



NETCONTINUUM

# Reducing the Risk of Web Attacks through Proper Input Validation

A TECHNOLOGY BRIEF



Hackers exploit vulnerabilities present in web applications due to software bugs, system integration incompatibilities or configuration errors. The most common way for hackers to orchestrate attacks is by injecting malicious code into the web application through HTML forms, headers or URL parameters. The ultimate goal of these attacks is theft or malicious destruction. Security industry participants use many names for these types of attacks, including but not limited to: SQL injection, Cross Site Scripting, OS Command Injection, Path Traversal and Meta character attacks.

This Technology Brief describes the methodologies that hackers use to exploit web applications that do not conduct proper input validation and how the NetContinuum NC-1000 web security gateway protects against web attacks caused by hackers injecting malicious code into web applications. For an introduction to the NC-1000 and its unique approach to web security, you may wish to begin by reading the companion white paper *Attack and Intrusion Prevention: a Practical Approach to Reducing Risk* available in the White Papers section of the NetContinuum website.

## **Background: The Scope of the Problem**

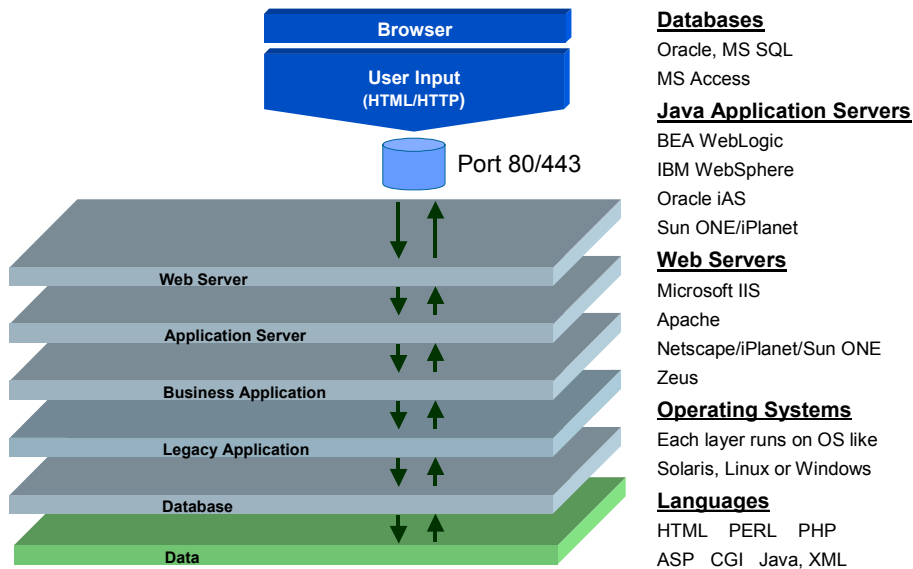
Web applications are complicated systems. They execute business logic and useful work based on a user's interaction with a web server via a browser. For example, a web application may help a user look up account information at their bank and transfer funds, or purchase goods online. The multi-layered complexity of these applications makes it difficult to secure them completely within the application software itself.

### **The nature of web applications**

Web applications are layered systems with multiple components including web servers, application servers, databases, operating systems, and numerous backend systems of an organization. Each component is composed of stacks of software code that may be derived from many sources. The software code accesses other pieces of software code to do work.

For the most part, each web application system is unique, comprising different components and software stacks. The software might be developed in-house and/or built by commercial software vendors. Each component and all software code stacks must operate correctly with each other. Integration between the software stacks is difficult. If any of the component pieces of code contains vulnerabilities or is configured incorrectly, the web application is vulnerable to an attack.

Fixing vulnerabilities at each layer requires separate solutions for each layer. Not only is this tedious, but fixing problems by making manual or software changes has the potential to introduce more issues. Each change requires a complete Quality Assurance cycle of the entire web application system. Additionally, because no single group within a company owns all of the web components and each component requires a different type of security, difficult inter-departmental coordination is required to secure the application manually.



*Figure 1: A web application has multiple integrated layers*

### All you need is a web browser

An attacker can target different layers, components and software stacks of a web application. The potential for a security breach exists in each layer. An attacker need not have direct access to manipulate the software code because the browser itself serves as an entry point into the web application and its dataflow.

A hacker making a web request with a browser starts a flow of data with malicious intent. In a web attack, the input is designed to cause an adverse reaction on one of the many components of a web application as shown in figure 1. Each layer must handle and manage the request properly to avoid becoming a potential breach or attack point.

### Input Validation Attacks

The same type of web attack may be described in many different ways. Most web attacks share the common method of injecting malicious code into the application. The goal is to exploit the application's inability to validate input.

Web attacks are given labels and names based on various attributes of the attack. The security industry assigns attack names based on the type of malicious code injected, the result of the attack, or the web component targeted for attack. For example:

- SQL injection attacks inject or alter existing SQL commands.
- OS Command Injection is an attempt to execute operating system commands on one of the application components.
- Meta character attack is a generic name for an attack that injects Meta character-based malicious code into applications.
- A path traversal attack is named after the result of an application not properly validating coding characters used to describe paths.

All of these attacks are designed to inject malicious code into a web application through one of many input vehicles.

### What is Malicious Code?

An attacker targets a web application with code specifically designed to trigger an adverse reaction in a web component or software stack. The attack input and reaction depend on the component and software stacks. All web languages such as HTML, PERL and CGI can be attacked. Web servers such as Apache and IIS have vulnerabilities. The most commonly-used databases today, MS SQL Server and Oracle, are subject to SQL injection attacks. And every operating system has system calls that can be exploited.

The goal of malicious code is to execute software commands on a component of the web application. Nearly every programming language allows the use of so called "system-commands." Many applications make use of this type of functionality. Accordingly, malicious input uses lines of code specific to the web application being attacked. Malicious input will have characteristics or key words that are typical of attacks directed at that component of the web application.

### Examples of input that should be regarded as malicious code

<b>Perl</b> open() (the #1) sysopen() glob()	<b>PHP</b> require() include() eval()
<b>C,C++</b> system() exec**()	<b>SQL</b> xp_cmdshell, sp_adduser into outfile

In addition to the commands described above, attacks can use Meta characters to launch malicious code. Meta characters are non-printable and printable characters that affect the behavior of programming language commands, operating system commands, individual program procedures and database queries. Meta characters are input characters not interpreted as data. They are typically injected into the web application as a method of attack. The actual characters used in an attack depend on the web components (operating systems, databases) and software stacks (programming languages) on the target application.

One of the most pervasive Meta character problems is a result of the standard Unix-like command shell (stored in `/bin/sh`), which interprets a number of characters specially. If these characters are sent to the shell, then their special interpretation will be used unless they are escaped. Hackers use this vulnerability to break programs.

Partial List of Meta Characters	
<code>;</code>	Semicolons for additional command execution
<code> </code>	Pipes for command execution
<code>&amp;</code>	Ampersands for command execution
<code>[ x20 ]</code>	Spaces for faking URLs and other names (especial in URLs), fake-mails and changing file-content
<code>[ x1b ]</code>	Escape: OS dependent
<code>[ x08 ]</code>	Backspace: OS dependant (for faking logfiles, changing file-content)
<code>&lt;&gt;</code>	LTs and GTs for file operations
<code>?</code>	programming/scripting-language related
<code>\$</code>	programming/scripting-language related
<code>@</code>	programming/scripting-language related
<code>( [ ] )</code>	programming/scripting/regex and language-related

## Injection Vehicles

HTTP is the ubiquitous protocol in use on the Internet. Web browsers and web servers must communicate in order to exchange information, and web attacks are limited to using the HTTP protocol to inject malicious code into the application. More precisely, a hacker can inject malicious code into a web application through URL parameters, HTML forms, or headers

### URLs

URL components include the protocol, server, path and query strings. Malicious code is most often passed or injected into the application in the path or query string component.

The structure of the URL is illustrated below:

```
http://www.netcontinuum.com/order/buy.asp?item=NC-1000&pmt=cash
```

- 1) **http://** denotes the application layer protocol.
- 2) **www.netcontinuum.com** is the server DNS name.
- 3) **/order/buy.asp** is the directory path to the resource of the resources being requested.
- 4) **item=NC-1000&pmt=cash** is the query string for parameters passed to an application that dynamically generates output.

### Forms

Almost all interactive web applications use HTML forms, which provide a mechanism for injecting malicious code into a web application. Chances are you have encountered a form when using a search engine, checking your web based email, entering account information, and so on. Forms are submitted using either a GET or a POST method. Each form has a submit button and a name of the input elements. If the submission is via GET, the input parameters are passed in the URL query string. If it is via POST the parameters are passed in the data portion of the request.

### Headers

The request header fields allow the client to pass additional information about the request, and about the client itself, to the server. The server parses these fields and uses the information as needed. Each header field can be a mechanism for inserting malicious code.

### Cookies

Cookies represent a frequently-used mechanism for inserting malicious code into an application. Cookies are a special header extension defined in RFC 2109 as an HTTP State Management Mechanism. Cookies can potentially be passed through all layers of the web application component system, thus providing a robust and flexible mechanism for attacking web applications.

In addition to being a critical state mechanism, cookies often contain single sign-on (SSO) tokens, session tokens and other confidential information that might be used by any component of the web application system. Accordingly, any component of the application is potentially vulnerable to attacks launched with cookies.

Hackers can easily view, modify and create cookies containing malicious code. This is called *cookie poisoning*, and represents a common mechanism for inserting malicious code into an application. Both persistent and non-persistent cookies, secure or insecure, can be a vehicle for delivering malicious code to the server with URL requests. There is a popular misconception that non-persistent cookies cannot be modified but this is not true. Common tools like Winhex are freely available for this end.

For example, a cookie might look like the following:

```
Cookie: lang=en-us; ADMIN=no; y=1 ; time=10:30GMT ;
```

The attacker can simply modify the cookie to;

```
Cookie: lang=en-us; ADMIN=(Malicious Code) yes; y=1 ; time=12:30GMT
```

### **A wide range of potential input validation attacks**

The following attacks are related in that a hacker injects malicious code into the application. The input is best described as software code, system commands and meta-characters. The vehicle for injection into web application is an HTML form, header or URL parameter.

The following list of well-known attacks is not comprehensive. Each attack is unique depending on the target application. When analyzing specific web attacks, the security expert will find ambiguity and overlap between attack types.

*SQL Injection Attacks* - Exploiting nonexistent or poorly designed input validation routines, SQL injection attacks represent a serious threat to any database-driven site. In a direct SQL command injection, an attacker creates or alters existing SQL commands to gain access to unintended data or even the ability to execute system-level commands on the host.

*Direct OS or System Command Injection* - A hacker can execute operating system commands by injecting them via HTML forms, cookies or a URL parameter. Using this type of attack the hacker is able to execute system-level functions such as removing and copying files, sending emails, and calling operating system tools to modify the application's input and output.

*Meta-characters* - Hackers change the behavior of a web application by inserting Meta characters into URL-encoded parameters within query strings, headers or forms. The risk depends on the operating system, programming languages and workflow of the affected application.

*Cross-Site Scripting* - This type of attack inserts malicious code into a web application. It has no effect on the web application. Instead, it uses the web application as a mechanism to transport an attack to a client browser. The attack creates a situation in which the web application can be used as a launch vehicle for an attack on the client.

A successful attack may disclose the a client's or customer's session token, attack the local machine, or spoof content to fool the user. Cross-site scripting attacks cause users or customers of the target website to execute malicious scripts unintentionally, often violating trust between the user and host website which is, presumably, a trusted site.

*Path Traversal Attacks* - Path traversal vulnerabilities allow a hacker to execute commands or view data outside of the intended target path. The file system may be accessed via numerous commands and functions used by the web application programming language. Path traversal attacks are normally carried out via unchecked URL input parameters, cookies and HTTP request headers.

### **Thwarting Input Validation Attacks with a Web Security Gateway**

A web application is made up of many unique components; a successful security solution needs to treat the entire web application (all components) as a single entity in order to be effective. Only by deploying a gateway at the perimeter of the application can a holistic security implementation be achieved.

If a request contains malicious code destined for any web application component, a perimeter gateway security system can invalidate the request and block the attack. This approach creates a single point of protection, ensuring that individual back-end protection solutions are redundant. It also defends across the entire spectrum of disruptive or hostile activity, providing a defense-in-depth security posture against new and unknown application-layer attacks.

The NetContinuum NC-1000 web security gateway does exactly this. It checks the data at the application entry point and verifies that it matches the expected input before allowing it to be passed to the web application. It checks user input and injection vehicles against expected norms before allowing it to be processed, protecting against both known and unknown attacks. It denies access to the underlying operating system environments for each component of the web application. Decomposing all incoming HTTP requests, it ensures that the requests are both valid and correctly formed and then blocks patterns already known to subvert web servers and applications, as well as blocking any unexpected or non-standard formatting.

#### **Blocking malicious input with the NC-1000**

Malicious code is most often passed or injected into the application in the URL path or query string components of the URL. Excessively long URLs or an excessively long URL component is also a strong indicator that a URL carries a malicious payload.

The NC-1000 blocks attempts to attack an application through URLs. With a full RFC-compliant TCP and HTTP handshake, the NC-1000 fully parses and inspects all components of the URL. It then searches the URL for malicious code in the form of keywords and meta-characters. The settings can be tailored and set to optimize security on the website. When it finds the malicious code, it blocks the request, protecting the application.

Below is an example of an SQL injection attack that would be blocked by the NC-1000. Notice the query portion of the URL contains a “cmd.exe” statement that gives the hacker access to the server.

Original MS SQL-Query:

```
SELECT id,name FROM products WHERE id LIKE '%$INPUT[prod]%' ;
```

Deformed HTTP-Request:

Excessively long URLs or URL components are indications of malicious code. Accordingly, the NC-1000 implements a host of functionality that limits the length of the URLs. URL length limits can be set on a per domain basis, which enables more accurate and powerful security for a specific application. The default setting for the NC-1000 is characters in the URL, which includes the path and query string. This value is same for all methods.

The NC-1000 allows administrators to set URL application security policy manually. Administrators can create attack signatures so that known or unknown vulnerabilities with malicious content can be dropped. An administrator can set “deny unless specifically allowed” policy for URLs. Similar to a network layer firewall where all IP: port combinations are blocked unless specifically opened, the NC-1000 can create policy whereby specific URL’s and domains cannot be accessed unless designated by the administrator.

#### **Controlling URL Access**

Forceful browsing is the process of directly requesting pages that should only be accessed through jumping from legal URLs. Forceful browsing is often used to bypass authentication. Although forceful browsing is not in itself an input attack, it is frequently used in conjunction with an input validation attack. Vulnerable files, hidden unreferenced files, backup files, and temp files can all be attacked with malicious code, even though the application deployment team never meant for those files to be exposed.

For example, website administrators may leave sample files or default installation files on the web server. When the web content is published, these vulnerable files remain accessible although are not referenced by any HTML in the web. Attackers make educated guesses about the URL and try to attack these files. The process of building HTML and ASP pages can leave behind vulnerable directories. Backup files are also dangerous as many developers embed things into development HTML that they should eliminate before applications go into production.

The NC-1000’s *Dynamic URL Allow Lists* are learned or generated on-the-fly except for initial landing pages. Start URLs/Landing page URLs are configured. These consist of the home page, main product page and other URLs that customers or clients would initially use to begin the application.

The NC-1000 creates valid URLs for the server group it is protecting by parsing the response pages for embedded hyperlinks. Clients make HTTP requests for the application. The application serves up

responses. The NC-1000 parses the responses searching for embedded hyperlinks. It puts any hyperlinks it finds on the Dynamic URL Allow Lists.

All incoming requests are matched against the allow list. If a request matches the URLs in the allow lists, it is allowed to pass the NC-1000. If it does not match the listed URLs, the NC-1000 redirects the request to one of the start pages. Thus, files that are not referenced from any valid HTML page are not accessible. This blocks malicious URL input and prevents hackers from attacking vulnerable URLs with malicious code.

#### **Preventing malicious input in headers**

The request header fields are another mechanism for inserting malicious code. The NC-1000 inspects and blocks requests headers that carry malicious code. Similar to URL protection, the NC-1000 inspects all headers for keywords and meta characters and blocks those request from reaching the applications. Security policy can also be set to block or apply other policies to requests with headers matching specified criteria. This enables the administrator to create a strong security policy tailored to the web application being delivered.

#### **Restricting methods**

The NC-1000 can be configured with a security policy to only allow requests with the common methods for HTTP 1.1. Common methods include GET, POST and DELETE. All other requests with invalid methods are denied. A typical configuration may exclude "PUT," which an attacker might use to send invalid input. This "deny unless allowed" method policy also stops attackers from attacking web services and other potential holes in the web application due to services on servers such as SOAP or WebDAV. Even though a company might not be building web service-enabled applications, those services are embedded into applications. Security is enhanced if policies are in place to block SOAP calls over HTTP when legitimate SOAP services were never deployed.

#### **Preventing cookie poisoning**

NetContinuum prevents cookie poisoning by encrypting and digitally signing the cookie generated by the application the NC-1000 is protecting. Encrypting the cookie prevents a hacker from viewing its contents, making it more difficult to create malicious code successfully. The NC-1000 applies a digital signature to the cookie, effectively preventing the malicious user from modifying the cookie's contents.

#### **Protecting forms submitted with GET**

Many applications use HTML forms, another popular mechanism for injecting malicious code into a web application. Forms are submitted via a GET or a POST method. Each form has a submit button and a name of the input elements. If the submission uses GET, the input parameters are passed in the URL query string. The NC-1000 inspects and blocks requests that carrying a form submittal via a GET with malicious code. Similar to URL protection, the NC-1000 inspects all GET-submitted forms' keywords and meta characters and blocks those request from reaching the applications.

#### **Thwarting disguised input validation attacks**

Preventing an application layer attack depends on string matching. Incorrect string matching creates of security holes.

The NC-1000 matches the URL requested with the security policy, then applies the correct policy (such as allow or deny). String matching is performed by comparing two strings, byte for byte. However, HTTP has multiple character representations for the same string. Binary comparison does not work if the strings do not use the same encoding.

Solving the string matching problem involves normalization, which in a nutshell means bringing the two strings to be compared to a common, canonical encoding prior to performing the security string match.

Attackers use numerous techniques to disguise URL input validation attacks, hoping to thwart successful string matching and bypass security devices. The term *polymorphic URL* has been coined to describe URLs that relate to the same resource but are written or disguised in many ways. These attacks are particularly effective at bypassing signature-based Intrusion Detection & Prevention Systems simply because the attack is coded slightly differently than the signatures in their databases.

By translating or normalizing the following encoding or disguising techniques, the NC-1000 identifies and prevents a large proportion of the attacks.

*String and fragmentation:* The NC-1000 easily defeats attempts to disguise attacks with common TCP string and fragmentation techniques. It accomplishes this with a full RFC-compliant handshake and reassembly of the packet.

*Unicode:* Unicode was developed to allow a Universal Character Set (UCS) that encompasses most of the world's writing systems. Unicode Encoding stores characters with multiple bytes. Wherever input data is allowed, data can be entered using Unicode to disguise malicious code and permit a variety of attacks. The NC-1000 has a Unicode translation engine. It decodes Unicode and blocks attacks disguised with Unicode.

*URL encoding:* URL-encoding is a technique defined in the URL/URI specifications for mapping 8-bit data to the subset of the US-ASCII character set allowed in a URL/URI. Without proper validation, URL encoding obscures text or attack sequences within the URL field for web transmission-based attacks. This is an effective way to probe and send malicious input to web servers without detection. By default the NC-1000 translates URL encoding and blocks attacks disguised with URL encoding.

*Double URL encoding:* Attackers may disguise attacks by encoding already-encoded URLs, effectively creating "double" encoded attacks. The NC-1000 translates double-encoded URLs, blocking malicious code.

*SSL/TLS encryption:* SSL encryption is another method for bypassing intrusion detection and prevention systems. SSL encrypts the URL so that it can bypass security devices without detection. The NC-1000 fully decrypts SSL and blocks malicious code disguised with SSL.

*Polymorphic Shell Attacks:* Recently new tools have emerged that attempt to obfuscate the detection of processor instruction sets using encoding techniques. The NC-1000 Auto-Correct feature recognizes these

encoding techniques, and then normalizes and translates encoded or disguised URLs. The application-layer security policy features can then block attempts to inject malicious URLs.

## Summary

The NC-1000 web security gateway prevents a class of web attacks caused by hackers injecting malicious code into web applications. These attacks include SQL injection, Cross Site Scripting, OS Command Injection, Path Traversal and Meta Character attacks.

The NC-1000 applies security policy at the website perimeter, enabling fine-grained access control. Administrators can define distinct security policies for different web applications, creating an environment in which all access is denied unless specifically allowed. By normalizing headers, checking for known and suspicious input, and restricting unwanted URL access, the NC-1000 defends against a broad spectrum of input validation attacks. This proactive security approach helps protect web applications from both known, named attacks and new or undiscovered attacks.