# attackercan / regexp-security-cheatsheet



*No description, website, or topics provided.*

65 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Find file Clone or download

attackercan new markdown parser Latest commit ec7d510 on Aug 21, 2017

| | | |
|---|---|---|
| RegexpSecurityParser | Upd: now counting *reg_* (ereg_replace etc) | 2 years ago |
| README.md | new markdown parser | 9 months ago |

README.md

# Regexp Security Cheatsheet

Research was done to find "weak places" in regular expressions of Web Application Firewalls (WAFs).
Repository contains SAST, which can help you to find security vulnerabilities in custom regular expressions in own projects.
Contribution is highly welcomed.

## High severity issues:

| # | Requirement | Vulnerable regex example | Bypass example |
|---|---|---|---|
| 1 | Regexp should avoid using `^` (alternative: `\A`) and `$` (alternative: `\Z`) symbols, which are metacharacters for start and end of a string. It is possible to bypass regex by inserting any symbol in front or after regexp. | `(^a\|a$)` | `%20a%20` |
| 2 | Regexp should be case-insensitive: `(?i:` or `/regex/i`. It is possible to bypass regex using upper or lower cases in words. Modsecurity transformation commands (which are applied on string before regex pattern is applied) can also be included in tests to cover more regexps. | `http` | `hTtP` |

| # | Requirement | Vulnerable regex example | Bypass example |
|---|---|---|---|
| 3 | In case modifier `/m` is not (globally) specified, regexp should avoid using dot `.` symbol, which means every symbol except newline (`\n`). It is possible to bypass regex using newline injection. | `a.*b` | `a%0Ab` |
| 4 | Regexp should not be vulnerable to ReDoS. OWASP ReDoS article 1. Find various evil patterns. 2. Generate evil string using e.g. “SDL Regex Fuzzer” | `(a+)+` | `aaaaaaaaaaaaaaaaaaaa!` |
| 5 | Number of repetitions of set or group `{}` should be carefully used, as one can bypass such limitation by lowering or increasing specified numbers. | `a{1,5}` | `aaaaaa (6 times)` |
| 6 | Nonstandard ranges (almost everything except a-z, 0-9, a-f, etc) | `[A-z] = [a-zA-Z] + [\]^_`` | `aaa[\]^_`aaa` |
| 7 | Regexp should only use plus “`+`” metacharacter in places where it is necessary, as it means “one or more”. Alternative metacharacter star “`*`”, which means “zero or more” is generally preferred. | `a'\s+\d` | `a'5` |

| # | Requirement | Vulnerable regex example | Bypass example |
|---|---|---|---|
| 8 | Usage of newline wildcards should be reasonable. `\r\n` characters can often be bypassed by either substitution, or by using newline alternative `\v`, `\f` and others. Wildcard `\b` has different meanings while using it in square brackets (“backspace”) and in plain regex (“word boundary”) - RegexLib | `a[^\n]*$` | `a\n`? `a\r`? |
| 9 | Regexp should be applied to right scope of inputs: `Cookies names and values`, `Argument names and values`, `Header names and values`, `Files argument names and content`. Modsecurity: `grep -oP 'SecRule(.*?)"' -n` Other WAFs: manual observation. | Argument values | Cookie names and values |
| 10 | Regular expression writers should be careful while using only whitespace character ( `%20` ) as separators. Rule can be bypassed e.g. with newline character, tabulation, by skipping whitespace, or alternatives. | `a\s(not[whitespace]|and)\sb` | `a not b` |
| 11 | Nonstandard combinations of operators | `a||b` | `any_string` |
| 12 | Special cases: whitespaces before operators | `(a |b)c` | `ac` |
| 13 | Usage of wrong syntax in POSIX character classes | `a[digit]b` | `aab` |

| # | Requirement | Vulnerable regex example | Bypass example |
|---|---|---|---|
| 14 | Opposite usage of brackets [], () and {} | `[SYSTEM\|PUBLIC]` or `(a-z123)` | `SYSTEM` or `abcdef` |

## Medium severity issues (non-expected behaviour: manual observation needed):

| # | Requirement | Vulnerable regex example | Bypass example |
|---|---|---|---|
| 15 | Check backlinks, and bear in mind that `\11` can be backlink -OR- 0x09 | `(\d{1})=\1` | `1!=2` |
| 16 | Unsafe usage of comments | `a(?#some comment about wildcards:\)(\w*)b` | `afffb` |
| 17 | Excessive usage of metacharacters in [] | `[\w+]` | |
| 18 | Rarely used wildcards. All wildcards except popular: A,Z,b,r,n,t,wW,sS,dD,u,x | `\a = 0x07; \e = 0x1B; \R = \r\|\n\|\r\n; \xXX = 0xXX; \ddd = 0oddd; \cX, \x{XXXX}, \H, \V, \G` | |
| 19 | Excessive escaping, e.g. escaping symbol which is not a wildcard | `\q` | |
| 20 | Unsafe usage of recursion, IF statements, etc | `(?R`, `(?(id)true\|false)`, ... | |
| 21 | Unsafe usage of ranges | `[\0-9]` = `\0\1\2\3...$%&'...789` | |

**Experimental rules (probably to be removed):**

| # | Requirement | Vulnerable regex example | Bypass example |
|---|---|---|---|

| # | Requirement | Vulnerable regex example | Bypass example |
|---|---|---|---|
| X | Greediness of regular expressions should be considered. Highlight of this topic is well done in Chapter 9 of Jan Goyvaert's tutorial. While greediness itself does not create bypasses, bad implementation of regexp Greediness can raise False Positive rate. This can cause excessive log-file flooding, forcing vulnerable rule or even whole WAF to be switched off. | | |
| X | Best Practice from slides of Ivan Novikov: Modsecurity should avoid using t:base64Decode function (t:base64DecodeExt instead). | `t:base64Decode` | `detected=bypassed` |

Vladimir Ivanov @httpsonly