# CWE Detail – CWE-88

## Description

The product constructs a string for a command to be executed by a separate component  
in another control sphere, but it does not properly delimit the  
intended arguments, options, or switches within that command string.

## Extended Description

When creating commands using interpolation into a string, developers may assume that only the arguments/options that they specify will be processed. This assumption may be even stronger when the programmer has encoded the command in a way that prevents separate commands from being provided maliciously, e.g. in the case of shell metacharacters. When constructing the command, the developer may use whitespace or other delimiters that are required to separate arguments when the command. However, if an attacker can provide an untrusted input that contains argument-separating delimiters, then the resulting command will have more arguments than intended by the developer. The attacker may then be able to change the behavior of the command. Depending on the functionality supported by the extraneous arguments, this may have security-relevant consequences.

## Threat-Mapped Scoring

Score: 0.0

Priority: Unclassified

## Observed Examples (CVEs)

**•** CVE-2022-36069: Python-based dependency management tool avoids OS command injection when generating Git commands but allows injection of optional arguments with input beginning with a dash (CWE-88), potentially allowing for code execution.

**•** CVE-1999-0113: Canonical Example - "-froot" argument is passed on to another program, where the "-f" causes execution as user "root"

**•** CVE-2001-0150: Web browser executes Telnet sessions using command line arguments that are specified by the web site, which could allow remote attackers to execute arbitrary commands.

**•** CVE-2001-0667: Web browser allows remote attackers to execute commands by spawning Telnet with a log file option on the command line and writing arbitrary code into an executable file which is later executed.

**•** CVE-2002-0985: Argument injection vulnerability in the mail function for PHP may allow attackers to bypass safe mode restrictions and modify command line arguments to the MTA (e.g. sendmail) possibly executing commands.

**•** CVE-2003-0907: Help and Support center in windows does not properly validate HCP URLs, which allows remote attackers to execute arbitrary code via quotation marks in an "hcp://" URL.

**•** CVE-2004-0121: Mail client does not sufficiently filter parameters of mailto: URLs when using them as arguments to mail executable, which allows remote attackers to execute arbitrary programs.

**•** CVE-2004-0473: Web browser doesn't filter "-" when invoking various commands, allowing command-line switches to be specified.

**•** CVE-2004-0480: Mail client allows remote attackers to execute arbitrary code via a URI that uses a UNC network share pathname to provide an alternate configuration file.

**•** CVE-2004-0489: SSH URI handler for web browser allows remote attackers to execute arbitrary code or conduct port forwarding via the a command line option.

**•** CVE-2004-0411: Web browser doesn't filter "-" when invoking various commands, allowing command-line switches to be specified.

**•** CVE-2005-4699: Argument injection vulnerability in TellMe 1.2 and earlier allows remote attackers to modify command line arguments for the Whois program and obtain sensitive information via "--" style options in the q\_Host parameter.

**•** CVE-2006-1865: Beagle before 0.2.5 can produce certain insecure command lines to launch external helper applications while indexing, which allows attackers to execute arbitrary commands. NOTE: it is not immediately clear whether this issue involves argument injection, shell metacharacters, or other issues.

**•** CVE-2006-2056: Argument injection vulnerability in Internet Explorer 6 for Windows XP SP2 allows user-assisted remote attackers to modify command line arguments to an invoked mail client via " (double quote) characters in a mailto: scheme handler, as demonstrated by launching Microsoft Outlook with an arbitrary filename as an attachment. NOTE: it is not clear whether this issue is implementation-specific or a problem in the Microsoft API.

**•** CVE-2006-2057: Argument injection vulnerability in Mozilla Firefox 1.0.6 allows user-assisted remote attackers to modify command line arguments to an invoked mail client via " (double quote) characters in a mailto: scheme handler, as demonstrated by launching Microsoft Outlook with an arbitrary filename as an attachment. NOTE: it is not clear whether this issue is implementation-specific or a problem in the Microsoft API.

**•** CVE-2006-2058: Argument injection vulnerability in Avant Browser 10.1 Build 17 allows user-assisted remote attackers to modify command line arguments to an invoked mail client via " (double quote) characters in a mailto: scheme handler, as demonstrated by launching Microsoft Outlook with an arbitrary filename as an attachment. NOTE: it is not clear whether this issue is implementation-specific or a problem in the Microsoft API.

**•** CVE-2006-2312: Argument injection vulnerability in the URI handler in Skype 2.0.\*.104 and 2.5.\*.0 through 2.5.\*.78 for Windows allows remote authorized attackers to download arbitrary files via a URL that contains certain command-line switches.

**•** CVE-2006-3015: Argument injection vulnerability in WinSCP 3.8.1 build 328 allows remote attackers to upload or download arbitrary files via encoded spaces and double-quote characters in a scp or sftp URI.

**•** CVE-2006-4692: Argument injection vulnerability in the Windows Object Packager (packager.exe) in Microsoft Windows XP SP1 and SP2 and Server 2003 SP1 and earlier allows remote user-assisted attackers to execute arbitrary commands via a crafted file with a "/" (slash) character in the filename of the Command Line property, followed by a valid file extension, which causes the command before the slash to be executed, aka "Object Packager Dialogue Spoofing Vulnerability."

**•** CVE-2006-6597: Argument injection vulnerability in HyperAccess 8.4 allows user-assisted remote attackers to execute arbitrary vbscript and commands via the /r option in a telnet:// URI, which is configured to use hawin32.exe.

**•** CVE-2007-0882: Argument injection vulnerability in the telnet daemon (in.telnetd) in Solaris 10 and 11 (SunOS 5.10 and 5.11) misinterprets certain client "-f" sequences as valid requests for the login program to skip authentication, which allows remote attackers to log into certain accounts, as demonstrated by the bin account.

**•** CVE-2001-1246: Language interpreter's mail function accepts another argument that is concatenated to a string used in a dangerous popen() call. Since there is no neutralization of this argument, both OS Command Injection (CWE-78) and Argument Injection (CWE-88) are possible.

**•** CVE-2019-13475: Argument injection allows execution of arbitrary commands by injecting a "-exec" option, which is executed by the command.

**•** CVE-2016-10033: Argument injection in mail-processing function allows writing unxpected files and executing programs using tecnically-valid email addresses that insert "-o" and "-X" switches. (KEV)

## Related Attack Patterns (CAPEC)

* CAPEC-137
* CAPEC-174
* CAPEC-41
* CAPEC-460
* CAPEC-88

## Modes of Introduction

**•** Implementation: REALIZATION: This weakness is caused during implementation of an architectural security tactic.

## Common Consequences

**•** Impact: Execute Unauthorized Code or Commands, Alter Execution Logic, Read Application Data, Modify Application Data — Notes: An attacker could include arguments that allow unintended commands or code to be executed, allow sensitive data to be read or modified or could cause other unintended behavior.

## Potential Mitigations

**•** Implementation: Where possible, avoid building a single string that contains the command and its arguments. Some languages or frameworks have functions that support specifying independent arguments, e.g. as an array, which is used to automatically perform the appropriate quoting or escaping while building the command. For example, in PHP, escapeshellarg() can be used to escape a single argument to system(), or exec() can be called with an array of arguments. In C, code can often be refactored from using system() - which accepts a single string - to using exec(), which requires separate function arguments for each parameter. (Effectiveness: High)

**•** Architecture and Design: Understand all the potential areas where untrusted inputs can enter your product: parameters or arguments, cookies, anything read from the network, environment variables, request headers as well as content, URL components, e-mail, files, databases, and any external systems that provide data to the application. Perform input validation at well-defined interfaces. (Effectiveness: N/A)

**•** Implementation: Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. (Effectiveness: N/A)

**•** Implementation: Directly convert your input type into the expected data type, such as using a conversion function that translates a string into a number. After converting to the expected data type, ensure that the input's values fall within the expected range of allowable values and that multi-field consistencies are maintained. (Effectiveness: N/A)

**•** Implementation: Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180, CWE-181). Make sure that your application does not inadvertently decode the same input twice (CWE-174). Such errors could be used to bypass allowlist schemes by introducing dangerous inputs after they have been checked. Use libraries such as the OWASP ESAPI Canonicalization control. Consider performing repeated canonicalization until your input does not change any more. This will avoid double-decoding and similar scenarios, but it might inadvertently modify inputs that are allowed to contain properly-encoded dangerous content. (Effectiveness: N/A)

**•** Implementation: When exchanging data between components, ensure that both components are using the same character encoding. Ensure that the proper encoding is applied at each interface. Explicitly set the encoding you are using whenever the protocol allows you to do so. (Effectiveness: N/A)

**•** Implementation: When your application combines data from multiple sources, perform the validation after the sources have been combined. The individual data elements may pass the validation step but violate the intended restrictions after they have been combined. (Effectiveness: N/A)

**•** Testing: Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible. (Effectiveness: N/A)

**•** Testing: Use dynamic tools and techniques that interact with the product using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The product's operation may slow down, but it should not become unstable, crash, or generate incorrect results. (Effectiveness: N/A)

## Applicable Platforms

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

**•** PHP (Class: None, Prevalence: Often)

## Demonstrative Examples

**•** However, validate\_name() allows  
 filenames that begin with a "-". An adversary could  
 supply a filename like "-aR", producing the "ls -l -aR"  
 command (CWE-88), thereby getting a full recursive  
 listing of the entire directory and all of its  
 sub-directories. There are a couple possible mitigations for this  
 weakness. One would be to refactor the code to avoid  
 using system() altogether, instead relying on internal  
 functions. Another option could be to add a "--" argument  
 to the ls command, such as "ls -l --", so that any  
 remaining arguments are treated as filenames, causing  
 any leading "-" to be treated as part of a filename  
 instead of another option. Another fix might be to change the regular expression used in validate\_name to force the first character of the filename to be a letter or number, such as:

**•** The program calls PHP's mail() function to compose and send mail. The fifth argument to mail() is a set of parameters. The program intends to provide a "-fSENDER" parameter, where SENDER is expected to be a well-formed email address. The program has already validated the e-mail address before invoking mail(), but there is a lot of flexibility in what constitutes a well-formed email address, including whitespace. With some additional allowed characters to perform some escaping, the adversary can specify an additional "-o" argument (listing an output file) and a "-X" argument (giving a program to execute). Additional details for this kind of exploit are in [REF-1250].

## Notes

**•** Relationship: At one layer of abstraction, this can overlap other weaknesses that have whitespace problems, e.g. injection of javascript into attributes of HTML tags.