Stopping Automated Attack Tools

An analysis of web-based application techniques capable of defending against current and future automated attack tools

Abstract

An almost infinite array of automated tools exist to spider and mirror application content, extract confidential material, brute force guess authentication credentials, discover code-injection flaws, fuzz application variables for exploitable overflows, scan for common files or vulnerable CGI's, and generally attack or exploit web-based application flaws. While of great value to security professionals, the use of these tools by attackers represents a clear and present danger to all organisations.

These automated tools have become increasingly popular for attackers seeking to compromise the integrity of online applications, and are used during most phases of an attack. Whilst there are a number of defence techniques which, when incorporated into a web-based application, are capable of stopping even the latest generation of tools, unfortunately most organisations have failed to adopt them.

This whitepaper examines techniques which are capable of defending an application against these tools; providing advice on their particular strengths and weaknesses and proposing solutions capable of stopping the next generation of automated attack tools.

Author

Gunter Ollmann, Professional Services Director, NGS – email: gunter [at] ngssoftware.com

Section 1: **Background**

For an increasing number of organisations, their web-based applications and content delivery platforms represent some of their most prized and publicly visible business assets. Whether they are used to provide interactive customer services, vital client-server operations, or just to act as informational references, these assets are vulnerable to an increasing number of automated attack vectors – largely due to limitations within the core protocols and insecure application development techniques.

As these web-based applications become larger and more sophisticated, the probability of security flaws or vulnerabilities being incorporated into new developments has increased substantially. In fact, most security conscious organisations now realise that their web-based applications are the largest single source of exploitable vulnerabilities.

Over recent years the ability to discover and identify these application flaws has become a critical assessment phase for both professional security agencies and would-be attackers. To increase the speed and reliability of identifying application-level vulnerabilities and potential exploitation vectors, both groups make extensive use of automated scanning tools.

These automated scanning tools are designed to take full advantage of the state-less nature of the HTTP protocol and insecure development techniques by bombarding the hosting server with specially crafted content requests and/or data submissions. Depending upon the nature of the scanning product its purpose may be to create a duplicate of the client-visible content (e.g. content mirroring); search for specific content (i.e. administrative pages, backup files, e-mail addresses for spam); fuzz application variables to elicit server errors and uncover exploitable holes (e.g. SQL injection, cross-site scripting), or even to conduct a brute force discovery of hidden content or customer authentication credentials.

While there are a vast number of defensive strategies designed to help protect a web-based application against actual exploitation, very few of these strategies provide adequate defence against the initial phases of an attack – in particular the high dependency upon automated scanning tools.

By adopting a number of simple design criteria and/or incorporating minor code changes to existing applications, many organisations will find that the current generation of application scanning tools are ineffective in the discovery of probable security flaws; thereby helping reduce the likelihood for future exploitation.

Section 2: **Automated Scanning**

Given the size and complexity of modern web-based applications, the use of automated scanners to navigate, record and test for possible vulnerabilities has become a vital stage in confirming an application's security. Without the use of automated scanning tools, the process of discovering existing security vulnerabilities is an extremely time consuming task and, when done manually, dependant upon the raw skills of the security consultant or attacker.

Therefore, automated scanning tools are a key component in any attacker's arsenal – particularly if they wish to identify and exploit a vulnerability with the least amount of effort and within the shortest possible timescale.

2.1. Developments in Automated Scanning

Just as web-based applications have evolved over the past decade, so too have the automated tools used to scan and uncover potential security vulnerabilities. Whilst the vast majority of these tools and techniques have come from non-commercial and "underground" sources, the quality of the tools is generally very high and they are more than capable of discovering vulnerabilities in most current application developments and/or deployments.

These automated scanning tools have undergone a series of evolutionary steps in order to overcome the security benefits of each advance in web-development technology, and can be divided into a small number of technological groupings or "generations".

This evolution of automated scanning tools can be quickly condensed into the following "generations":

- **1st Generation –** The first generation of automated application scanners did no processing or interpretation of the content they attempted to retrieve. These tools would typically use lists of known file locations (e.g. file locations associated with common IIS administration pages, Compaq Insight Manager pages, Apache root paths, etc.) and sequentially request each URL. At the end of the scan, the attacker would have a list of valid file locations that could then be investigated manually. A common example of a 1st generation tool is a CGI Scanner.
- **2nd Generation –** The 2nd generation of automated scanners used a form of application logic to identify URL's or URL components contained within an HTML-based page (including the raw client-side scripting content) and navigate to any relevant linked pages – repeating this process as they navigate the host content (a process commonly referred to as 'spidering' or 'spydering'). Depending upon the nature of the specific tool, it may just store the content locally (e.g. mirroring), it may inspect the retrieved content for key values (e.g. email addresses, developer comments, form variables, etc.), build up a dictionary of key words that could be used for later brute forcing attacks, or compile a list of other metrics of the application under investigation (e.g. error messages, file sizes, differences between file contents, etc.) for future reference.
- **2.5 Generation –** A slight advance over second generation scanners, this generation of scanners made use of a limited ability to reproduce or mimic the applications presentation layer. This is typically accomplished by the tool memorising a number of default user clicks or data submissions to get to a key area within the application (e.g. logging into the application using valid credentials) and then continuing with standard 1st or 2nd generation tool processes afterwards. Automated scanning tools that utilise this approach are commonly used in the load or performance testing of an application. Also included within this generational grouping are scanning tools that can understand "onclick" events that build simple URL's.
- **3rd Generation –** 3rd generation scanning tools are capable of correctly interpreting client-side code (whether that be JavaScript, VBscript, Java, or some other "just in time" interpreted language) as if rendered in a standard browser, and executing in a fashion similar to a real user.

Whilst there are literally thousands of tools that can be classed as 1st, 2nd or even 2.5 generation, there are currently no reliable 3rd generation scanning tools capable of correctly interpreting client-side code without a great deal of customisation or tuning for the specific web-technology application under investigation.

2.2. What is an automated scanner?

As far as web-based applications are concerned, there are a number of methods and security evaluation techniques that can be used to uncover information about an application that has a security context. An automated scanner makes use of one or more discovery techniques to request data and scans each page returned by the web server and attempts to categorise or identify relative information.

Within the security sphere, in the context of an attack, the key functions and discovery techniques that can be automated include the following:

- **Mirroring –** The attacker seeks to capture or create a comprehensive copy of the application on a server or storage device of their choosing. This mirrored image of the application content can be used for:
  - Theft and repackaging of intellectual property.
  - Part of a customer deception crime such as man-in-the-middle attacks, Phishing, or identity theft.
- **Site Scraping or Spidering –** The attacker's goal is to analyse all returned data and uncover useful information within the visible and non-visible sections of the HTML or client-side scripts. Information gleaned in this process can be used for:
  - Harvesting of email addresses for spam lists.
  - Social engineering attacks based upon personal data (such as names, telephone numbers, email addresses, etc.).
  - Ascertaining backend server processes and software versions or revisions.
  - Understanding development techniques and possible code bypasses based upon "hidden" comments and notes left behind by the application developer(s).
  - Uncovering application details that will influence future phases in the exploitation of the application (e.g. references to "hidden" URL's, test accounts, interesting content, etc.).
  - Mapping the structure of application URLs and content linking/referencing.
- **CGI Scanning –** The inclusion of exhaustive lists of content locations, paths and file names to uncover existing application content that could be used in later examinations or for exploitation. Typically, the information being sought includes:
  - Likely administrative pages or directories.
  - Scripts and controls associated with different web servers and known to be vulnerable to exploitation.
  - Default content and sample files.
  - Common "hidden" directories or file path locations.
  - Shared web services or content not directly referenced by the web-based application.
  - File download repository locations.
  - Files commonly associated with temporary content or backup versions.
- **Brute Forcing –** Using this technique, an attacker attempts to brute force guess an important piece of data (e.g. a password or account number) to gain access to additional areas or functionality within the application. Common techniques make use of:

- Extensive dictionaries.
- Common file or directory path listings.
- Information gathered through site scraping, spidering and CGI scanning.
- Hybrid dictionaries that include the use of common obfuscation techniques such as elite-speak.
- Incremental iteration through all possible character combinations.

★ **Fuzzing –** Closely related to brute forcing, this process involves examining each form or application submission variable for poor handling of unexpected content. In recent years, many of the most dangerous application security vulnerabilities have been discovered using this technique. Typically each application variable is tested for:

- Buffer overflows,
- Type conversion handling,
- Cross-site scripting,
- SQL injection,
- File and directory path navigation,
- Differences between client-side and server-side validation processes.

2.2.1. Automated Tool Classes

When discussing automated application scanning and security tools, the most common references or classes for breakdown are:

★ **Web Spider –** any tool that will spider, scrape or mirror content. Search engines can often be included within this grouping.

★ **CGI Scanner –** any tool that uses a file or path reference list to identify URL's for future analysis or attack.

★ **Brute Forcer –** any tool capable of repetitive variable guessing – usually user ID's or passwords.

★ **Fuzzer –** typically an added function to a web spider or personal proxy tool which is used to iterate through a list of "dangerous content" in an attempt to elicit an unexpected error from the application. Any unexpected errors would be manually investigated later with the purpose being to extend the "dangerous content" into a viable attack vector.

★ **Vulnerability Scanner –** most often a complex automated tool that makes use of multiple vulnerability discovery techniques. For instance the vulnerability scanner may choose to use spidering techniques to map the application after which it then inspects the HTML content to discover all data submission variables and then proceeds to submit a range of knowingly bad characters or content to elicit an unexpected response – finally it attempts to classify any discovered vulnerabilities.

Section 3: **Frequently Used Defences**

Over the years a number of defences have been experimented with in order to help protect against the use of automated scanning tools. Most of the defensive research and experimentation has been conducted by web sites that have to protect against tools that capture the contents of the web application/site (e.g. downloading of all images from a 'porn' site) or brute force guessing customer login credentials.

The most 10 most frequently utilised defences are:

- Renaming the server hosting software
- Blocking HEAD requests for content information,
- Use of the REFERER field to evaluate previous link information,
- Manipulation of Content-Type to "break" file downloads,
- Client-side redirects to the real content location,
- HTTP status codes to hide informational errors,
- Triggering thresholds and timeouts to prevent repetitive content requests,
- Single-use links to ensure users stick to a single navigation path,
- Honeypot links to identify non-human requests,
- Turing tests to block non-human content requests.

3.1. Server Host Renaming

An early method of thwarting 1st generation automated tools exploited their reliance upon the host server version information. Application logic within these early tools made use of a check to see exactly what type of web server they were running against by reading the Server variable within the HTTP headers and then using this information to select the most appropriate list of checks it would then execute.

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Content-Location: http://www.example.com/PageIsHere.html
Date: Fri, 01 Jan 2005 01:01:01 GMT
Content-Type: text/html
Accept-Ranges: bytes
Last-Modified: Fri, 01 Jan 2005 01:01:01 GMT
Content-Length: 1337
```

By changing the Server variable from one server type/description to another (e.g. "Microsoft-IIS/5.0" becomes "Apache/1.3.19 (Unix)"), this could often be enough to deceive the tool and prevent it from discovering vulnerable CGI's and URL's.

Advantages	Disadvantages
Easy to Implement The changing of the host Server variable is a simple process and can be done at any time by the system administrator. **Low Impact** No changes to the web application are necessary and there will be very little impact (if any) to the hosted web application **Useful Against:** 1st Generation CGI Scanners	**Old Technique** Almost all later generation automated tools have overcome this protection method and ignore the Server variable. **Specialist Tools** A new range of server-discovery tools have been developed to specifically identify the server type through discrepancies in how HTTP responses are created.

3.2. Blocking of HEAD Requests

There are a number of legitimate methods in which a client browser can request content from a web-based application. The most common, GET and POST, are used to elicit a response from the application server and typically receive HTML-based content. If the client browser does not want to receive the full content – but instead wishes to know whether a link exists or that the content is unchanged for instance – it can issue a HEAD request (with formatting almost identical to a GET request).

Many 1st generation automated scanners choose to use HEAD requests to spider an application or identify vulnerable CGI's instead of GET requests because less data is transferred and consequently the scanning or enumeration can be conducted at a greater speed.

Defending against automated scanners that rely upon HEAD requests is trivial. Almost all web hosting servers can be configured to not respond to HTTP HEAD requests – and only provide content via an approved list of HTTP options. This type of configuration is quite common; however, there may be ramifications for data throughput (this may increase as any content request must now retrieve the full volume of data instead of just the file/page headers) and the number of dropped connections may also increase (some tools, after identifying that HEAD requests do not work, will use GET requests and forcibly drop the connection once it has received the header data within the GET response).

Advantages	Disadvantages
Easy to Implement Dropping support for the HTTP HEAD response is easily achieved through standard web server administration facilities. **Low Impact** No changes to the web application are necessary and there will be very little impact (if any) to the hosted web application **Useful Against:** 1st Generation CGI Scanners 1st Generation Web Spiders 1st Generation Fuzzers	**Higher Data Throughput** Dropping support for HTTP HEAD requests means that client-browsers must perform a full HTTP GET request, even if they are just trying to discover the last update date/time. Therefore higher volumes of network traffic are likely. **Dropped Connections** Some tools will use the HTTP GET instead of HEAD to pull down header information and will just drop the TCP connection when they have this information. This dropping process will create extra web log entries.

3.3. Use of the REFERER Field

One of the most popular methods of governing access to the web applications content is often through the use of the Referer entity-header field within the client browser's submitted HTTP header. Ideally, each time a client web browser requests content or submits data, the HTTP header should contain a field indicating the source URL from which the client request was made. The application then uses this information to verify that the users request has come via an approved path – delivering the requested content if the referrer path is appropriate, or stopping the request if the Referer field is incorrect or missing.

For instance, the user is browsing a content page with a URL of http://www.example.com/IWasHere.html containing a link to the page http://www.example.com/Next/ImGoingHere.html. By clicking on the link, the user will make a HTTP request to the server (www.example.com) containing the following headers:

```
GET /Next/ImGoingHere.html HTTP/1.1
Host: www.example.com
Referer: http://www.example.com/IWasHere.html
Accept-Language: en-gb
Content-Type: application/x-www-form-urlencoded
```

The application must maintain a list (or use an algorithm) for validating appropriate access paths to the requested content, and will use the Referer information to verify that the user has indeed come from a valid link. It is not uncommon to reduce the amount of checking by

restricting the check to verifying that it just contains the same domain name – if not, the client browser is then redirected to the sites main/initial/login page.

Many 1st and 2nd generation automated scanners do not use (or update) the Referer field within the HTTP header of each request. Therefore, by not processing content requests or submissions with missing or inappropriate Referer data, the application can often block these tools.

It is important to note that some browsers may be configured to not submit a Referer field, or they may contain a link or data of the user's choice as a method of reducing any leakage of personal information. Additionally, if the user follows a link from another site (e.g. a search engine) or their saved favourites, any content restrictions based upon Referer information will also be triggered.

Advantages	Disadvantages
Robust Method Use of the REFERER method provides a robust mechanism for tracking how a user has reached the specific application content. **Identification of Page Editing** REFERER information can be used to identify content pages that have been saved locally for manual editing and consequential attacks. In addition, in the case of frequent data submissions, if the site content has been copied or mirrored without permission it is possible to identify the offending site location. This process can be useful for spotting man-in-the-middle attacks. **Useful Against:** CGI Scanners Mirroring Software 1st & 2nd Generation Web Spiders 1st Generation Vulnerability Scanners	**Direct Links** If an application user connects to application resources from their client browser favourites, emailed links, or types in the URL, no REFERER information will be available. **Browser Privacy** Some client browsers can be configured to not submit REFERER information as a form of privacy control. In addition, some personal firewalls and corporate proxies may also strip away this information.

3.4. Content-Type Manipulation

Another method of preventing automated tools from downloading vast amounts of site content is through the use of Content-Type entity-header field manipulation.

The Content-Type field is typically used to indicate the media type of the entity-body sent to the recipient or, in the case of a HEAD request, the media type that would have been sent had the request been a GET request. For example, in the following request the Content-Type has been set by the server to be text/html:

```
HTTP/1.0 200 OK
Location: http://www.example.com/ImGoingHere.html
Server: Microsoft-IIS/5.0
Content-Type: text/html
Content-Length: 145
```

Alternatively, MIME Content-Type can be defined within the actual content through META tags using the HTTP-EQUIV attribute. Tags using this form are supposed to have the equivalent effect when specified as an HTTP header, and in some servers may be translated to actual HTTP headers automatically or by a pre-processing tool.

```
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=koi8-r">
```

The application server can define a MIME Content-Type for each and every data object, and is normally used to define how the client browser should interpret the data. There are dozens

of content types defined and in common usage, with more being defined all the time. Some of the most frequently encountered definitions include:

MIME Type	File Extension	What to Do
text/html	html, htm	view in browser
text/richtext	doc, rft	view in text editor
image/gif	gif	view in browser
image/jpeg	jpg, jpeg	view in browser
image/x-png	png	view in browser
video/avi	avi	play in media player
video/mpeg	mpg, mpeg	play in media player
application/pdf	pdf	view in Adobe Acrobat
application/java	java, class	execute in sandbox
application/msword	doc	open using Microsoft Word
application/octet-stream	bin, exe	download-to-disk dialog
application/x-zip	zip	ask user whether to download to disk, or open with WinZip

By altering file extensions and assigning them non-default MIME types through the use of the servers Content-Type response, it is often possible to trick 1st and 2nd generation automated scanning tools into either ignoring application links or misinterpreting the data they receive.

Automated web spiders and vulnerability scanners are tuned to ignore files that do not contain HTML content (e.g. GIF, JPG, PDF, AVI, DOC, etc.) and the majority of existing tools do not analyse MIME information contained within server HTTP headers. Therefore, for example, by renaming .HTML files to .JPG and ensuring that the Content-Type remains "text/html", a document containing valid HTML content but called "ImGoingHere.jpg" will be correctly rendered as a web page in a browser, but will be ignored by an automated scanner.

Advantages	Disadvantages
Server Configuration Setup of mime types can be done at the web server without major modifications to the web application. **Application Coding** Since "Content-Type" can be defined within META tags, it is possible to implement this solution from within the application itself. Application developers can choose exactly which content will be protected using this mechanism. **Useful Against:** Mirroring Software Web Spiders 1st Generation Vulnerability Scanners	**Browser Presets** While client browsers are supposed to use the information passed through the server Content-Type variable, some browsers may maintain a preset list of actions for a specific file type causing the content to not be correctly interpreted.

3.5. HTTP Status Codes

The majority of users are familiar with the common status codes "200 OK", "302 Redirect" and "404 File Not Found". The HTTP protocol provides for a multitude of status codes which a web server can select and send to the client browser following a data connection or request. These status codes are divided into the following 5 key groupings:

Status Code	Allocated Meaning
1xx	Informational
2xx	Successful
3xx	Redirection
4xx	Bad Request
5xx	Internal Server Error

From an automated tools perspective, the “200 OK” status code is typically interpreted as a valid request was made to the server (e.g. the page exists and the URL is correct), while any other returned status code in the 4xx and 5xx groupings can be used to ascertain whether the request was invalid or triggered a server-side fault. Depending upon the nature of the automated tool, a 5xx status response could be indicative that malicious content insertion may be possible (e.g. SQL injection, unsigned integer denial of service) and is worthy of manual investigation and further attack.

For instance, a CGI scanner will cycle through a list of known files and file paths – rapidly requesting content from the web-based application server. If a “200 OK” is received, the CGI scanner then reports to the attacker that the path or vulnerable page/content exists. If a “404 Not Found” is received, the scanner assumes that the content doesn’t exists and is therefore not vulnerable to that attack vector – and most likely will not report anything back to the attacker.

However, all modern HTTP web servers allow for bespoke error handling and customisation of status code representations. Consequently, a highly successful method of defeating the usefulness of automated scanners is to always present the same status code (i.e. “200 OK”) for every request – regardless of whether the request was legitimate, requested non-existent content, or generate an unknown server error. This means that the automated scanner cannot base its findings on HTTP status codes, and must then use some form of content inspection logic to analyse the actual content of the HTML body instead.

Advantages	Disadvantages
Simple The process of turning all error messages or status updates into “200 OK” messages is an easy task and can be accomplished without modification of the web application if required. **User Experience Consistency** Legitimate web application users are likely to be less confused or anxious if they are not confronted with different error messages. **Developer Control** The application developer can also trap all application error requests and issue a standard response (or selection of responses) that, while the rendered HTML content will be different, will still be a valid page with a “200 OK” status message. This provides application-level flexibility in the responses instead of modifying the web server’s configuration. **Useful Against:** Fuzzers Brute Forcers CGI Scanners Vulnerability Scanners	**Log Analysis** If each response generates a “200 OK” message even if the request is invalid, post attack analysis of the web server logs will not be particularly insightful. Instead an alternative logging system must be built into the custom web application.

3.6. Client-side Redirection

For many automated scanners, the process of identifying a link or embedded URL is done by searching for relevant “HREF=” references within the HTML content. However, there are a number of alternative methods for indicating URL’s within the HTML body of a server response.

A mechanism called “client-side redirection” is commonly used to redirect browsers to the correct content location after requesting invalid, nonexistent or recently moved content. The most common non-scripted method is through the use of the “Refresh” field (note that the “Refresh” field also allows for a wait period before being automatically redirected). Just like the “Content-Type” field, the “Refresh” field can be contained within the HTTP header or used within an HTTP-EQUIV META tag; for example:

```
HTTP/1.0 200 OK
Server: Microsoft-IIS/5.0
Content-Type: text/html
Refresh: 3;URL=http://www.example.com/ThisWay.html
```

Or

```
<META HTTP-EQUIV="Refresh" CONTENT="3;URL=http://www.example.com/ThisWay.html">
```

To use client-side redirection as a protective measure against automated scanners, the application developer must ensure that each URL for (valuable) content is initially intercepted by a page designed to automatically redirect the client browser to the correct/real content. For additional security, the application server could also enforce a minimum “wait” time before responding to requests for the real content.

The effect on many automated scanning tools is to induce a “200 OK” status code for each request – therefore having many of the benefits described in the earlier section.

Advantages	Disadvantages
Simple Configuration The configuration of client-side redirection is a simple process and can be achieved with very little fuss or effort. **Controllable Delays** The Refresh variable allows for controllable automatic redirection of the client browser. The application could also validate that a client browser has indeed waited the correct amount of time before requesting the redirected page content. If the client browser hasn’t waited, the request could be dropped or interpreted as a possible attack. **Choice of Location** The application developer can choose where they wish to place the redirection field (i.e. HTTP header or META data) and can switch between methods as necessary. **Useful Against:** Web Spiders Mirroring Software Fuzzers Vulnerability Scanners	**Repetitive Delays** Overuse or reliance upon this method is likely to affect the user experience of the application – particularly if time delays are used. **Browser History** If the user relies upon their “browser history” to navigate back and forward through previously requested pages it can become difficult for them to find the correct page.

3.7. Thresholds and Timeouts

In applications where session ID's are used to maintain the state of a connection (e.g. uniquely track the user or identify the fact that they have already successfully authenticated themselves), it is also common practice to measure two key interaction variables – the time and frequency of each request or data submission.

Normally, by monitoring the elapsed time since the last data submission, an application can "timeout" a session and force the user to re-login if they have not used the application for an extended period (e.g. an e-banking application that automatically logs out the user after 5 minutes of inactivity). However, it is also possible to monitor the time taken between data submissions – thereby identifying whether an automated tool is processing URL's at a speed that is unattainable or unlikely for a legitimate human user.

In addition, multiple requests for the same application content using the same session ID can also be monitored. This is commonly implemented as part of an authentication process designed to identify brute force guessing attacks (e.g. repeated guesses at the password associated with an email address on a free web-mail application server) – typically tied to account lockout and/or session cancellation. A similar process can be used to identify repeated attempts to access or submit to the same URL (e.g. a particular CGI or page) – as would occur during a fuzzing attack using an automated tool.

Consider the following HTTP POST data submission:

```
POST /Toys/IWantToBuy.aspx HTTP/1.1
Host: www.example.com
Referer: http://www.example.com/Toys/ILikeThisOne.aspx
Accept-Language: en-gb
Content-Type: application/x-www-form-urlencoded
Content-Length: 437
Cookie: SessionID=sse9d7783790

Postcode=SW11%201SA&Var1=Yes&Var2=Yes&Account=';--<H1>
```

In this example we see one captured POST submission to the application server. The attacker is fuzzing the "Account" field of the "/Toys/IwantToBuy.aspx" page by repeatedly trying different attack strings (e.g. ';--<H1> in this instance). We know that it is the same attacker because all previous requests have used the same session ID. To identify the attack, the application server maintains a couple of extra data variables associated with the session ID information in its backend database – in this case "last requested URL" and a numeric counter. Each time the "last requested URL" is the same, the counter is incremented. Once the counter threshold is reached (e.g. 5 repeated requests), the session ID is revoked and any subsequent data submissions using that session ID are then ignored.

The use of thresholds and timeouts within an application can prove to be successful against all generations of automated scanner. However, once an attacker understands the limits of these two mechanisms (i.e. how many times can he request the same page, and how "slow" the requests need to be to pretend to be human) the automated tools can often be configured to not trigger these application responses.

Advantages	Disadvantages
Request Frequency The ability to identify how fast a user is requesting new application content or submitting data requests is an important advantage in identifying an automated attacks – regardless of the type of tool the attacker is using. **Multiple Request Thresholds** By counting the number of repeated requests to the same page content, it is easy to identify a malicious attack (automated or manual) and trigger an appropriate response system – regardless of the type of request or	**Robust Session Management** This solution requires a robust and well thought out session management system. This type of management must be custom built by the application developer as current off-the-shelf solutions are currently inadequate.

content that may have been submitted.

Controlled through Session Management
By tying these thresholds and timeouts to the SessionID, it becomes an easier task to manage more sophisticated responses to an attack in progress.

Per-page Control
Different thresholds and responses can be associated with individual application pages or requirements. This enables an application developer to fine tune responses to an automated attack.

Useful Against:
Web Spiders
Mirroring Software
Fuzzers
Brute Forcers
Vulnerability Scanners

3.8. Onetime Links

Related to the application logic utilised in managing URL request and data submissions through the HTTP Referer field, in some cases it is possible to assign a unique "referrer value" to each page the client browser requests. This "referrer value" is then used to manage the location of the user within the application and identify any requests deemed to be out of order.

For instance, consider the online retailers purchasing page /BuyStageOne.aspx?**track=1104569** which contains the following URL's in the page content:

```
http://www.example.com/Index.aspx?track=1104569
http://www.example.com/BuyStageTwo.aspx?track=1104569
```

Each URL, including the users current location, identifies a tracker variable ("track=") with a numeric value (initially "1104569"). If the user clicks on any link, this tracker value will also be submitted to the application server. Now, assuming that the user clicks on the last link to proceed with the purchasing process, he will proceed to the page "BuyStageTwo.aspx", but will also be issued with a new unique tracker number and, for example, the contents of the new page (e.g. /BuyStageTwo.aspx?track=**1104570**) may also contain the following URL's:

```
http://www.example.com/Index.aspx?track=1104570
http://www.example.com/BuystageTwo.aspx?track=1104570
http://www.example.com/BuyStageThree.aspx?track=1104570
```

Key things to note with this onetime link anti-scanner implementation are:

- The tracking number changes with each page, and the earlier number is revoked so that it cannot be used again by the user.
- Tracking numbers are bound to a per-user session ID.
- Before the application will process any page request or data submission, it must first verify the integrity of the session (i.e. is the session ID real and make sure it hasn't been revoked) and then verify that the tracking value is correct.
- Each URL or link, including the link "back" to the previous page (/BuyStageTwo.aspx) has a new tracking number. The default browser "back" and "forward" buttons will not work – therefore this functionality must be provided within the page itself.
- Any attempt to follow a URL without a tracking number, or use an invalid tracking number, would be handled by the application as either a user error or seen as an attack (automated or otherwise).

- ★ Although this example uses a sequential increase in tracker numbers, this is not necessary and the values could be random if required (the use of random tokens is recommended).
- ★ Whenever the user requests application content containing the correct tracker number, the tracking value can only be used once as a new value is assigned with the server response.

This kind of implementation is successful against most 1st and 2nd generation automated scanners. Many Spidering and Mirroring tools parallelise their requests to speed up the discovery/download process and would therefore fail to handle the per-request changing tracking numbers. Fuzzers too would be affected by this location state management system.

Note: whilst the examples above make use of URL's containing tracking numbers, the use of HTTP POST submissions instead of GET requests are to be recommended. For a full discussion on the best security practices for URL handling, readers are directed to the paper "Host Naming and URL Conventions" also written by the same author.

Advantages	Disadvantages
One Way The application users can only navigate the application content in a manner governed by the onetime linking. **Control of "Back" functionality** The default browser "back" button will not function as the user expects, therefore the application can prevent users from using this method of navigation. This control is useful for shopping cart applications. **Prevents Multithreaded Attacks** Since a new tracker ID is created with each submission and must be used for the next user request, automated tools can not multi-thread requests and instead must follow a single thread. **Slows Manual Attacks** Since the user must follow a strict path to their requests, even manual attacks are affected and are slowed down considerably. **Useful Against:** Web Spiders Mirroring Software Fuzzers Brute Forcers CGI Scanners Vulnerability Scanners	**Dynamic Page Generation** This method requires that the application dynamically generate page content. **Navigation History** The user will not be able to use navigation history or the browser "back" button to review previous application content. Users may become frustrated with this limitation. **Saved Links** Due to the tie between SessionID's and URL tracker information, users will not be able to use application links that are saved – e.g. "Add to Favourites" or emailed to others.

3.9. Honeypot Links

Since many scanning tools will automatically identify URL's within the HTML body of a page and blindly request linked content, it is possible to include "hidden" links within an applications content that will direct an automated tool to a continually monitored page. Fake or monitored links such as these fall under generic the category of "honeypots". By embedding these links within the HTML body in such a way that they would never be visibly rendered or "clickable" by a human user, any client request for this "hidden" content is most probably associated with an attack.

For example, the following content extract uses comment fields (i.e. <!-- and -->) and background colours (i.e. setting the link colour to be the same as the background colour) to

"hide" two URL's that would not normally be followed by a human user, but are typically followed by automated tools.

```
<BODY BGCOLOR="white">
Valid Links <BR>
<A HREF="http://www.example.com/index.html">Home</A><BR>
<A HREF="../Toys/IWantOneOfThose.html">Mine!</A><BR>
Invalid Link <BR>
<!-- HREF="../Bad.HTML"> -->
Hidden Link <BR>
<FONT COLOR="white"><A HREF="../Bad2.HTML">hidden</A></FONT>
</BODY>
```

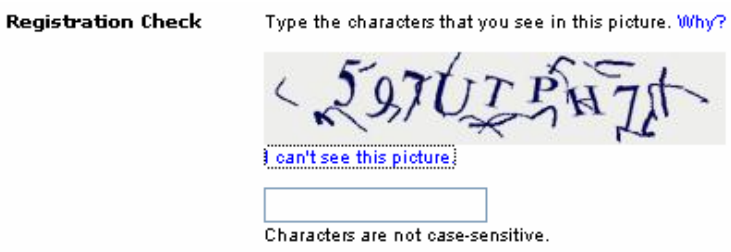
The web-based application would be designed in such a way that automated responses (e.g. session ID cancellation, automatic logoff, blocking of the attackers IP address, detailed forensics logging, etc.) are initiated should any request be made to access a honeypot link. Against standard automated scanners, the most likely response is to issue a default page (e.g. the home page) for all requests from that IP address or session ID – no matter what the request is – and initiate any background investigative processes.

Advantages	Disadvantages
Simple Setup It is a simple task to add honeypot links to the HTML content of the application and they can be obfuscated from standard browser rendering in a multitude of ways. **Customised Responses** The application developer can choose any response they wish to initiate after a request for a honeypot link is requested. Responses may range from informing the user that they are being monitored, through to session cancellation or detailed forensic logging. **Useful Against:** Web Spiders Mirroring Software	**Search Engine False Positives** Since search engines use the same techniques as automated spidering attack tools to identify and build URL's, there is a high probability of false positives. However, use of robots.txt to restrict which paths a search engine may navigate to would help prevent these false positives.

3.10. Graphical & Audio Turing Tests

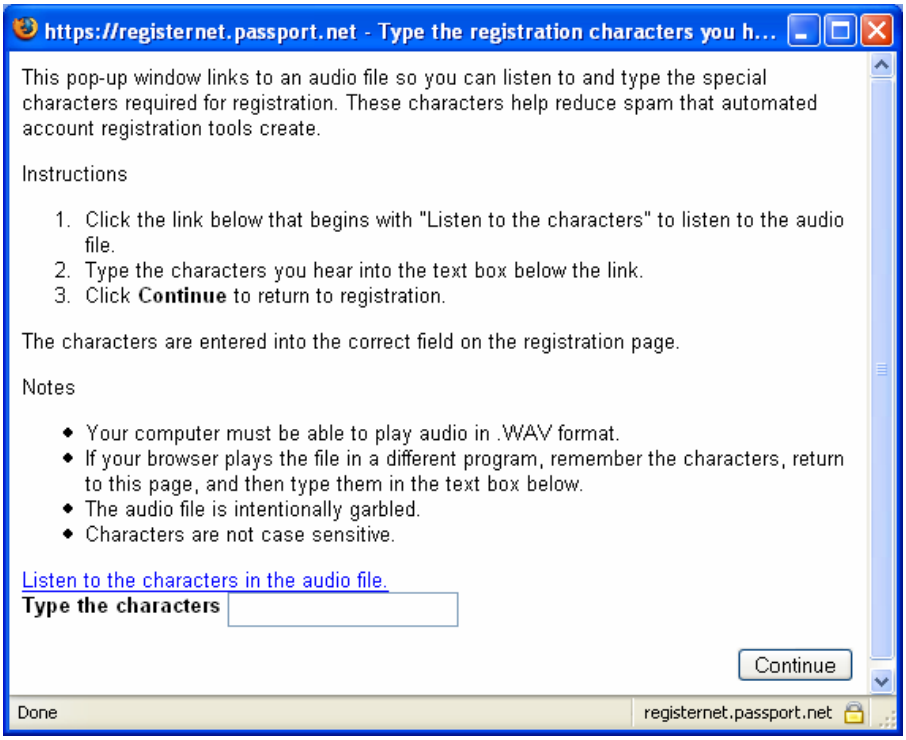
There are a number of ways in which the application can force the user to interpret onscreen or audio information, and submit a response that could not normally be supplied through an automated process (unless you include brute-force guessing) before proceeding into another section of the application. The most common implementations make use of graphical images containing a key word or value that cannot be discovered using tools (such as OCR), but must be manually entered in to a form field by the user.

For example, the following graphic is copied from the account creation phase of the Microsoft Passport online service. The background squiggles and leaning text is designed to help prevent automated OCR (Optical Character Recognition) packages from evaluating the text "597UTPH7".



Unfortunately, graphics such as the one above can be very difficult to understand for some people due to its complexity or personal circumstances (e.g. colour blindness, failing sight).

Therefore, alternative Turing tests that make use of audio sound bites can be used as an alternative. Microsoft's Passport registration also allows users to listen to a voice saying the pass phrase which must be entered correctly to set up the account. To make the process more difficult for automated dictation tools, some background noises and hisses may be included with the real pass phrase data. An example of Microsoft Passport support for a voice-based Turing test is shown below.



This kind of user identification testing is typically used at key points within high-volume applications (e.g. popular webmail services, online domain registration queries, etc.) that have, or are likely to, experience attacks or be used for non-authorised activities. Their purpose is to validate that it is a real person using the application – not an automated tool.

In theory, the ability to differentiate between a real person and a tool or computer system can be done through a specific test. These tests are often called Turing tests, and recent work in this area has led to the development of CAPTCHA (Completely Automated Public Turing Test to Tell Computers and Humans Apart – http://www.captcha.net/) systems for web-based applications.

Advantages	Disadvantages
Strong Non-Human Identification This type of testing is very difficult for automated tools to overcome – even tools that have been tuned for targeting a specific application. **Useful Against:** Web Spiders Mirroring Software Vulnerability Scanners	**High Failure Rate** As the tools become smarter, the Turing tests must become more difficult. Consequently some legitimate application users may also have trouble interpreting the information necessary for passing the test. **Cumbersome** Adding application functions that rely on passing a Turing test cannot be used too often as they will adversely affect usage of the application by legitimate users. These tests should only be used at key points within the application – such as initially creating the users new account or as part of an authentication process.

Section 4: Anti-tool Client-side Code

Whilst the techniques examined in the previous section provide various degrees of protection against automated tool attacks, there exists an additional array of defences capable of defending against all but 3rd generation scanners. These additional defences make use of client-side code.

The use of client-side code, from a security perspective, tends to be overlooked – largely due to a poor understanding of the different coding techniques and adverse publicity associated with frequent client-server content validation flaws. Although an attacker can indeed bypass client-side scripting components that validate content or enforce a sequence of events within the client browser fairly easily, it is still possible to use client-side code as a positive security component as long as suitable validation occurs at the server-side.

4.1. The Strengths of Client-side Code

As a mechanism for protecting against automated attack tools and scanners, client-side code provides numerous advantages over other protection mechanisms. However the greatest advantage is derived from the fact that current automated tools either cannot execute the code, or are extremely limited in their ability to interpret any embedded code elements.

The trick to using client-side code in a security context lies in ensuring that the client browser really did execute the code (i.e. validating execution) and did not simply ignore or bypass it. This can be achieved by forcing the client browser to submit a unique value that can only be obtained as part of the actual code execution. These code execution values, or "tokens", are submitted with any data request or submission, and validated by the server-side application prior to the processing of any other client-supplied data. This process can be referred to as "tokenisation"

While the final client-side code implementation may take on many forms, this functionality can be achieved using any modern client-side interpreted language including JavaScript, VBScript, Java, or even Flash. In fact, if so required, even compiled client-side components (e.g. ActiveX) could be used so long as the client-browser is likely to have it installed - although this is not recommended due to probable code flexibility issues. For ease of implementation it is recommended that client-side interpreted languages, which are available by default within modern client browsers, be used.

4.2. Client-side Scripting Alternatives

There are a near infinite number of ways to utilise client-side code elements as a protection device against automated attack tools – with each one influenced by factors such as the nature of the web-based application (e.g. e-banking, retailing, informational, etc.), the type of user (e.g. customer, administrator, associate, etc.), or even the personal preferences of the development staff.

However, there are three primary classes of client-side code elements capable of defending against most automated attack tools:

- ★ Token Appending.
- ★ Token Calculator.
- ★ Token Resource Metering.

4.2.1. Token Appending

The simplest of the client-side scripting techniques, token appending makes use of pre-calculated tokens embedded within the HTML body of the server-supplied content which must then be appended to any data submission or request by the client browser.

For instance, in the example below the HTML content contains a dynamically built link that uses JavaScript to populate the missing "token" value. Any tool that inspects the raw HTML is likely to identify the HREF entity but fail to include the necessary token value.

```
<SCRIPT LANGUAGE="javascript">
  var token="0a37847ea23b984012"
  document.write("<A
HREF='http://www.example.com/NextPage.aspx?JSToken="+token+"'>Link</A>")
</SCRIPT>
```

Alternatively, in the second example below, we see a POST submission form version. A JavaScript function, "addtoken()", exists in the head section of the HTML document which is called from the submission form ("myform") with the onClick routine. By default, the "token" field is set to "Fail" – meaning that any failure to process the JavaScript correctly will result in a POST submission containing the data "token=Fail" and would be interpreted by the server-side application as a possible attack.

The only valid way of submitting data using this form is by executing the "addtoken()" function after clicking on the submit button. This JavaScript function then modifies the "token" value by replacing the default "Fail" with the real value ("0a37847ea23b984012") and completes the submission.

```
<HTML>
 <HEAD>
  <TITLE>Example Post</TITLE>
  <SCRIPT>
    function addtoken() {
      document.myform.token.value="0a37847ea23b984012";
      document.myform.submit();
      }
  </SCRIPT>
 </HEAD>
<BODY>
  <FORM NAME="myform" ACTION="http://www.example.com/BuyIt.aspx" METHOD="POST">
    <INPUT TYPE="TEXT" NAME="ItemName" >Item Name<BR>
    <INPUT TYPE="RADIO" NAME="Buy" VALUE="Now">Now
    <INPUT TYPE="RADIO" NAME="Buy" VALUE="Later">Later<BR>
    <INPUT TYPE="HIDDEN" NAME="token" VALUE="Fail">
    <INPUT TYPE="BUTTON" VALUE="SUBMIT" onClick="addtoken()">
  </FORM>
</BODY>
</HTML>
```

The principles governing this tokenisation of a link are very similar to those discussed in section 3.8 "Onetime Links", and the server-side responses to an identified attack can be the same.

Advantages	Disadvantages
Easy to Implement The process of including client-side scripting tokenisation is very simple and can be achieved without any major changes to the web-based application. **Stops 1st, 2nd and 2.5 Gen Scanners** Current scanners and automated attack tools are incapable of correctly interpreting client-side code. Therefore, they cannot perform this kind of URL tokenisation. **Non-visible Impact on Users** Assuming that client-side scripting is enabled (the default for most client browsers), there is no visible or perceived impact on the users experience of the application **Useful Against:** All automated scanners	**Relies on Client-side Code** Some older browser types may not support the chosen client-side scripting language. In addition, some legitimate users may have disabled scripting support. Therefore, they are not able to use applications that make use of token appending strategies.

4.2.2. Token Calculator

Using almost identical techniques as the Token Appending class discussed previously, the Token Calculator class extends these principles by adding a dynamic token creation process.

Instead of using static tokens (e.g. token.value="0a37847ea23b984012"), the client-side script functionality is extended to include routines that actually calculate a token from scratch. For example, in the code snippet below, JavaScript is used to combine the "fake" default token with the session cookie and the page name, and then calculate a CRC32 checksum – that then replaces the "fake" token – and submits the form data to the application.

```
<HEAD>
  <TITLE>Example Post</TITLE>
    <SCRIPT TYPE="text/javascript" SRC="crc32.js"></SCRIPT>
    <SCRIPT TYPE="text/javascript" SRC="cookies.js"></SCRIPT>
    <SCRIPT>
        function encodetoken() {
           var token = document.myform.token.value;
           var cookie = getCookie("SessionID");
           var page = location.pathname;
           document.myform.token.value = crc32(token + cookie + page);
           document.myform.submit();
        }
    </SCRIPT>
</HEAD>
```

The routines used to calculate the token can take practically any form and may be as complex or as simple as the application developer feels comfortable with. The only limitation is that the server-side application must be able to verify the integrity and correctness of the token with each client browser data request or submission. Some examples of token calculation include:

- ★ Concatenating several text variables embedded within the HTML document to create a single submission token.
- ★ Performing a mathematical routine based upon variables embedded within the HTML document to create a numeric submission token.
- ★ Using HTML document properties (e.g. session cookies, browser-type, referrer field, etc.) or other user supplied form fields (e.g. user name, date of birth, etc.) to create a unique submission token for that page.

Advantages	Disadvantages
Easy to Implement The process of including client-side scripting tokenisation is very simple and can be achieved without any major changes to the web-based application. **Stops 1st, 2nd and 2.5 Gen Scanners** Current scanners and automated attack tools are incapable of correctly interpreting client-side code. Therefore they cannot perform this kind of URL tokenisation. **Non-visible Impact on Users** Assuming that client-side scripting is enabled (the default for most client browsers), there is no visible or perceived impact on the users experience of the application **Stronger than Token Appending** Since the token must be calculated instead of just appending a static value, any automated tool that can understand simple client-side scripting commands (such as	**Relies on Client-side Code** Some older browser types may not support the chosen client-side scripting language. In addition, some legitimate users may have disabled scripting support. Therefore they are not able to use applications that make use of token appending strategies.

taking variable A and appending it to string B) will be thwarted. **Token Versatility** Since the token is created "on the fly", it is possible to include additional information about the client browser than can only come from the client (e.g. language, local time, etc.) which can be used to make a stronger token. **Useful Against:** All automated scanners	

4.2.3. Token Resource Metering

Token Resource Metering extends and refines the principles of the Token Calculator and Token Appending strategies by increasing the complexity of the client-side code execution so that the client browser, when calculating the token, incurs a measurable time delay. The calculated token forms an "electronic payment" and can be used to slow down an automated attack. This process of slowing down data requests or submissions is commonly referred to as "Resource Metering".

The trick to a successful resource metering strategy lies in ensuring that the calculated token is easily calculated and validated at the server-side, but requires measurable effort to calculate at the client-side. For a full explanation of Resource Metering and how it can be used in a security context, readers are directed to the comprehensive paper titled "Anti Brute Force Resource Metering", also by the same author.

Advantages	Disadvantages
Stops 1st, 2nd and 2.5 Gen Scanners Current scanners and automated attack tools are incapable of correctly interpreting client-side code. Therefore, they cannot perform this kind of URL tokenisation. **Slows down 3rd Generation Attacks** Any automated attack tool must first correctly interpret the client-side code and execute it. This execution incurs a computational overhead that causes a time delay in data submission which means that the attack is slowed down. **Works Against Multithreaded Attacks** Due to processing overheads, conducting multithreaded attacks will cause the client machines CPU to maximum load and slow down the attack to the equivalent of calculating and submitting one token at a time. **Works Against Manual Attacks** The time delays incurred during the calculation of the token resource also affects manual attacks and will slow down the attack – most likely forcing the attacker to seek a softer target. **Useful Against:** All automated scanners Most manual repetitive attack techniques	**Relies on Client-side Code** Some older browser types may not support the chosen client-side scripting language. In addition, some legitimate users may have disabled scripting support. Therefore, they are not able to use applications that make use of token appending strategies. **Client Machine Specification** Since Token Resource Metering requires the client machine to carry out a computationally intensive routine, the code will take different amounts of time to execute for each hardware configuration. Therefore, the same code executing on a new desktop computer will be many times faster than that executing on a small PDA device over WAP.

Section 5: Conclusions

The use of automated tools to identify security weaknesses within web-based applications and to attack vulnerable content is an increasingly common practice. Therefore, it is important that organisations take adequate precautions to defend against the diverse range of tool techniques and increasingly sophisticated automated scanners used by current and future attackers.

The methods described within this whitepaper provide varying degrees of protection against these attack tools. Simple techniques such as changing host service names, blocking of HTTP HEAD requests and the use of non-informative status codes should be considered an absolute minimum for today's environments. More sophisticated techniques requiring tighter integration with the dynamically generated application content help to provide better protection, but must be factored early on into the application development lifecycle if they are to be effective.

As the attack tools become even more sophisticated and overcome many of the simpler defence techniques, organisations will be forced to consider the use of client-side code techniques. These techniques currently have the ability to stop all of the generic automated attack tools currently available – including most of the more sophisticated commercial vulnerability scanners (this is important since keygen's and license-bypass patches are in common use).

5.1. Comparative Studies

The techniques explained within this whitepaper for stopping automated attack tools all have their own unique strengths and weaknesses. For an organisation seeking to protect their web-based application from future attack, it is important that the appropriate defensive strategy be adopted and that the right anti-automated tool technique is applied. The following tables help to provide a comparative study of the different techniques previously discussed – however, it is important that application designers realise that these comparisons are of a general nature only (due to the phenomenal array of different automated attack tools which are currently available).

5.1.1. Technique vs. Tool Generation and Classification

Technique	Tool Generation				Tool Classification				
Host Server Renaming	**	*				*			*
Blocking HEAD	*				*	*			
REFERER Fields	***	**	*		**	***			*
Content-Type Manipulation	***	**	*		**				
Client-side Redirection	**	*			*	*	*		*
HTTP Status Codes	**	**	**		*	*	*	*	*
Thresholds & Timeouts	***	***	**	**	*	*	***	***	**
Onetime Links	*	***	**	*	*		***	***	**
Honeypot Links	***	***			***				
Turing Tests	***	**			***				**
Token Appending	***	***	***	*	**	***	***	**	***
Token Calculators	***	***	***	*	**	***	***	**	***
Token Resource Metering	***	***	***	***	***	***	***	***	***
	1st Generation	**2nd Generation**	**2.5 Generation**	**3rd Generation**	**Web Spidering**	**CGI Scanning**	**Brute Forcing**	**Fuzzers**	**Vuln. Scanning**

Key: [] No benefit, [*] Some benefit, [**] Noticeable Benefit, [***] Valuable Protection

5.1.2. Technique vs. Implementation and Client Impact

Technique	Implemented By		Client		Ease
Host Server Renaming	Y		N	N	Trivial
Blocking HEAD	Y	Y	N	N	Trivial
REFERER Fields		Y	N	Y	Average
Content-Type Manipulation	Y	Y	Y	N	Simple
Client-side Redirection		Y	N	Y	Simple
HTTP Status Codes	Y	Y	N	Y	Simple
Thresholds & Timeouts		Y	N	Y	Average
Onetime Links		Y	Y	Y	Average
Honeypot Links		Y	N	N	Trivial
Turing Tests		Y	Y	Y	Very Hard
Token Appending		Y	N	N	Average
Token Calculators		Y	N	N	Average
Token Resource Metering		Y	Y	Y	Hard
	System Administrators	Application Developers	Client Visible	Client Impact	Ease of Implementation

5.2. Authorised Vulnerability Scanning and Security Testing

It is important to understand that the use of automated attack tools play an important role in the legitimate identification of security vulnerabilities. Organisations should be mindful to ensure that any anti-tool defences they install can be overcome for authorised security testing. Failure to include such a mechanism is likely to result in extended and difficult security assessment and penetration testing exercises, which is likely to lead to high costs or a less thorough evaluation.

It is recommended that system administrators and application developers provide a mechanism to turn off many of the more sophisticated anti-automated tool defences based upon factors such as IP address, connection interface or even reserved SessionID's. For most of the defensive techniques discussed in this paper, any of these mechanisms could be used. However, care should be taken to ensure that this "bypass" mechanism is by default switched off, and must be temporarily enabled to allow automated security testing of the application and its hosting environment.

5.3. Combining Defence Techniques

It is important organisations design their online web-based applications in such a manner as to take advantage of as many of the anti-tool defensive techniques as possible. The ability to stack and combine compatible techniques will strengthen the application against attack – providing a valuable defence in depth.

5.4. Custom Attack Tools

The techniques outlined in this whitepaper provide varying degrees of defence against automated attack tools which are available through most commercial, freeware and underground sources. However, it is important to note that, should an attacker seek to purposefully target an organisation and take the necessary time and effort to fully engage or evaluate an applications defence, it is likely that they will be able to construct a custom automated attack tool capable of bypassing most of them.

Organisations should evaluate the likelihood of a potential attacker crafting a custom tool to overcome the applications anti-tool defences and seek to adopt appropriate strategies for

detecting anomalies in application usage (e.g. repetitive data submissions, high volumes of network traffic at odd hours from out of zone IP addresses, repeated use of the same credit card, etc.).

That being said, organisations that incorporate several of the defence techniques discussed in this paper (in particular adopting client-side code elements) will find that potential attackers are more likely to turn their attention to softer targets

5.5. Additional Resources

"The Phishing Guide", *Gunter Ollmann, 2004*

"Hacker Repellent", *Amit Klein, 2002*

"Security Best Practice: Host Naming and URL Conventions", *Gunter Ollmann, 2005*

"Anti Brute Force Resource Metering", *Gunter Ollmann, 2005*

About Next Generation Security Software (NGS)

NGS is the trusted supplier of specialist security software and hi-tech consulting services to large enterprise environments and governments throughout the world. Voted "best in the world" for vulnerability research and discovery in 2003, the company focuses its energies on advanced security solutions to combat today's threats. In this capacity NGS act as adviser on vulnerability issues to the Communications-Electronics Security Group (CESG) the government department responsible for computer security in the UK and the National Infrastructure Security Co-ordination Centre (NISCC). NGS maintains the largest penetration testing and security cleared CHECK team in EMEA. Founded in 2001, NGS is headquartered in Sutton, Surrey, with research offices in Scotland, and works with clients on a truly international level.

About NGS Insight Security Research (NISR)

The NGS Insight Security Research team are actively researching and helping to fix security flaws in popular off-the-shelf products. As the world leaders in vulnerability discovery, NISR release more security advisories than any other commercial security research group in the world.