# CWE Detail – CWE-1333

## Description

The product uses a regular expression with an inefficient, possibly exponential worst-case computational complexity that consumes excessive CPU cycles.

## Extended Description

Some regular expression engines have a feature called "backtracking". If the token cannot match, the engine "backtracks" to a position that may result in a different token that can match. Backtracking becomes a weakness if all of these conditions are met: The number of possible backtracking attempts are exponential relative to the length of the input. The input can fail to match the regular expression. The input can be long enough. Attackers can create crafted inputs that
 intentionally cause the regular expression to use
 excessive backtracking in a way that causes the CPU
 consumption to spike.

## Threat-Mapped Scoring

Score: 0.0

Priority: Unclassified

## Observed Examples (CVEs)

**•** CVE-2020-5243: server allows ReDOS with crafted User-Agent strings, due to overlapping capture groups that cause excessive backtracking.

**•** CVE-2021-21317: npm package for user-agent parser prone to ReDoS due to overlapping capture groups

**•** CVE-2019-16215: Markdown parser uses inefficient regex when processing a message, allowing users to cause CPU consumption and delay preventing processing of other messages.

**•** CVE-2019-6785: Long string in a version control product allows DoS due to an inefficient regex.

**•** CVE-2019-12041: Javascript code allows ReDoS via a long string due to excessive backtracking.

**•** CVE-2015-8315: ReDoS when parsing time.

**•** CVE-2015-8854: ReDoS when parsing documents.

**•** CVE-2017-16021: ReDoS when validating URL.

## Related Attack Patterns (CAPEC)

* CAPEC-492

## Modes of Introduction

**•** Implementation: A RegEx can be easy to create and read using unbounded matching characters, but the programmer might not consider the risk of excessive backtracking.

## Common Consequences

**•** Impact: DoS: Resource Consumption (CPU) — Notes:

## Potential Mitigations

**•** Architecture and Design: Use regular expressions that do not support backtracking, e.g. by removing nested quantifiers. (Effectiveness: High)

**•** System Configuration: Set backtracking limits in the configuration of the regular expression implementation, such as PHP's pcre.backtrack\_limit. Also consider limits on execution time for the process. (Effectiveness: Moderate)

**•** Implementation: Do not use regular expressions with untrusted input. If regular expressions must be used, avoid using backtracking in the expression. (Effectiveness: High)

**•** Implementation: Limit the length of the input that the regular expression will process. (Effectiveness: Moderate)

## Applicable Platforms

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

## Demonstrative Examples

**•** The regular expression has a vulnerable backtracking clause inside (\w+\s?)\*$ which can be triggered to cause a Denial of Service by processing particular phrases. To fix the backtracking problem, backtracking is removed with the ?= portion of the expression which changes it to a lookahead and the \2 which prevents the backtracking. The modified example is:

**•** The regular expression has a vulnerable backtracking clause inside (\w+\s?)\*$ which can be triggered to cause a Denial of Service by processing particular phrases. To fix the backtracking problem, backtracking is removed with the ?= portion of the expression which changes it to a lookahead and the \2 which prevents the backtracking. The modified example is: