것이다. 또한 민감한 데이터나 값의 트랜잭션 수행을 위해 GET 요청의 사용을 지양하고 POST 방식을 사용한다.

3.6 정보 유출 및 부적절한 오류 처리

애플리케이션은 다양한 문제점을 통해 의도하지 않게 애플리케이션 자체의 구성 및 내부 작업에 대한 정보를 누출하거나 개인정보를 침해할 수 있다. 또한 상세한 에러 메시지나 디버그 관련 메시지를 통해서도 정보가 유출될 수 있다. 공격자는 이러한 취약점을 사용하여 민감한 정보를 훔치거나 심각한 공격을 감행한다.

애플리케이션은 공격자에게 정보가 유출되는 것을 막기 위해 다음과 같은 사항을 유의해야 한다.

- 자세한 오류 처리를 제한하거나 기능을 비활성화한다. 특히, 최종 사용자에게 스택 추적 정보, 경로 정보와 디버그 정보를 보여주지 않는다.
- 비슷한 시간대에 산출되는 여러 결과에 대한 에러 메시지는 비슷하거나 동일하게 보여준다.
- 데이터베이스와 IIS, Apache 등과 같은 웹서버들은 치명적이거나 예외적 결과를 반환할 수 있는데 이러한 것들이 발생시키는 에러에 대해 적절히 점검 및 설정해야 한다.
- 사용자에게 보여지는 에러 메시지를 적절히 걸러내 반환하는 기본적인 에러 처리기를 만든다.

3.7 취약한 인증 및 세션 관리

자격 증명 및 세션 토큰 등이 적절히 보호되지 못함으로 인해 공격자는 비밀번호, 키, 혹은 인증 토큰을 손상시켜 사용자나 관리자 계정을 가로챌 수 있다. 이는 권한 및 계정 통제를 약화시키고 개인정보 침해를 유발한다.

인증 취약점에 대응하기 위해 고려해야 사항은 다음과 같다.

- 고유의 세션 관리 메커니즘을 사용한다. 보조적인 세션 처리기는 사용하지 말아야 한다.
- URL 또는 요청 내에서 새로운, 사전 설정된, 또는 유효하지 않은 세션 식별자는 허용하지 않는다.
- 커스텀 쿠키의 사용을 제한하거나, 사용하지 않는다.
- 적절한 수와 강도의 요소를 갖는 단일 인증 메커니즘을 사용한다.
- 암호화 되지 않은 페이지에서 로그인 과정을 시작하지 않도록 한다.
- 성공적인 인증 또는 권한 수준 변경에 대해 새로운 세션을 재생성한다.
- 모든 페이지에 로그아웃 연결을 둔다.
- 보호되어야 할 데이터의 가치에 따라 비활동 세션을 자동으로 로그아웃시킬 수 있도록 타임아웃 기간을 설정한다.
- 질문과 답변, 암호 재설정과 같은 보조적인 인증 기능을 사용한다. 보안을 위해 암호를 복잡하게, 너무 자주 변경하는 것은 오히려 역효과를 초래할 수 있다[6]. 30일마다 변경되는 10문자 조합 이상의 암호를 갖는 것이 좋다. 암호 재설정을 위해 등록된 이메일 주소로 비밀 정보를 보내는 것을 주의해야 한다.
- 세션 식별자 또는 유효한 자격의 부분이 URL 또는 로그에 드러나지 않도록 한다.
- 사용자가 새로운 암호를 변경할 때 이전 암호를 확인한다.
- IP 주소, 주소 범위 마스크, DNS, DNS lookup, 참조 헤더와 같이 변조되기 쉬운 자격 증명만으로 인증하지 않는다.

3.8 불안전한 암호화 저장

암호화함으로써 민감한 데이터를 보호하는 것은 대부분 웹 애플리케이션의 중요한 역할이지만 암호화에 실패하는 것이 매우 보편적이다. 애플리케이션은 흔히 불완전하게 설계된 또는 부적절한 암호화 알고리즘을 사용해서 심각한 오류를 범하게 되는데 이러한 결함으로 인하여 민감한 데이터가 노출될 수 있다.

이러한 취약점을 극복하기 위해서는 첫째, AES, RSA 공개키 암호화, SHA-256과 같은 검증된 알고리즘을 사용하고 MD5 / SHA1 과 같은 취약한 알고리즘을 사용하지 않는다. 둘째, 오프라인 상에서 키를

생성하고 개인키 저장에 유의한다. 셋째, 데이터베이스 자격인증 또는 MQ queue 접근 내역과 같은 기반구조 자격인증을 적절히 보호한다. 넷째, 암호화된 데이터가 쉽게 복호화되지 않도록 한다.

3.9 불안전한 통신

애플리케이션은 민감한 통신을 보호해야 할 필요가 있을 때 네트웍 트래픽을 암호화 하는데 실패하곤 하지만, 모든 인증된 연결과 인터넷으로 접근 가능한 웹 페이지, 그리고 백엔드 연결은 암호화되어야 한다. 그렇지 않으면 애플리케이션은 인증 또는 세션 토큰을 노출시킬 수 있다.

이에 대한 가장 좋은 보호 방법은 모든 인증된 연결에 또는 민감한 데이터가 전송될 때마다 SSL을 사용하는 것이다. SSL을 구성하는 데는 많은 세부 항목들이 있으므로 먼저 사용될 환경을 분석하고 이해하도록 한다.

3.10 URL 접속 제한 실패

애플리케이션은 인증되지 않은 사용자에게 URL이나 link를 보여주지 않는 것으로 민감한 기능을 보호하고자 하나, 공격자는 이러한 URL에 직접 접근하여 권한이 부여되지 않은 동작을 수행할 수 있다.

애플리케이션의 역할과 기능을 매핑시키는 매트릭스를 생성시켜 접근 권한을 설계함으로써 제어되지 않는 URL 접근으로부터 애플리케이션을 보호한다.

4. 결론

웹 애플리케이션의 사용 범위와 중요성이 커짐에 따라 그에 대한 보호의 노력 역시 많이 요구되고 있다. 대부분의 조직에서는 취약점을 찾고 인증, 접근 제어, 입력값 증명, 에러 처리, 로깅, 암호화 등을 포함한 수많은 중요 보안 영역에 대해 코드를 검토하고 보안 테스트를 실시함으로써 코드가 제대로 설계되고 구현되었는지 입증하고자 한다. 하지만 이와 같은 노력에도 불구하고 애플리케이션이 갖고 있는 취약점은 프로젝트 내에 근본적인 원인이 있는데 개발자의 보안 훈련이 부족하거나, 시스템 개발 과정 전반에 걸쳐 애플리케이션 보안 활동이 미흡했거나, 기술이 부족한 것 등이다.

언제 어디서 어떻게 접근해 올 지 모르는 공격자로부터 중요 애플리케이션을 보호하기 위해서는 애플리케이션을 개발하는 전 단계에서 관련된 모든 사람들이 보안 지식과 기술을 충분히 습득하고 능동적이고 지속적인 대처를 해야 할 것이다.

참고문헌

[1] Aspect Security, "Starting Out With Application Security", http://www.aspectsecurity.com/owasp.htm, 2007.

[2] Norhazimah Abdul Malek, "SECURING APPLICATONS FROM HACKERS", Computimes, 28 November 2005.

[3] OWASP, "Top 10 2007", http://www.owasp.org/index.php/Top_10_2007, 2007.

[4] 시큐아이닷컴, "웹 어플리케이션 보안 및 대응", 부산 사이버안전 전략 세미나, 2006.

[5] 한국정보보호진흥원, "홈페이지 개발 보안 가이드", http://www.kisa.or.kr/index.jsp, 2005.

[6] OWASP, "Guide to Authentication", http://www.owasp.org/index.php/Guide_to_Authentication, 2007.