# CWE Detail – CWE-84

## Description

The web application improperly neutralizes user-controlled input for executable script disguised with URI encodings.

## Extended Description

N/A

## Threat-Mapped Scoring

Score: 0.0

Priority: Unclassified

## Observed Examples (CVEs)

**•** CVE-2005-0563: Cross-site scripting (XSS) vulnerability in Microsoft Outlook Web Access (OWA) component in Exchange Server 5.5 allows remote attackers to inject arbitrary web script or HTML via an email message with an encoded javascript: URL ("jav&#X41sc&#0010;ript:") in an IMG tag.

**•** CVE-2005-2276: Cross-site scripting (XSS) vulnerability in Novell Groupwise WebAccess 6.5 before July 11, 2005 allows remote attackers to inject arbitrary web script or HTML via an e-mail message with an encoded javascript URI (e.g. "j&#X41vascript" in an IMG tag).

**•** CVE-2005-0692: Encoded script within BBcode IMG tag.

**•** CVE-2002-0117: Encoded "javascript" in IMG tag.

**•** CVE-2002-0118: Encoded "javascript" in IMG tag.

## Modes of Introduction

**•** Implementation: N/A

## Common Consequences

**•** Impact: Unexpected State — Notes:

## Potential Mitigations

**•** Implementation: Resolve all URIs to absolute or canonical representations before processing. (Effectiveness: N/A)

**•** Implementation: Carefully check each input parameter against a rigorous positive specification (allowlist) defining the specific characters and format allowed. All input should be neutralized, not just parameters that the user is supposed to specify, but all data in the request, including tag attributes, hidden fields, cookies, headers, the URL itself, and so forth. A common mistake that leads to continuing XSS vulnerabilities is to validate only fields that are expected to be redisplayed by the site. We often encounter data from the request that is reflected by the application server or the application that the development team did not anticipate. Also, a field that is not currently reflected may be used by a future developer. Therefore, validating ALL parts of the HTTP request is recommended. (Effectiveness: N/A)

**•** Implementation: Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component. The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks. (Effectiveness: N/A)

**•** Implementation: With Struts, write all data from form beans with the bean's filter attribute set to true. (Effectiveness: N/A)

**•** Implementation: To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set. (Effectiveness: Defense in Depth)

## Applicable Platforms

**•** None (Class: Not Language-Specific, Prevalence: Undetermined)