

## Application Testing:

### Web Application Testing

---

Gartner Group recently estimated that 75 percent of all security breaches are due to vulnerabilities within the web application layer.

We hear about incident after incident where the attacker used the Web application to gain access to confidential information. Even eBay has been a victim; hackers stole user names and passwords, then stole customer credit card information by setting up fraudulent auction sites that mimicked eBay's own site.

The firewall and IDS even when well configured, cannot mitigate all application layer attacks.

The answer lies in looking at security, not just from an operations perspective, but as an integral part of the entire development lifecycle. Security must be built into the Web application development process itself.

Even in mature development organisations, the most commonly overlooked areas of the application development process is Web application security.

Security has traditionally been focused on the network and server layer of an application's architecture. However, this lack of understanding often leads to weak application security.

Development platforms such as Websphere, WebLogic and ColdFusion provide powerful environments that greatly improve development productivity. However, these require careful configuration to ensure that only the appropriate services are running in the live environment. Unfortunately, many development houses neglect to configure the production environment properly, which leaves the Web application vulnerable.

Web application security is usually an afterthought for most organisations. As security is not directly related to functional requirements, users do not focus on it and developers generally fail to put in the necessary time to ensure that Web applications are secure. Even developers who do see the importance of Web application security usually view it as task that is performed as part of the QA process. As a result, many Web applications may be functionally rich, but still vulnerable to unwanted intrusions and attacks at the application layer.

Web applications are not static systems. Changes to Web applications create risks; what was once secure is now vulnerable. If security is viewed as a single event, a vulnerability that enters the system after the audit will go undetected

## Application Testing:

### Non Web Applications

---

Application security testing strategies are typically geared toward one of two paths; the so-called 'white box' or 'black box' approaches.

A 'black box' security assessment will involve extensive analysis and reverse engineering of the application concerned, ensuring that all possibility to circumvent program flow or achieve compromise of resources is isolated.

In a security context, 'black box' testing is typically an accurate simulation of what a malicious individual would be able to achieve given adequate time and resource. Such testing is typically conducted under 'zero knowledge' conditions.

In contrast the 'white box' approach requires a knowledge and understanding of the workings of the application - typically including provision of information such as source code and UML/architectural diagrams. This approach is easily implemented with the existing functionality test plans of the application.

Whilst the 'black box' approach allows for an accurate picture of the risk a non-privileged user may pose to the system integrity, a considerable amount of time is spent familiarising with the application functionality. 'White box' testing, in its true sense, includes security testing only of elements existing within the application test plans and functionality set - neglecting the all-important test of the system as a whole.

SecureTest merge the white box and black box strategies in their application testing methodologies - in order to identify likely risks associated with implementation, and also to make most efficient use of the time allocated for the penetration test.

The SecureTest application testing methodology has proven useful in real world situations - for example identifying vulnerabilities existing outside the architect's understanding of how the system is implemented, caused by programming 'short cuts' and poor adherence to software engineering practices

## Application Testing:

### Mobile Phone Applications

---

Traditionally, security has been viewed as a problem to be managed using firewalls & rigorous patch management processes. However as organisations open new routes to communicate and trade with their clients, security of these have to be addressed, not only at the infrastructure level but also in the application layer.

In the past, the mobile phone was seen by the hacker community as something of an unknown – developer tools were only available commercially, and access to the phone operating system required very specialist knowledge. Moreover, each handset was different, so by the time the knowledge became available to the general internet, the phone was probably nearing end-of-life.

Developers producing applications for mobile devices could code reasonably safe in the knowledge that vulnerabilities in their apps were unlikely to be discovered.

However, with the advent of smartphones, all this changed. Now there were widely available tools to access the filesystems of the mobile. This immediately exposed significant opportunities to interfere with the purchase of premium applications and content.

For example:

I want to pay for and download an application from a mobile gaming site.

I enter my details into the phone, or into the web site. My phone receives a WAP push message giving it the details of where to download the application.

By viewing the properties for the message, we find the url the phone will try to access over GPRS/WAP reads:

```
http://target IP/novo/ota?id=xyz
```

Once this message has been accepted, the mobile phone connects to the server and downloads a file named 'ota.jad'. This file is placed in a temporary directory in the smartphone. Ordinarily, this temporary file would not be accessible by the handset owner, however, with 3rd party tools letting the user access the entire filesystem, this file can be downloaded from the handset and read. In this case the file contains the following information:

```
MicroEdition-Configuration: CLDC-1.0  
MicroEdition-Profile: MIDP-1.0  
MIDlet-Jar-URL: http://target IP/midlets/p900/game.jar  
MIDlet-Icon: /graphic.png  
MIDlet-Data-Size: 800  
MIDlet-Name: xxx  
MIDlet-Version: 1.0  
MIDlet-Jar-Size: 78452
```

So now we have the source for the application download. So instead of downloading to the phone, where scraping the installed application will be hard work, we simply download it to a PC, and distribute using a P2P file sharing application.

## Application Testing:

You may find that access to the download site is restricted: change your PC browser user agent to that of a mobile phone user agent; a simple task, particularly in Firefox. You should then be able to access the site.

And organisations wonder why their content is being ripped off!

So the hacker has stolen the content, resulting in lost sales, but are there more serious implications for mobile security?

In the case of applications which allow trading or transactions using a mobile, the mobile application source code is often accessible, particularly in the case of Java apps. One simply decompiles it, and reads the code!

Next, we can investigate routes to modify the code to remove, for example, client side input validation which could lead to an SQL Injection attack against the server. This is less complex than it sounds!

We recompile the code, run it using a phone emulator on a PC, then we have access to the online service without the protection that was initially built in to the application.

The opportunity for a malicious user from here on is frightening.

To mitigate the code-modification problem, there are a number of solutions, each with their own pitfalls:

First, validate all input received at the server from the mobile. Don't assume that it's your original mobile application running on the phone - once it's in free circulation, there's no knowing what has been done to the app

One can obfuscate the application code, making it extremely difficult to read, although any determined attacker will eventually work out the application logic.

Once your mobile client application has been distributed to the market, it isn't your application any more. Don't trust any input you receive from it.

Ideally one would hold more of the application code server side in order that the user cannot modify the content, however this would result in increased traffic to and from the handset, increasing call cost and longer user wait times.