



# Mobile Application Security Testing

**Author:**

**Gursev Kalra**

Senior Security Consultant  
Foundstone Professional Services

## Table of Contents

Table of Contents.....	2
Introduction.....	3
Test Environment.....	3
Mobile Installable Applications .....	4
Application Footprint Analysis .....	4
Reverse Engineering .....	7
Reversing a Java MIDlet.....	7
Reversing a Symbian Application .....	9
Browser Based Mobile Applications.....	12
The User-Agent Request Header .....	12
The Accept Request Header .....	14
Intercepting Application Traffic with a Web Proxy .....	15
Configuring Proxy Settings for Nokia S40 Phones.....	16
Configuring Proxy Settings for Nokia S60 Phones.....	19
Web Proxy, Mobile Applications and HTTPS .....	19
Various Proxy Configurations .....	22
Proxy on a Static Public IP.....	23
Proxy on a WLAN.....	23
Proxy with One Phone.....	24
Proxy with an External Internet Connection.....	25
Conclusion.....	26
About the Author .....	27
Acknowledgements .....	27
About Foundstone Professional Services .....	27

## Introduction

The consumer mobile application market has seen explosive growth in the last couple of years. These applications have provided convenient access to bank accounts, credit card data, personally identifiable information (PII), travel itineraries and personal emails to name a few. The enterprise mobile applications extend corporate networks beyond the perimeter devices and thus potentially expose these organizations to new types of security threats.

Security risks associated with these applications can often be identified and mitigated by subjecting them to security testing. Compared to desktop or web applications, mobile applications are harder to test for security, and hence, in our experience, are often less tested. At the same time, these applications are not necessarily more secure than desktop or web applications.

We, therefore, document a methodology followed for testing mobile applications. This methodology is similar to that which experienced security testers would use for desktop or web applications, so we focus on the specific challenges that are unique to mobile application security testing. We will discuss the testing approach for mobile installable applications as well as those that are browser based. We discuss in detail the use of tools, such as proxies, to intercept and monitor all application traffic. While the examples discussed in this paper focus on Nokia related devices, the lessons learnt have been applied at Foundstone to a variety of mobile platforms with the tools being different.

## Test Environment

Before we begin discussing the details of the methodology, it is useful to define our test environment in use throughout this paper. The software and hardware in use include:

- Nokia PC Suite
- Nokia E61i (Nokia S60 Series device). All the operations were performed on this phone unless otherwise noted.
- Nokia 6212 (Nokia S40 Series device). Used to elaborate some scenarios.
- Home wireless router

## Mobile Installable Applications

These applications get installed to the phone file system. Similar to applications that are installed on desktops, a new application adds files, modifies registry entries and configuration settings, registers new services, and performs other such activities during installation. For a good penetration test, it is important that all these components be analyzed and tested. File system analysis and application reversing are two important aspects of performing client side analysis and they are discussed in this section.

## Application Footprint Analysis

For applications that get installed on the phone, application footprint analysis begins even before the installation. The file system fingerprint of installed applications must be closely monitored and analyzed. Developers often assume that the phone memory is a safe storage location and use it to store sensitive information like usernames, passwords and other PII. Phone file system analysis is a tedious process and should be done carefully. When analyzing the phone's file system, the main goals are:

1. Identify the files created on the phone by the application during installation. If the option is available, install the application to external storage such as a Compact Flash card. This is because some mobile phone models (like Nokia S40 and S60 series phones) do not allow access to all the contents of the phone's internal storage making our job of analyzing those files difficult. Once the files are identified, further analysis like reversing the application, modifying the application and extracting hidden secrets can be performed.
2. Identify changes made to existing files over multiple application operations.
3. Analyze the information written to the phone file system during various stages of operation.

We will follow a series of steps to analyze the phone file system which are discussed in detail below:

**Step 1 – Directory & File Selection:** Even though the phone file system is very small in size as compared to a desktop, the level of effort required to analyze it can be further reduced by careful selection of file-system components. The purpose of directory selection is to reduce the number of phone file system directories and hence, files, to be analyzed during testing. The following process can be followed to come up with the minimum list of directories to be monitored for your phone and application:

1. Generate a recursive directory listing of the phone file system and store it on a different computer for future reference.
2. Install the mobile application and create a second recursive directory listing<sup>2</sup> of the phone file system. Do not perform any other actions beyond application installation. Any configuration changes or application browsing may alter or remove files which, in turn, might affect the testing.
3. Compare both of the directory listings using diff tools like ExamDiff<sup>1</sup> for Windows or diff for \*NIX systems. This will provide a listing of the newly created files.
4. To be thorough, manually analyze the file system to identify the directories in which applications might install their files.
5. After careful consideration, develop a final list of files and directories which will be considered for analysis.

In our test environment, out of the 16 directories available by default on the root file-system of our test phone (Nokia E61i), only 6 were chosen for active monitoring. All the newly created files were copied to a desktop for further analysis.

**Step 2 – Fingerprinting:** In step 1, our main concern was to identify the files and directories for active monitoring. Now we shift our focus to concentrate on what is inside the files and when the contents get changed. The easiest way to keep track of file changes is to create content hashes (MD5, SHA1, SHA 256 etc...) and compare them amongst various application runs. Use tools like md5deep<sup>2</sup> (or sha1deep or sha256deep) to generate a MD5 (or SHA1 or SHA256) hash of all the files in the directories of interest. Any change made to contents of an existing file will change the file hash which can be detected when performing comparisons of file hashes between various runs.

A command to create recursive file hashes of the relevant directories on the phone file system would look something like:

---

<sup>1</sup> [http://www.prestosoft.com/edp\\_examdiff.asp](http://www.prestosoft.com/edp_examdiff.asp)

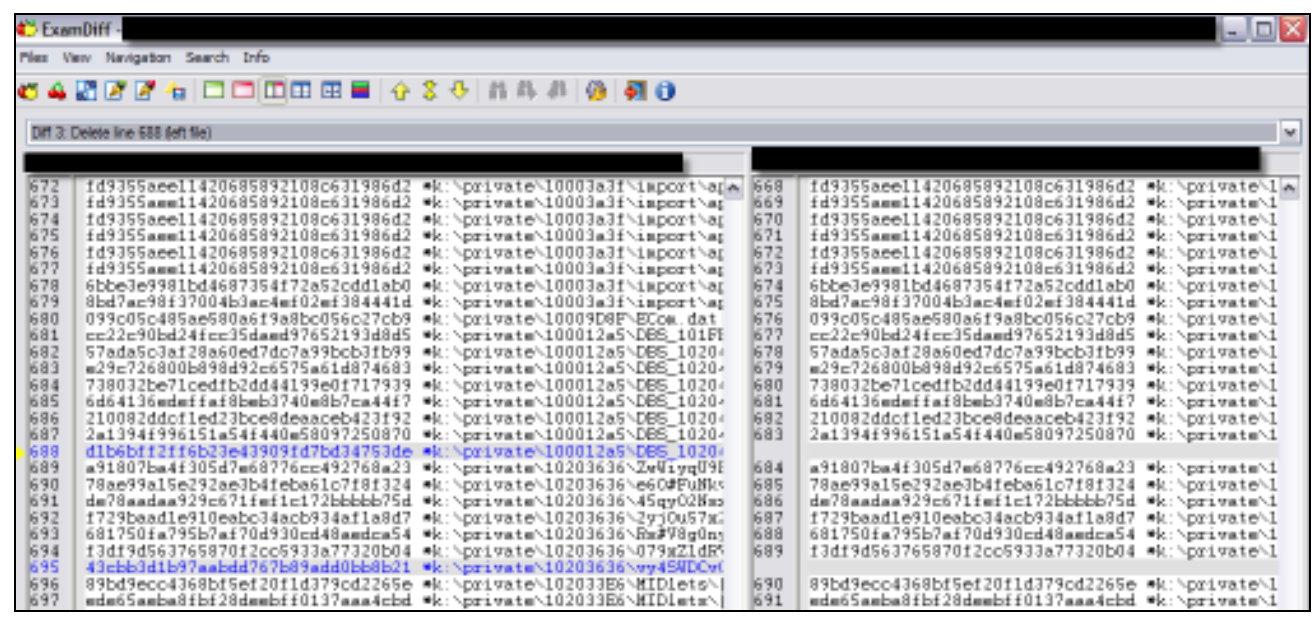
<sup>2</sup> md5deep belongs to a set of open source hashing utilities which can be obtained from <http://md5deep.sourceforge.net/>. Other hash utilities available with this suite are hashdeep, sha1deep, sha256deep, tigerdeep and whirlpooldeep. The distributions supported are Linux, Solaris, Mac OS X, BSD and HP/UX. Other hashing tools are also available.

```
md5deep -k -r k:\sys k:\system k:\shared k:\resource k:\private k:\data >
First_FingerPrint.txt
```

This command recursively explores the directories and calculates the MD5 hash of all the files encountered. The output generated is written to the file First\_FingerPrint.txt. Now we have a post installation hash of all files from our directories of interest.

**Step 3 – Configuration and usage:** Launch and configure the application and exercise the various use cases supported by the application. For example, add a credit card to a mobile wallet application, or read emails if the software is a mail client.

**Step 4 – Fingerprinting and comparison:** Create a new hash fingerprint of all the directories of interest and compare the results with the results from step 2. Make a copy of files that are new and / or have their content changed. Figure 1 shows the sample diff of files generated in Step 4 and Step 2.



**Figure 1: Difference between two fingerprint files. Two additional files (highlighted in blue) on the left were created by the application after it was run.**

**Step 5 – Content analysis:** The files copied in step 1 and 4 should be analyzed to look for sensitive information or other important data that the application might store in the mobile memory.

During Foundstone's testing of mobile applications, we have found instances of unencrypted credit card numbers, username and passwords for email accounts. Testers can create appropriate regular expressions to aid in their searches.

**NOTE:** *For Nokia phones, remember to run the phone in "Data Transfer Mode" while generating file hashes. Hashes of the files on phone memory can be easily calculated after mounting the phone memory as a data device on your computer.*

## Reverse Engineering

We have thus far successfully installed, configured and thoroughly exercised the application under test. A thorough file system analysis is also completed. We thus know about all the files that have been installed on the phone, the changes made to these files after various actions performed and contents of these files. Armed with this knowledge, we enter the reversing phase. Application reverse engineering is performed with the following objectives:

1. Attempt to uncover application logic and code. This will help us modify the application code if needed and thus overcome any security measures the application might have. Trusting the client environment is a common occurrence in many mobile applications and hence, if we can find some way to bypass client security mechanisms, it often provides us with privileged access to the server and beyond.
2. Identify implementation and / or design flaws and find possible methods to exploit them.
3. Look for hidden secrets like passwords, encryption keys and other sensitive data in application code.

The two most common types of applications available for Nokia phones are Java MIDlets and Symbian applications. An introduction to reversing each of these follows.

### ***Reversing a Java MIDlet***

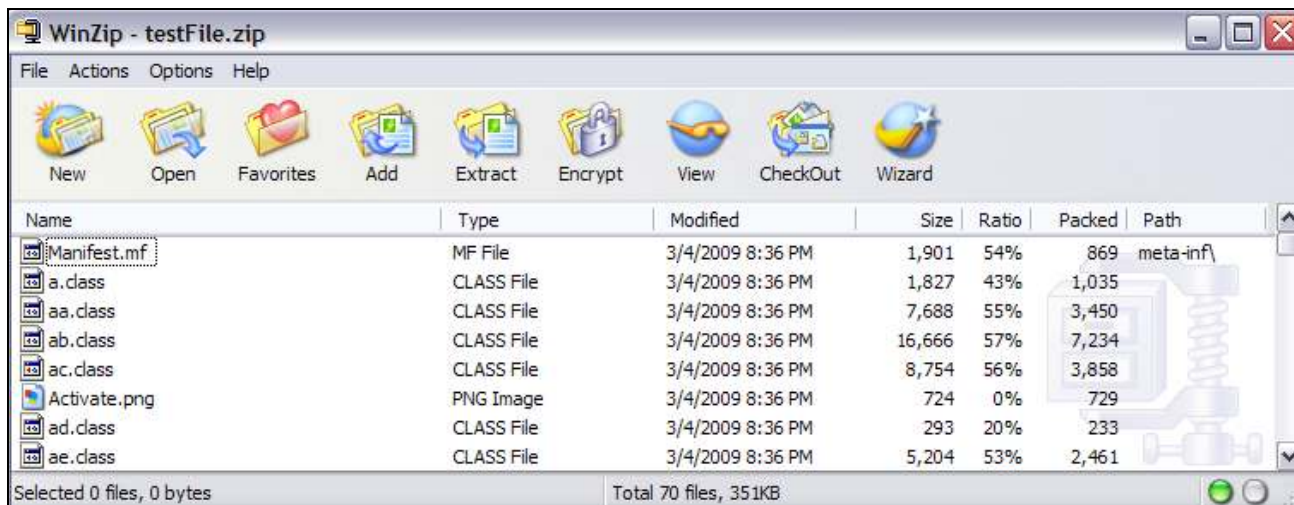
The Java Application Descriptor (JAD) file and the Java Archive (JAR) file together are the two components of a MIDlet suite. The JAD file describes the MIDlet suite and includes its name, the location and size of the JAR

file, and the configuration and profile requirements. The file may also contain other attributes, defined by the Mobile Information Device Profile (MIDP)<sup>3</sup>, the developer, or both.

The purpose of reversing is to extract information from JAR files, decompile them and then dissect the code as per the test requirements. If required, the mobile application can be modified or even re-written during the reversing phase to aid in testing.

**Extracting contents from a JAR:** The JAR file format is built on the ZIP file format. Therefore, it is possible to create or extract JAR files using a `jar` command that is available with the JDK. The command `jar -xf testFile.jar` extracts all files from the JAR file.

Popular ZIP programs like WinZip, WinRAR and gunzip can also be used to extract contents of a JAR file. To use WinZip, change the extension of the `.jar` file to `.zip` and extract the contents like a ZIP archive. Figure 2 below shows the contents of a `.jar` file opened within WinZip.



**Figure 2: A JAR file opened by WinZip**

**Decompiling class files:** After extracting all / selected files from the application JAR, select the class files on which further analysis is to be performed. Class files represent Java byte code and can be easily decompiled using a tool such as JAD<sup>4</sup>.

<sup>3</sup> <http://java.sun.com/products/midp/>

**Source Sifting for hidden secrets:** Scan the decompiled code and extracted contents for hidden secrets like encryption keys, hardcoded username/password or any other sensitive information from source code.

**Source code analysis, modification and re-compilation:** Investigate the decompiled code to identify application functionality of interest. Per your requirement, modify or delete the obtrusive code sections and recompile the jar file.

Decompiling, code analysis, modification and re-compilation become difficult if the application code was obfuscated during the compilation process. For more details on decompiling, recompiling and signing of client side Java code, refer to Foundstone's detailed paper on this topic available on our website<sup>5</sup>.

### ***Reversing a Symbian Application***

Symbian applications are compiled to .SIS and .SISX files. Symbian applications can be installed on Nokia devices using Nokia PC Suite or by copying the file to phone memory and executing it from there. Executing .SIS and .SISX files extracts their contents and installs the corresponding application on the supported Symbian device. These applications run natively on the Symbian Operating System.

SISWare and SISXplorer are two freeware tools which will aid you in Symbian application reversing.

### **SISWare**

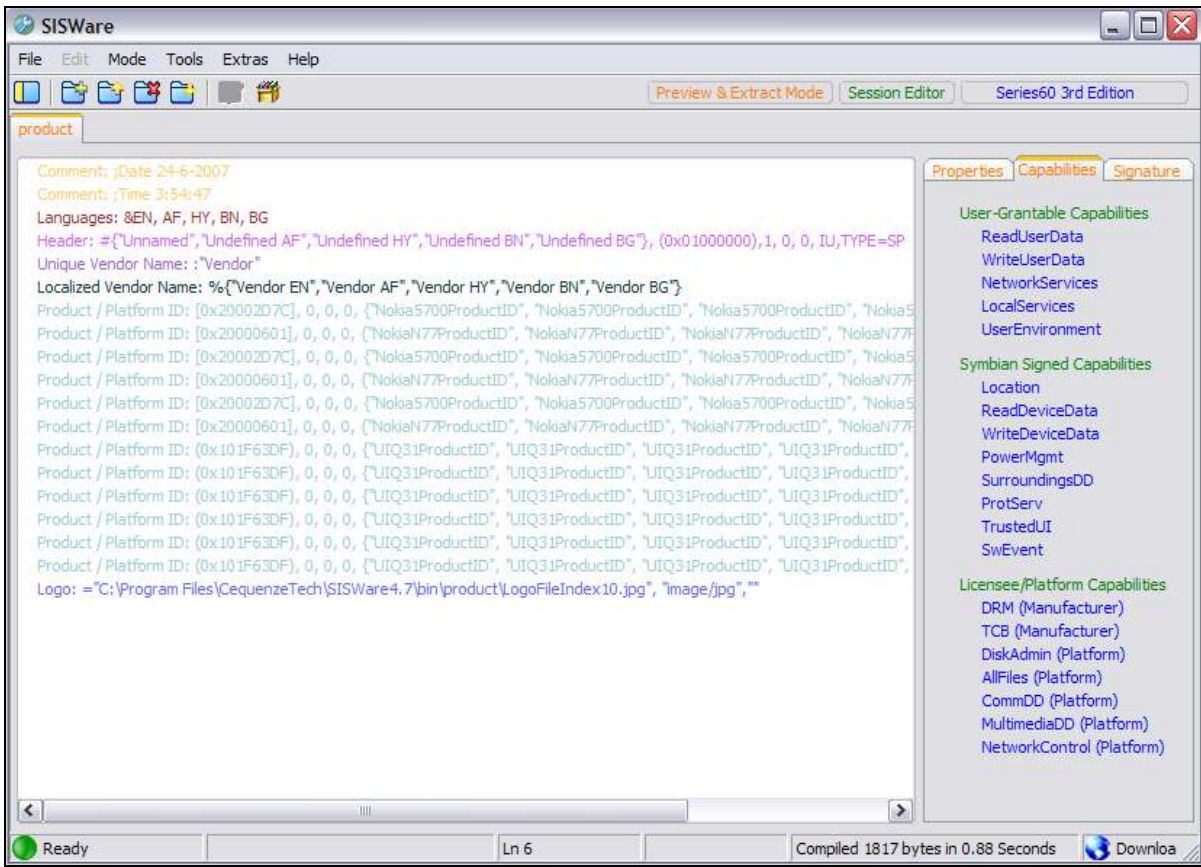
SISWare<sup>6</sup> is a Symbian Application Install package builder.

---

<sup>4</sup> JAD is a decompiler for the Java programming language. JAD provides a command line user interface to extract Java source code from class files. More information is available at [http://en.wikipedia.org/wiki/JAD\\_\(Java\\_Decompiler\)](http://en.wikipedia.org/wiki/JAD_(Java_Decompiler)).

<sup>5</sup> [http://www.foundstone.com/us/resources/whitepapers/java\\_client\\_side.pdf](http://www.foundstone.com/us/resources/whitepapers/java_client_side.pdf)

<sup>6</sup> <http://www.cequenzetech.com/products/mobile/sisware>



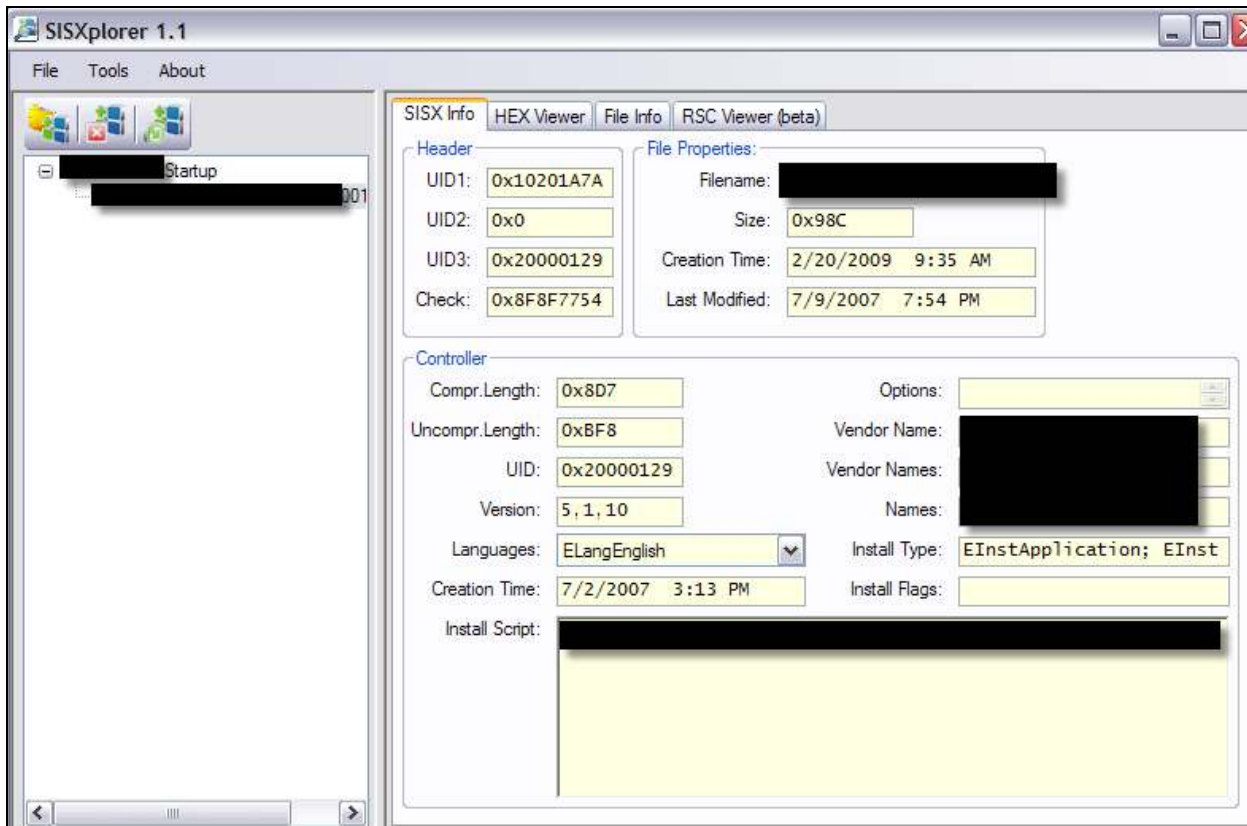
**Figure 3: SISWare**

SISWare supports SISX and SIS file creation and extraction for Symbian9.x as well as previous versions. The current version of SISWare is 4.8 and is what we describe and use below. Key features of this useful utility include:

- Open and extract files from a SIS package.
- Edit, compile and re-build a SIS file.
- Batch extraction of multiple SIS files.
- Sign and Un-sign SIS files.
- Create self-signed certificate file.

A sample screenshot of SISWare is shown in Figure 3.

## SISXplorer



**Figure 4: SISXplorer Showing the Contents of a Symbian Application**

SISXplorer<sup>7</sup> is a read only utility which allows you to inspect and extract all the content from Symbian 3rd edition installation packages. A summary of the functions supported by SISXplorer are:

- SIS Viewer / Extractor - View and extract the files from SIS archives
- TXT Viewer - View text files
- E32 Viewer - View detailed header information
- RSC Viewer - View resources

<sup>7</sup> <http://www.symbian-toys.com/sisexplorer.aspx>

- MIF Viewer - View and extract SVG icons
- Integrated MBM Manager - View and extract images and icons from themes
- Integrated Hexadecimal Viewer – Inspect file content
- Quick Open / Extract - Quickly extract files
- Multiple Archives View - Simultaneously work with multiple SIS archives
- Information Viewer – View detailed information about each archive and contained files

A screenshot of this tool is shown in Figure 4.

## Browser Based Mobile Applications

A number of mobile applications we encounter are browser based and do not require any installable components. Often, attempts to access these applications using common desktop browsers either redirects users to the regular application (as opposed to the mobile version) or returns an error page. Web servers analyze the request headers to identify the device type and the browser to make their decisions on content rendering.

Two request headers<sup>8</sup> often used to make content rendering decision are the `User-Agent` and The `Accept request header`.

### ***The User-Agent Request Header***

All web browsers include a `User-Agent` header in their requests. This header is used to identify the web browser (Internet Explorer, Firefox, Opera for instance) and also the device on which the browser is running. Almost all the major mobile phone browsers include the device type in the `User-Agent` header. A number of web servers / applications, in turn, depend on this request header to render content and provide functionality. For example, when <http://m.google.com> is accessed from a computer using Internet Explorer or Firefox, the web server redirects the browser to <http://www.google.com/mobile/>. When the same URL (<http://m.google.com>) is accessed from a mobile phone browser, the user is presented with a page where they can download a variety of Google mobile applications. As you would expect, changing (as we will

---

<sup>8</sup> <http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html>

illustrate below) the `User-Agent` header on your desktop to that of a mobile device supported by the web server, causes the web pages returned to be those intended for that device. This behavior can be utilized to benefit testing and we, therefore, discuss active `User-Agent` manipulation using a number of techniques.

**Using a Web Proxy:** Most web proxies (like Paros<sup>9</sup> or Fiddler<sup>10</sup>) provide the user with an option to modify request headers using either regular expressions or through direct substitutions. In most cases, this is a one-time configuration change and requires you to know the `User-Agent` header for the device being tested. For example, the Nokia E61i phone `User-Agent` header is:

```
User-Agent: NokiaE61i-1/3.0 (1.0633.22z.05) SymbianOS/9.1 Series60/3.0  
Profile/MIDP-2.0 Configuration/CLDC-1.1
```

Configuring the web proxy to replace the existing `User-Agent` header to the one above may allow you to potentially access web applications supporting E61i phones from your computer.

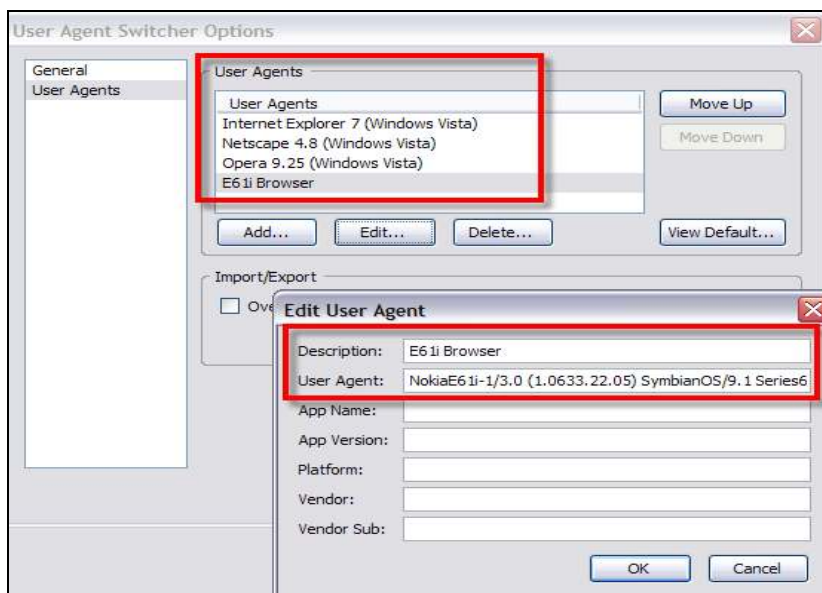
**Firefox User Agent Switcher add-on<sup>11</sup>:** This convenient add-on allows you to create and store multiple arbitrary user agent headers as can be seen in Figures 5 and 6. A mobile user agent profile can be activated to browse the application just as you do from your mobile phone without the need for a web proxy to change the headers.

---

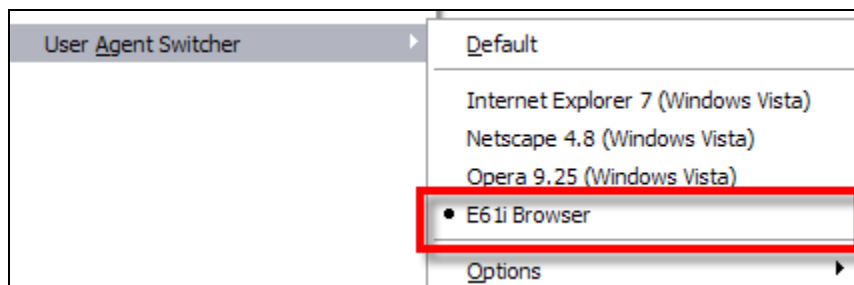
<sup>9</sup> <http://www.parosproxy.org/>

<sup>10</sup> <http://www.fiddlertool.com/>

<sup>11</sup> <https://addons.mozilla.org/en-US/firefox/addon/59>



**Figure 5: Configuration of User Agent Switcher Firefox Add-on.**



**Figure 6: User Agent Switcher activated to change the User-Agent header in Firefox to that of the E61i**

### **The Accept Request Header**

All web browsers include an `Accept` header in their requests. This header is used to notify the server about the type of data the browser can accept. For instance, web servers utilize the `Accept` header to determine whether the browser can read Wireless Markup Language (WML) pages and thus return the appropriate content. The `WMLbrowser` add-on<sup>12</sup> for Firefox adds support to view WML pages and modifies the `Accept` header to reflect the same. The advantage of using a tool such as this is that once your desktop browser is configured to browse through the application, all testing can be performed as a regular web application penetration test.

<sup>12</sup> <https://addons.mozilla.org/en-US/firefox/addon/62>

This, in our experience works in most cases. However, every once in a while we run into situations where despite changing the headers described above, the server will not treat your desktop browser as if it were a mobile device. In this case, an external web proxy can be set-up for the mobile device and all the application traffic can be routed through it. We discuss this next but, it is important to note that browsing the application from your mobile phone does have the drawbacks of a lack of speed with which testing can be performed as well as the tools available for use.

### **Intercepting Application Traffic with a Web Proxy**

Until this point, while we have successfully watched the application's device footprint and perhaps accessed the application from a desktop browser, we know little about the type of requests being sent to the server or the resources that were being accessed there. Even if we are successful in application reversing / creating a new client, it helps to route our application traffic through a web proxy so that we can intercept it, view application behavior in detail and modify data for data validation, authorization and other areas of testing. Intercepting application traffic gives us complete control over the client server interaction and this, in turn, helps us perform a thorough test. Further, after successfully routing application traffic through a web proxy, all testing can be performed in a similar fashion as a regular web application using familiar tools and techniques. The examples discussed below were tested with Nokia devices on GPRS data networks. However, mobile phones over 3G networks can also be configured in a similar fashion.

**The Basics:** In GPRS networks, data communications of a mobile phone are routed through "GPRS Access Points". To initiate an Internet connection from the device, it is first necessary to point it to the correct Access Point Name (APN). Telecom operators specify different APNs for different types of services such as web browsing, email and instant messaging. Each APN, as you would expect, has an associated IP Gateway / Proxy which is used to route all traffic. As testers, if we are able to change the IP Gateway / Proxy of an Access Point, we can route all application traffic through our own proxy. The technique for changing this configuration varies from device to device and, on some, can be fairly complicated. Through the rest of this paper, we will configure proxy settings for Nokia S40 and S60 phones.

### ***Configuring Proxy Settings for Nokia S40 Phones***

The Nokia S40 phone is an example of a device where changing proxy settings can be non-trivial. Network access settings for S40 devices are specified in a provisioning document and there is no functionality provided by the phone operating system itself to edit the proxy settings for an access point. However, the provisioning document can be created on a PC, and then transferred to the phone. In order to do this we need to understand the basics of two languages commonly used to define such configuration: WML and WBXML.

**WML** is based on XML and is a markup language intended for devices that implement the Wireless Application Protocol (WAP). In the context of this paper we are primarily concerned with the fact that mobile service providers typically use WML to represent their Access Point Names and corresponding settings.

**WBXML** (WAP Binary XML) is a binary representation of WML. It was developed by the Open Mobile Alliance as a standard to allow XML documents to be transmitted in a compact manner over mobile networks. The provisioning document for Nokia S40 phones is in the WBXML file format. Given its binary format, it helps to use a WML to WBXML converter to create a provisioning document. One such converter is the open source WBXML Library<sup>13</sup>. For our purpose, we will use the `xml2wbxml` executable within this library to create the provisioning document. The steps to do this for the Nokia S40 are detailed below:

1. The WML file containing the GPRS settings for the mobile service provider can be obtained by contacting the service provider and / or searching for it on the Internet. While using configuration settings from the Internet sources, make sure to confirm the same with the service provider before installing them on the phone.
2. Edit the WML file containing the Access Point settings of the mobile service providers. In the WML file, change the proxy settings to point to your own proxy IP and port. This proxy will be utilized to monitor, intercept and modify the traffic generated by the mobile application. See the sample below:
3. Compile the updated WML file to create a WBXML provisioning document with the following command:

```
xml2wbxml.exe -o Modile_Service_Provider.prov Modile_Service_Provider.wml
```

Make sure that the output provisioning document ends with the `.prov` extension.

---

<sup>13</sup> <http://sourceforge.net/projects/wbxml/lib/>

4. Push the provisioning document to the test phone using Bluetooth. On receiving the provisioning document, the phone will recognize the configuration settings and will issue a prompt to save them. It is important to not just copy the provisioning document to the phone file system since the provisioning documents are recognized only when they are deployed Over the Air (OTA).
5. Next, browse to "Menu → Settings → Configuration → Default config. sett." and set the configuration to the compiled configuration.
6. Select "Act. def. in all apps." to ensure that the new custom configuration is utilized for all communications.

**Example WML: XYZ-Mobile.wml**

```
<?xml version="1.0"?>
<!DOCTYPE wap-provisioningdoc PUBLIC "-//WAPFORUM//DTD PROV 1.0//EN"
"http://www.wapforum.org/DTD/prov.dtd">
<wap-provisioningdoc version="1.0">

  <characteristic type="BOOTSTRAP">
    <!-- Name of the configuration -->
    <parm name="NAME" value="XYZ-MobileWeb AP"/>
  </characteristic>

  <!-- Defines how network access occurs -->
  <characteristic type="ACCESS">
    <parm name="RULE" value="Default Rule"/>

    <!-- Connect through specified proxy -->
    <parm name="TO-PROXY" value="XYZWebProxy"/>
  </characteristic>

  <!-- Defines proxy parameters -->
  <characteristic type="PXLOGICAL">
    <parm name="PROXY-ID" value="XYZWebProxy"/>
    <parm name="NAME" value="XYZWeb Proxy"/>
    <characteristic type="PXPHYSICAL">
      <parm name="PHYSICAL-PROXY-ID" value="XYZWeb Proxy"/>
      <parm name="PXADDR" value="192.168.30.102"/>
      <parm name="PXADDRTYPE" value="IPV4"/>
      <parm name="PUSHENABLED" value="0"/>
      <characteristic type="PORT">
        <parm name="PORTNBR" value="8888"/>
      </characteristic>
    </characteristic>

    <!-- Connect through specified access point -->
    <parm name="TO-NAPID" value="XYZWebNAP"/>
  </characteristic>
</wap-provisioningdoc>
```

```
</characteristic>
</characteristic>

<!-- Defines Network Access Point (NAP) parameters -->
<characteristic type="NAPDEF">
  <parm name="NAPID" value="XYZWebNAP"/>
  <parm name="NAME" value="XYZWeb NAP"/>
  <parm name="BEARER" value="GSM-GPRS"/>
  <parm name="NAP-ADDRESS" value="wap.XYZ.com"/>
  <parm name="NAP-ADDRTYPE" value="APN"/>
  <characteristic type="NAPAUTHINFO">
    <!-- PAP is "normal" authentication -->
    <parm name="AUTHTYPE" value="PAP"/>
  </characteristic>
</characteristic>

</wap-provisioningdoc>
```

**Example: Sample xml2wbxml run to compile a WML file to WBXML**

```
C:\Tools>xml2wbxml.exe -n -o XYZ-Mobile prov XYZ-Mobile.wml
Opened logfile `xml2wbxml.log'

*****
  Converting file: XYZ-Mobile.wml
Element: <wap-provisioningdoc>
Attribute: version = 1.0
End Attributes
...
Element: <parm>
Attribute: name = NAME
Attribute: value = XYZWeb Proxy
End Attributes
...
Element: <parm>
Attribute: name = PXADDR
Attribute: value = 192.168.30.102
End Attributes
Element: <parm>
Attribute: name = PXADDRTYPE
Attribute: value = IPV4
End Attributes
...
Element: <characteristic>
Attribute: type = PORT
End Attributes
Element: <parm>
Attribute: name = PORTNBR
```

```
Attribute: value = 8888
```

```
End Attributes
```

```
End Element
```

```
...
```

```
xml2wbxml succeeded
```

```
...
```

```
C:\Tools>
```

### ***Configuring Proxy Settings for Nokia S60 Phones***

In comparison to the steps described above, changing proxy settings for Nokia S60 smart phones is easy and can be quickly accomplished. Given this, if you have a choice of devices to use for testing, always go with the smart phone. The steps on a Nokia S60 E61i phone:

1. Browse to Menu → Settings → Connection → Access points.
2. Go to Options → New access point → Use existing settings.
  - a. To use an access point of the service provider, select one of the configuration settings of the service provider and change the connection name to Custom Proxy.
  - b. If the test phone is WLAN enabled, choose a WLAN access point for new settings and change the connection name to Custom WLAN Proxy. This will enable you to host a web proxy on your local network.
3. In either case, open the newly created access point for editing; go to Advanced Settings and change the proxy server IP and port to point to your own web proxy.

### **Web Proxy, Mobile Applications and HTTPS**

It is important to note that everything discussed above would work well if the mobile application being tested does not utilize HTTPS. In the real world, however, this is often not the case. Mobile applications will generally rely on the device's certificate store to determine trusted certificate authorities. If the signing certificate is not found in this trusted certificate store, communication is not established.

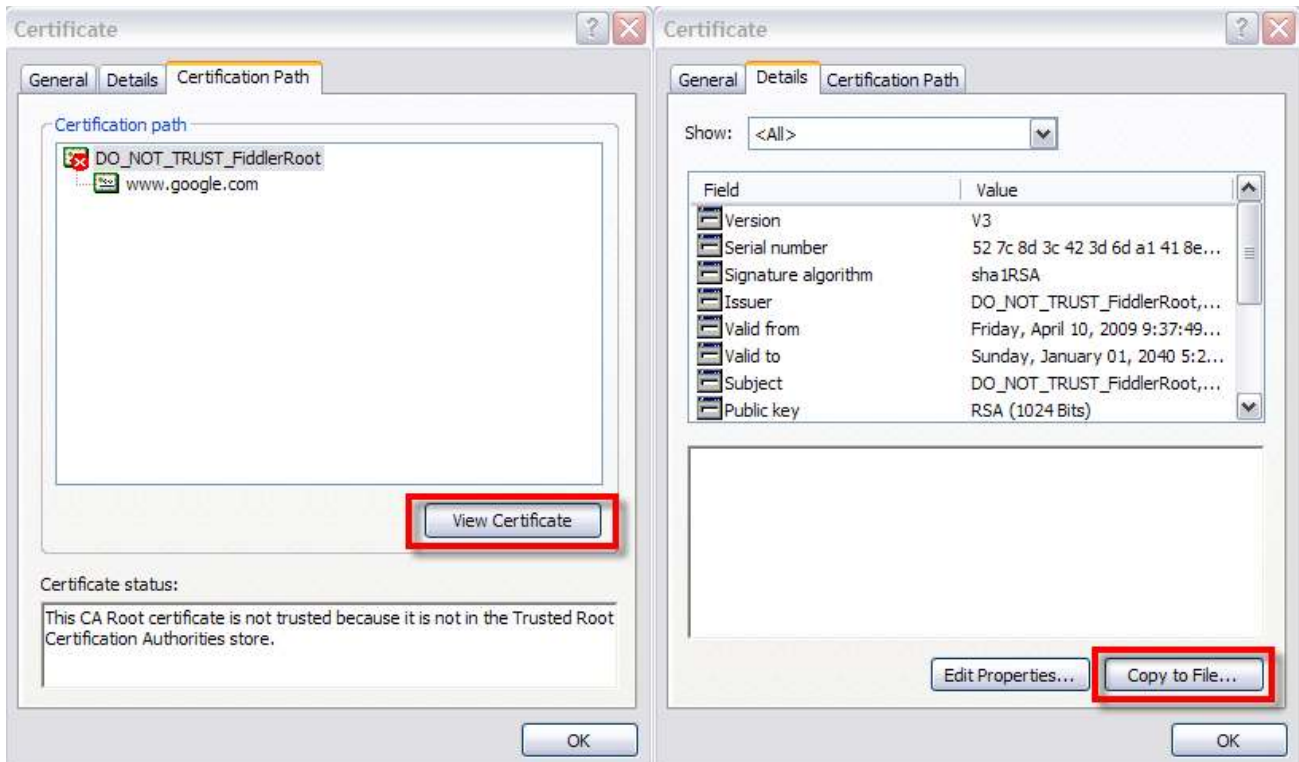
Fiddler is an example of a proxy which has its own signing certificate, which, as you would expect, is not part of the device's trusted store. With a little bit of tweaking however, it is possible to use such proxies for scenarios where HTTPS is involved. To use these proxies for traffic interception, we must first import their

signing certificates into the device's trusted certificate store. One of the possible ways to achieve this on Nokia S40 and S60 devices is discussed below:

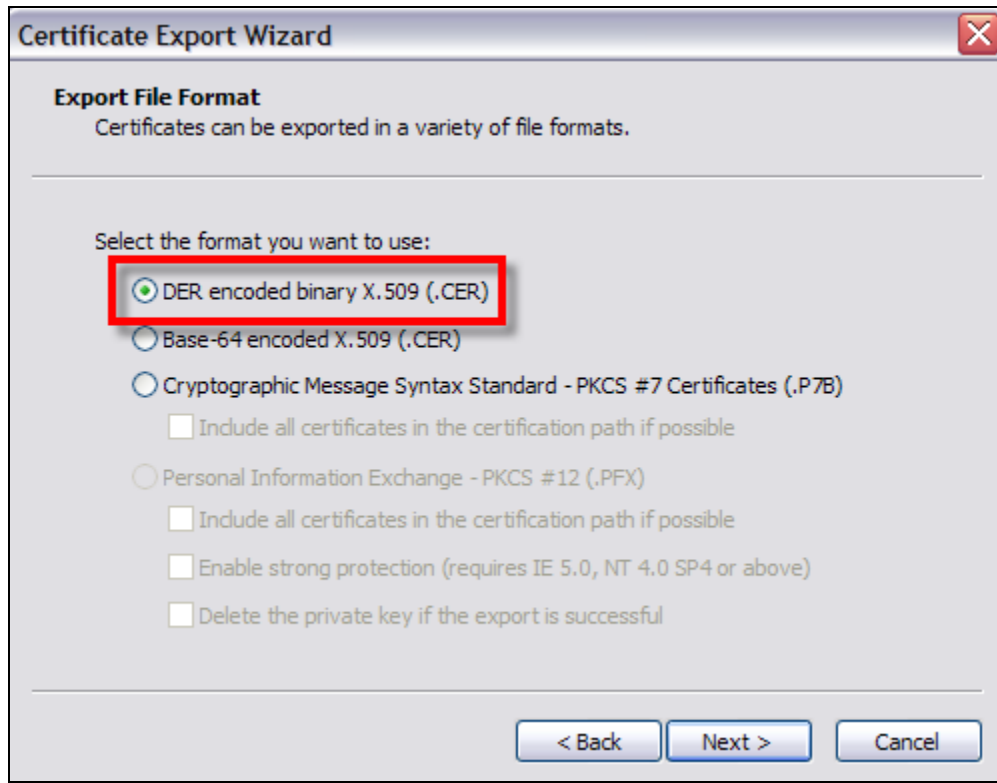
1. From your PC, extract the web proxy's signing certificate in DER format. This can be easily achieved in Internet Explorer using the following steps:
  - a. Configure Internet Explorer to route traffic through the web proxy of your choice.
  - b. Visit any page website that requires SSL.
  - c. Ignore the browser certificate error and continue browsing to the main page of the application.
  - d. Choose the "View Certificate" option of Internet Explorer to view the web proxy's signing certificate (Figure 7 shows the options to be used in Internet Explorer).
  - e. Copy the web proxy's signing certificate to a file in DER format (Figure 8 shows certificate export in DER format).
2. Copy the certificate file obtained above to a Web server
3. Set the MIME type of the directory to which the certificate is copied to `application/x-x509-cert` (consult your web server documentation for information on how to achieve this)
4. Use the web browser in the S40/S60 device to browse to the certificate file
5. Import the certificate

Once these steps or similar ones for the test device have been completed, you will be well on your way to test SSL enabled mobile applications.

**IMPORTANT:** Make sure that the certificate imported above is deleted or untrusted soon after your testing is complete. Failing to do so may make your phone vulnerable to "Man in the Middle" attacks in the future.



**Figure 7: Viewing a Web Proxy's signing certificate**



**Figure 8: Exporting Web Proxy's signing certificate in DER format**

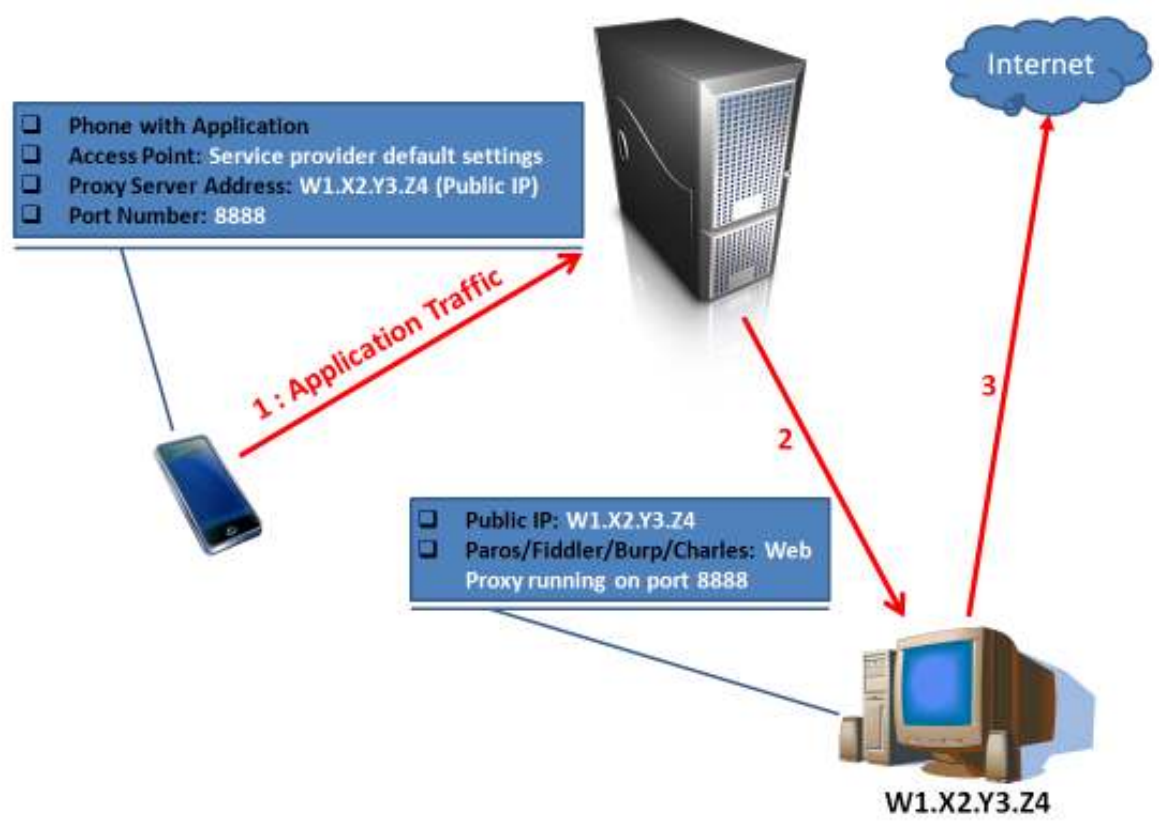
**Various Proxy Configurations**

Depending on the availability of hardware, various proxy configurations can be utilized to route, intercept and modify application traffic. This section summarizes some of the proxy configurations which you might find useful for your testing:

- 1. Proxy over a Static Public IP
- 2. Proxy over WLAN
- 3. Proxy with One Phone
- 4. Proxy with an External Internet Connection

**Proxy on a Static Public IP**

If you have a static public IP at your disposal for testing, this is the configuration for you. Configure your firewall so that it allows traffic to reach the port on which the web proxy is listening (8888, in this case). This setup is easy to configure on mobile phones. For Nokia S40 phones, a single provisioning document can be deployed on any number of phones when required. For Nokia S60 phones, one custom configuration can be used throughout the testing.

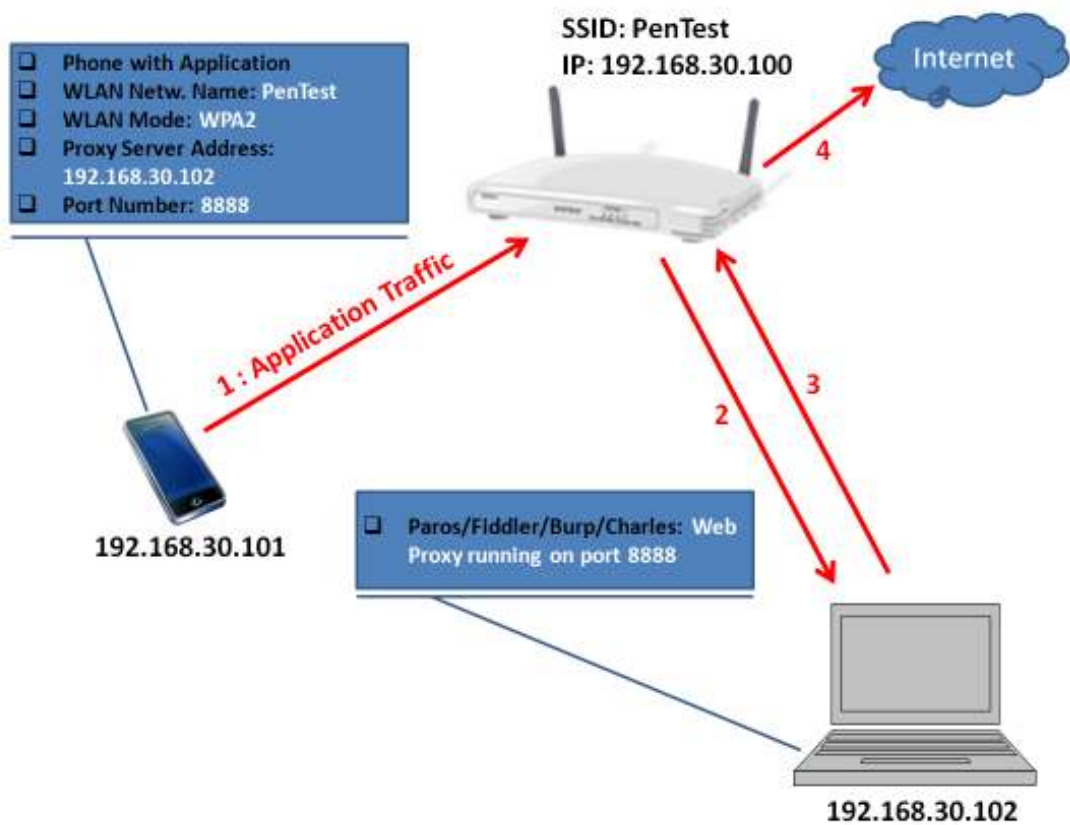


**Figure 9: Outgoing traffic flow when a public IP is used to host the web proxy**

**Proxy on a WLAN**

If you do not have a static public IP but the test phone supports wireless LAN, it is just as simple to setup a proxy on your internal network. Assuming the WLAN network utilizes static IPs or DHCP with a long lease

period, only one provisioning document will be necessary to complete multiple tests for Nokia S40 phones. Similarly for S60 phones, one custom configuration can be used throughout the testing.

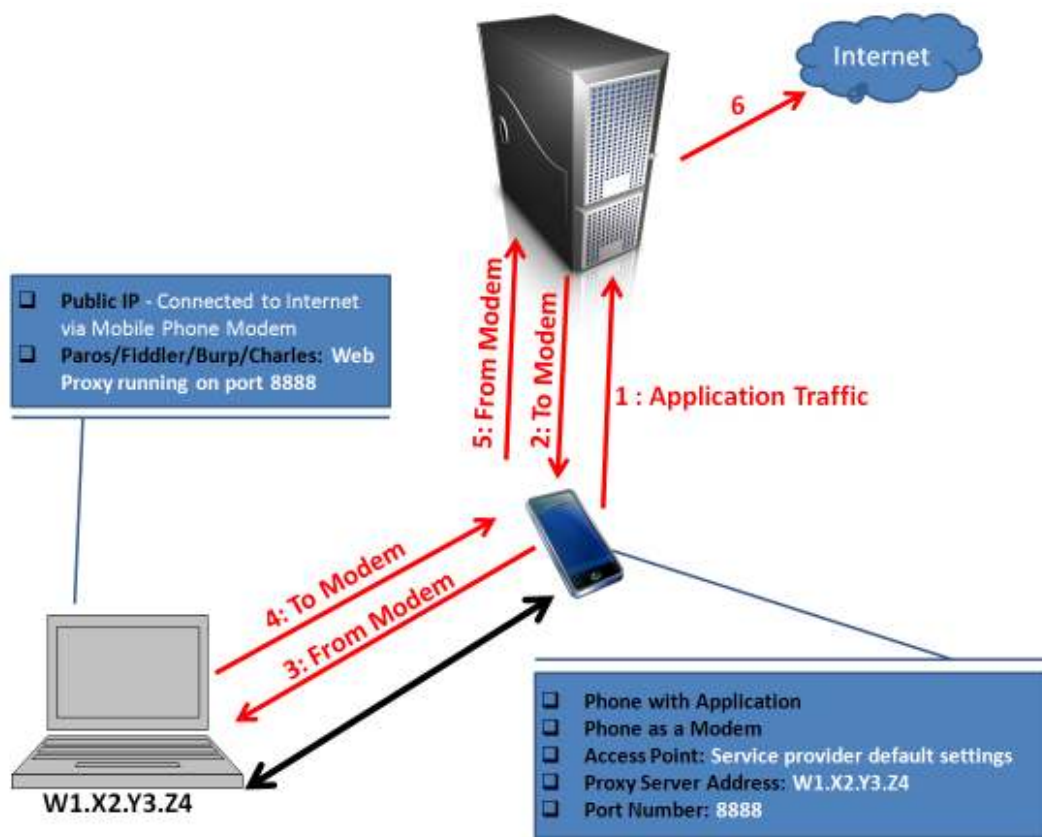


**Figure 10: Outgoing traffic flow when a web proxy is hosted in a WLAN environment**

### **Proxy with One Phone**

If neither of the two options discussed above are available, testing can still be performed. This configuration is also useful when the server for the application being tested resides inside the premises of the service provider and the application hosting server does not accept any external traffic. This setup assumes that the test phone has a modem which can be used to connect your computer (which is hosting a web proxy) to the Internet. This is otherwise known as tethering. Once your computer is connected to the Internet, obtain its

IP address<sup>14</sup>. This address must then be configured as the address for the device web proxy. The procedure for changing the proxy is described above but, it is important to note that you might have to do this multiple times if the IP address assigned to your computer changes.

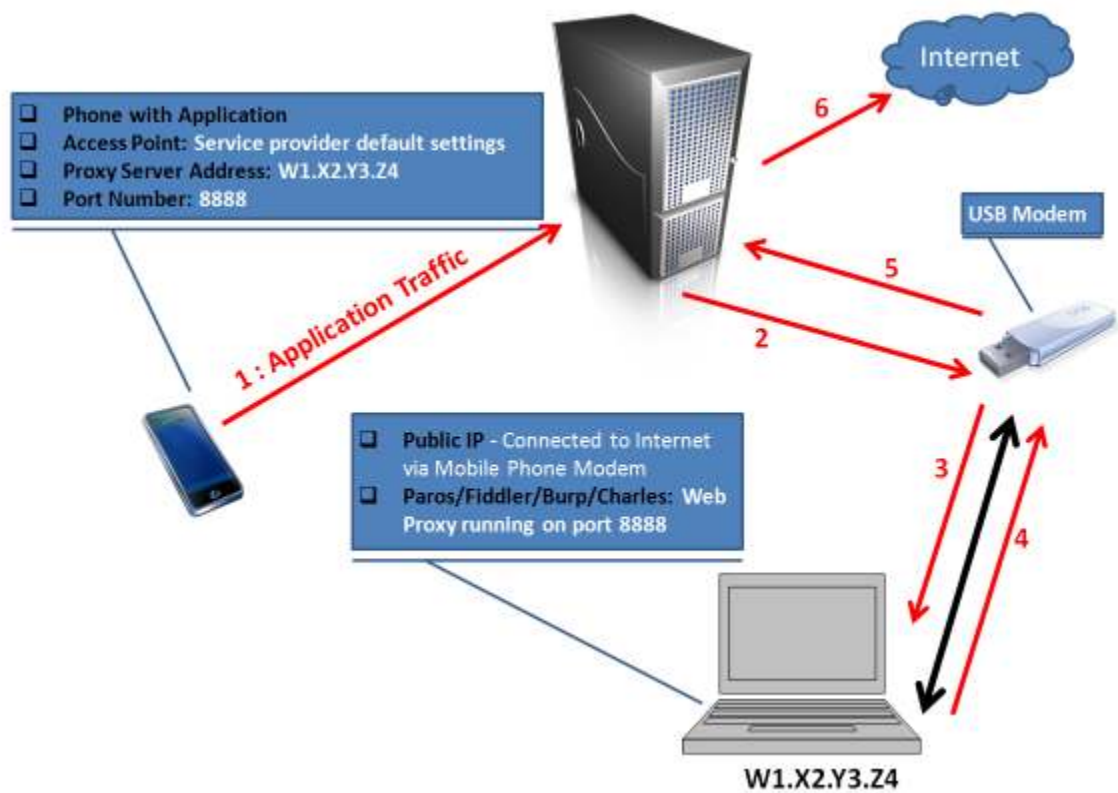


**Figure 11: Outgoing traffic flow when testing phone is used as a modem**

### **Proxy with an External Internet Connection**

A final configuration that may apply in some scenarios assumes that you have a data card or a USB modem or another phone which can be used to connect your web proxy hosting computer to the Internet. Once your computer is connected to the Internet, obtain its external IP address as before. This address must then be configured as the address for the device web proxy.

<sup>14</sup> A service such as [www.whatismyip.com](http://www.whatismyip.com) maybe used for this purpose.



**Figure 12: Outgoing traffic flow when a USB modem is used to connect web proxy hosting computer to the Internet**

### Conclusion

Mobile applications have already made inroads into large enterprises and, with time, we will start seeing more mission critical applications being available for mobile devices. This whitepaper is intended to help the reader understand how they can leverage the tools and techniques they are already familiar with and apply those to mobile applications.

## About the Author

Gursev Kalra serves as a Senior Security Consultant at Foundstone Professional Services, A Division of McAfee. Gursev focuses on web application penetration testing, external assessments and mobile application security testing. Gursev has also developed internal tools for internal / external network assessments. At Foundstone, Gursev has had the opportunity to work with some of the biggest financial institutions, technology and telecom companies.

## Acknowledgements

Rudolph Araujo provided significant support with reviewing the white paper.

## About Foundstone Professional Services

Foundstone® Professional Services, a division of McAfee. Inc. offers expert services and education to help organizations continuously and measurably protect their most important assets from the most critical threats. Through a strategic approach to security, Foundstone identifies and implements the right balance of technology, people, and process to manage digital risk and leverage security investments more effectively. The company's professional services team consists of recognized security experts and authors with broad security experience with multinational corporations, the public sector, and the US military.