Regular Expression Guidelines and Performance Considerations for URL Filtering

Contents

Introduction

This document describes the guidelines and performance considerations for using regular expressions in URL filtering with the UTD engine. URL filtering in the UTD engine uses the PCRE2 regular expression library.

Contributed by Eugene Khabarov, Cisco Engineering.

Prerequisites

Requirements

Cisco recommends that you have knowledge of these topics:

- Regular expressions (regex) syntax
- URL Filtering concepts

- Unified Threat Defense (UTD) configuration
- HTTPS/HTTP protocol differences

Components Used

This document is not restricted to specific software and hardware versions.

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. If your network is live, ensure that you understand the potential impact of any command.

Background Information

While PCRE2 is powerful, certain complex or 'greedy' expressions can cause excessive backtracking and can hit internal limits in the regex engine. When this occurs, a pattern can take too much time to process and ultimately be treated as 'no match'.

Key Points

- PCRE2 enforces internal limits on backtracking steps or match time in order to protect system resources.
- Some patterns are syntactically valid but computationally unsafe and can trigger 'catastrophic backtracking'.
- When these limits are exceeded, the regex engine can abort processing and return no match, even if the URL logically matches the pattern.

Patterns to Avoid

Avoid regex constructs that combine:

- Nested quantifiers, for example: **(...+)***, **(.*)***, **(.+)**+, and so on
- Wildcards (.) repeated over large portions of the string, especially near the end of the pattern
- Unescaped dots in domain names when used together with repetition

For example, here the pattern is syntactically valid but can be expensive to process:

```
^([a-zA-Z0-9-]+.)*portal.example.com$
```

Note: In this case, **([a-zA-Z0-9-]+.)*** is a group with a nested quantifier (+ inside *) plus a wildcard (.). On some non-matching inputs, the regex engine can explore a very large number of backtracking paths.

Recommended Best Practices

Always Escape Dots in Hostnames

Use **\.** in order to match a literal dot, for example:

```
^([a-zA-Z0-9-]+\.)*portal\.example\.com$
```

Anchor Patterns and Restrict Characters

Use ^ and **$** and restrict to expected characters (for example, **[a-zA-Z0-9-]** for host labels) in order to reduce backtracking.

Avoid Nested, Unbounded Repetition Where Possible

Prefer simpler constructs rather than complex patterns that try to cover everything in one regex. Consider several specific entries instead of one very broad expression.

Test Patterns in a PCRE2-Compatible Tester

Before deployment, test regex patterns in a PCRE2-compatible environment and avoid patterns that raise 'catastrophic backtracking' or similar warnings.

Note: If a regex pattern hits the internal limits of the PCRE2 engine, it can be treated as 'no match' by the URL Filtering engine. In such cases, URL classification falls back to category or reputation, not the whitelist/blacklist regex result. The exact limits are implementation-specific and can change between releases. You must design regexes conservatively.

Differences in URL Matching for HTTP and HTTPS

The UTD engine inspects URLs differently for HTTPS and HTTP traffic. This affects how regular expressions must be designed for URL Filtering.

HTTPS (TLS) Traffic

For encrypted HTTPS traffic, the UTD engine does not decrypt the payload by default.

- URL Filtering uses the Server Name Indication (SNI) from the Transport Layer Security (TLS) ClientHello.
- The regex pattern is applied to the SNI hostname only, for example: **api.example.com**

In this case, a hostname-based pattern is matched against the hostname string **api.example.com** such as:

```
^([a-zA-Z0-9-]+\.)*example\.com$
```

HTTP (Unencrypted) Traffic

For plain HTTP traffic, the UTD engine can see the full HTTP request (request line and headers).

Depending on implementation, the string given to the regex engine can include:

- The full URL or request line (for example, **GET /path?param=value HTTP/1.1**) or
- The Host header combined with the path (for example, **api.example.com/path**)

As a result, the regex input for HTTP can contain additional characters such as **/**, **?**, and query strings, not just the bare hostname.

Configuration Implications

A regex designed purely for hostnames (for example, only matching **api.example.com**) can match HTTPS correctly (SNI) but fail to match HTTP request which contains a full URL or host+path string.

In order to filter both HTTP and HTTPS traffic with the same pattern, you must:

- Design patterns primarily around hostnames
- Verify behavior against both HTTP and HTTPS in the UTD logs

Verify

Enable Debug Logging

Step 1. Run the **debug utd engine standard url-filtering level info** command in order to enable debug logging.

Step 2. Run the **show logging process ioxman module utd | include api.example.com** command in order to verify the logs.

Example output:

```
2025/11/27 11:45:28.195000350 {ioxman_R0-0}{255}: [utd] [21292]: (note): :(#0):INSP-URLF event->server_
2025/11/27 11:45:28.195001873 {ioxman_R0-0}{255}: [utd] [21292]: (note): :(#0):INSP-URLF   URL: api.exa
2025/11/27 11:45:28.195009216 {ioxman_R0-0}{255}: [utd] [21292]: (note): :(#0):INSP-URLF Regex matched
2025/11/27 11:45:28.195022442 {ioxman_R0-0}{255}: [utd] [21292]: (note): :(#0):INSP-URLF URLF whitelist
2025/11/27 11:45:33.530605572 {ioxman_R0-0}{255}: [utd] [21292]: (note): :(#0):INSP-URLF   URL: api.exa
2025/11/27 11:45:33.530606333 {ioxman_R0-0}{255}: [utd] [21292]: (note): :(#0):INSP-URLF Regex not matc
2025/11/27 11:45:33.530614980 {ioxman_R0-0}{255}: [utd] [21292]: (note): :(#0):INSP-URLF URLF whitelist
```

Configuration Examples

Host-Based Matching

In order to allow all subdomains of **example.com**, use this recommended hostname-focused pattern (baseline):

```
^([a-zA-Z0-9-]+\.)*example\.com$
```

This pattern:

- Matches **example.com**, **api.example.com**, **foo.bar.example.com**, and so on
- Is suitable for HTTPS (SNI) matching
- Can also match HTTP if the string seen by the engine is the bare hostname

HTTP Host/Path Matching

If HTTP includes host/path and you want to ignore the path, you can match the hostname prefix and let the regex stop at a word boundary instead of a trailing. *****, for example:

```
^([a-zA-Z0-9-]+\.)*example\.com\b
```

Note: Here, **\b** (word boundary) effectively allows characters such as **/** or **?** in order to follow the hostname without requiring an explicit **.*** wildcard. This is generally cheaper than adding **.*** at the end and aligns better with the guidance in order to avoid additional unbounded wildcards.

Caution: The exact string passed into the regex engine for HTTP requests is implementation-specific and can evolve. When in doubt, test patterns against both HTTP and HTTPS traffic in a lab environment and verify matches in the UTD logs before you deploy to production.

Related Information

- Cisco Catalyst SD-WAN Security Configuration Guide, Cisco IOS XE Catalyst SD-WAN Release 17.x
- Cisco Technical Support & Downloads