LUCENE QUERIES IN DTM

Apache Lucene is the accepted standard in the text searching technology space and is relatively well known. On the surface, the syntax looks simple, but using it correctly often requires careful consideration.

Lucene queries are used in Digital Threat Monitoring (DTM) in three different places:

- In the DTM **Alerts** view when searching for alerts. Here, the Lucene query is used in the search bar on the **Alert List** page and executed for all your Alerts. See **Lucene queries for DTM alerts** **(https://docs.mandiant.com/home/dtm-lucene-queries-for-alerts)** for specific information including limitations.
- In your DTM **Monitors** with the `Lucene Text Query (Advanced)` Topic. In this use, the Lucene query is included in your Monitor condition used to match documents as they are ingested.

  For more information about the processes involved in matching Monitor conditions in Documents collected by DTM, see **Monitor Matching Methodology** **(https://docs.mandiant.com/home/dtm-monitor-matching-methodology)**.

- In DTM **Research Tools** when searching for historical documents. Here, Lucene is used in the search bar and executed against all documents DTM has collected.

Query Types

Lucene fundamentally supports five different types of queries:

- **Term**: Simplest type of query, usually a word.
- **Phrase**: Two or more words that must be present and adjacent.
- **Range**: Two terms marking the start and end of a range that any matching value must be between.
- **Wildcard**: A single word with either a single character (?) or multiple characters (*) being unknown.
- **Regular Expression** (regex): A regular expression describing a single token.

Term Query

A term query is the simplest type of Lucene query. A term is simply a single word or expression, for example:

```
hello
```

This query would match any documents that contained the term `hello` anywhere in the document as determined by the analyzer assigned to the field. For example, the following would match:

- `Hello world!`
- `Say Hello`

One limitation of a term query is that the entire term needs to match in order to qualify as a hit. So, for example, the following will not match:

- `Many hellos friend!`
- `Othello`

Term queries can be combined by simply adding additional terms:

```
hello world
```

This query would match the following:

- `Hello`
- `World`
- `Hello World`
- `World Hello`
- `The world says hello.`

Notice that in this example the order of the terms does not matter and that either term matching will cause the query to match. If all terms must match in order, use a **Phrase Query**. If you need to specify that both terms must be present, use **Boolean Operations**.

If you quote a single word it will be treated like a term query, which can have unintended consequences.

Phrase Query

Phrase queries are similar to term queries but they indicate that you require all terms in the phrase to be present and in the order specified. You specify a phrase simply by quoting the terms that you'd like to match:

```
"hello world"
```

would match the following:

- `Hello world!`
- `It's a good morning to say hello world.`

but would not match the following:

- `Hello, my name is Bob.`
- `Hello says the world.`

Range Query

Range queries let you specify a range of values that would match.

You specify ranges like this (the operator `TO` is case sensitive):

`[start_inclusive TO end_inclusive] OR {start_exclusive TO end_exclusive}`

Range fields can operate on any data type (such as text or dates). When comparing text, the ranges are sorted lexicographically. Using `[]` indicates that the range is *inclusive* meaning it will match even if the value lands on the boundary and using `{}` is *exclusive* meaning if a value lands on the boundary it will be excluded.

For example:

`[aaaa TO zzzz]`

would match the following:

- `aaaa`
- `dddd`
- `zzzz`

but not the following:

`1111`

And the following query:

`{aaaa TO zzzz}`

would match the following:

`dddd`

but not the following:

- `aaaa`
- `zzzz`
- `1111`

Wildcard Query

Wildcard queries let you search inside single terms (not phrases) by specifying a glob pattern using either `?` for a single character substitution or `*` to replace zero or more characters.

So the following:

`te?t`

would match both of the following:

- `this is some text`
- `match test`

And the following query:

`test*`

would match the following:

- `testify`
- `tester`

You can't use wildcards at the beginning of a term. Due to the nature of how this query works, the performance impact of running these queries is significant.

Regular Expression Query

Regular expression (regex) queries let you search for complicated patterns of text in source documents.

For example:

`/[sd]uper/`

would match both of the following:

- `"5701">super`
- `"5710">duper`

Regular expressions can be computationally expensive and complex regex queries can cause matches to take an excessive amount of time.

Both wildcard and regex queries only operate on single terms.

Query Modifiers

There are various modifiers that you can use to adjust how your queries get executed.

Field

If you only want your queries to match a particular field you can append the field before the query, followed by a colon:

```
body:"text to search for"
```

> A field specifier can be used with any type of query.

Fuzzy and Proximity Searching

For term and phrase queries, you can specify an integer that lets you change how specific a match must be in order to hit.

For terms, you can specify a fuzzy match by doing the following:

```
world~2
```

This query specifies that the term `world` must match within a **Levenshtein distance (https://en.wikipedia.org/wiki/Levenshtein_distance)** of two. The distance describes how many single character substitutions are allowed and still match. For example, this means that `warld` and `aarld` would match; but `aaald` would not.

For phrases you can specify a proximity match by doing the following:

```
"hello world"~2
```

Proximity matches look similar to fuzzy matches but operate very differently. This query specifies that the term `hello` must be found within two terms of `world`.

So the following would match all of the following:

- `Hello World`
- `Hello from World`
- `Hello from the World`

but would not match the following:

```
Hello from the wide World
```

Boolean Operations

Multiple conditions can be chained together using Boolean expressions. Lucene defines the following operators:

- `<query> OR <query>` or `<query> || <query>` : used to combine multiple queries where either one hitting will cause a match. This is the default operator; not specifying an operator will result in OR being used.
- `<query> AND <query>` or `<query> && <query>` : used to combine multiple queries where both must hit to cause a match.
- `+<query>` : **Required** operator, when preceding a query indicates that this query **must** hit for a match to occur.
- `-<query>` : **Prohibited** operator, when preceding a query indicates that this query **must not** hit for a match to occur.
- `<query> NOT <query>` or `<query> ! <query>` : Indicates that the second query must not hit for a match to occur. If you omit a preceding query to the NOT operator, the expression will never match.

> Boolean operators are case sensitive.

When considering multiple queries, Lucene automatically inserts the OR operator between terms. For example, the following functionally identical examples:

```
hello world
hello OR world
hello || world
```

All of these queries would match any text field containing `hello` or `world`.

- Instead of OR, you can use AND to specify both conditions must be met:
  `(this AND that) AND "other thing"`
- You can also indicate that Lucene queries must (+) or must not (-) be satisfied. For example, the following indicates that "this" must be found and "that" must not be found:
  `+this -that`
- You can combine all of these things in a single query. For example, the following would require that a Document must have a person identified with a name of "alice" but must not contain "bob" or "john":
  `identity_name:(+alice -(bob john))`

Similarly, the following are functionally the same:

```
hello AND world
hello && world
```

Both queries would match any text field containing both `hello` and `world`.

The following examples are also functionally the same:

```
hello -world
hello NOT world
hello ! world
```

Each query would match any text field containing `hello` as long as it didn't contain `world`.

`hello +world`

would match any text field containing `world` and it could optionally include `hello`.

Query Grouping

Multiple queries can be grouped by surrounding them in `(` and `)`. This approach can be useful if you want to nest your conditional logic to create more specific matches.

For example:

`("hello world" AND "john smith") OR ("goodbye world" AND "john doe")`

This query would match any document with a text field that:

- Contained the phrase "hello world" and also the phrase "john smith" or
- Contained the phrase "goodby world" and also the phrase "john doe"

You can also nest multiple layers deep:

`("hello world" AND "john smith") OR ("goodbye world" AND ("john doe" OR "jane doe"))`

This query would match any document with a text field that:

- Contained the phrase "hello world" and also the phrase "john smith" or
- Contained the phrase "goodby world" and also
  - The phrase "john doe" or
  - The phrase "jane doe"

Specific Data Types Searches

The following are some common examples of searches for specific data types.

Timestamp and Date

The following date formats are supported:

- `2006-01-02T15:04:05.999999999Z07:00`
- `2006-01-02T15:04:05Z07:00`
- `2006-01-02T15:04:05`
- `2006-01-02T15:04`
- `2006-01-02T15`
- `2006-01-02`
- `20060102`
- `2006-01`
- `200601`
- `2006`

IP Address

When searching for IP addresses, it's valid to search for CIDR ranges on both IPv4 and IPv6 addresses. The following are examples of valid IP address queries, all of which would match `192.168.1.1`:

- `192.168.1.1`
- `192.168.1.0/24`
- `::ffff:c0a8:101`
- `::ffff:c0a8:100/24`
- `0:0:0:0:0:ffff:c0a8:0101`
- `0000:0000:0000:0000:0000:ffff:c0a8:0101`

Special Character Usage

There are certain characters that have special meaning to Lucene, including the following:

`+-&|!(){}[]^"~*?:/ -- also ' '` (0x20, space)

These special characters can be escaped by prepending the character with a backslash `\`. So, if you wanted to search for a term containing an asterisk, for example, `hello\*world` would match the literal term `hello*world`.

> When text is processed by an Analyzer, most of these special characters will generally be stripped. So using these characters may not always have the intended effect in DTM, even if the characters are properly escaped.

Examples

The following are a few example queries combining special characters and query types:

1. **`body:exitq`**
   (https://advantage.mandiant.com/dtm?tab=researchTools&query=Ym9keTpleGl0cQ%3D%3D&dateRange=%2CTue+Nov+21+2023+23%3A59%3A59+GMT-0500+%28Eastern+Standard+Time%29)
   Search for all documents that contain a `body` field that contains the string `exitq`.

2. `exitq AND (bank OR credit)` (https://advantage.mandiant.com/dtm?tab=researchTools&query=ZXhpdHEgQU5EIChiYW5rIE9SIGNyZWRpdCk%3D&dateRange=%2CTue+Nov+21+2023+23%3A59%3A59+GMT-0500+%28Eastern+Standard+Time%29)
   Search all documents that contain `exitq AND bank` OR `exitq AND credit` .
3. `timestamp:["2022-06-03T19:01:53.442Z" TO "2022-08-03T19:01:53.442Z"] AND cve:"CVE-2020-15674"` (https://advantage.mandiant.com/dtm?tab=researchTools&query=dGltZXN0YW1wOlsiMjAyMi0wNi0wM1QxOTowMTo1My40NDJaIiBUTyAiMjAyMi0wOC0wM1QxOTowMTo1My40NDJaIl0gQU5EIGN2ZToiQ1ZFLTIwM[truncated]0500+%28Eastern+Standard+Time%29)
   Search for documents for the mention of a specific `cve` that have a specific range of timestamps.
4. `identity_name:purplefox422 AND service_name:raidforums.com` (https://advantage.mandiant.com/dtm?tab=researchTools&query=aWRlbnRpdHlfbmFtZTpwdXJwbGVmb3g0MjIgQU5EIHNlcnZpY2VfbmFtZTpyYWlkZm9ydW1zLmNvbQ%3D%3D&dateRange=%2CTue+Nov+21+2023-[truncated]0500+%28Eastern+Standard+Time%29)
   Searching for this TAA about **purplefox422** and pivoting over to all the posts by this persona on **raidforums**.
5. `identity_name:"john smith" AND doc_type:paste +password` (https://advantage.mandiant.com/dtm?tab=researchTools&query=aWRlbnRpdHlfbmFtZToiam9obiBzbWl0aCIgQU5EIGRvY190eXBlOnBhc3RlICtwYXNzd29yZA%3D%3D&dateRange=%2CTue+Nov+21+2023+23%3A59[truncated]0500+%28Eastern+Standard+Time%29)
   This query would match any DTM-analyzed paste that mentioned someone named `John Smith` and contained the word `hacked` anywhere in the Document. Note, the `+` in front of `password` is required because the default joining operator is `OR` . If you omit the `+` , you'd get any document containing the word `password` .

   > You could rewrite the previous query as: `+identity_name:"john smith" +doc_type:paste +password` because the `+` and `AND` operators are equivalent.

6. `doc_type:web_content_publish AND (domain:"boards.4chan.org" OR domain:"boards.4channel.org")` (https://advantage.mandiant.com/dtm?tab=researchTools&query=ZG9jX3R5cGU6d2ViX2NvbnRlbnRfcHVibGlzaCBBTkQgKGRvbWFpbjoiYm9hcmRzLjRjaGFuLm9yZyIgT1lgZG9tYWluOiJib2FyZHMuNGNoYW5uZWwub[truncated]0500+%28Eastern+Standard+Time%29)
   Search domain-related Documents for the domains `boards.4chan.org OR boards.4channel.org` . Notice `"` is used to include `.` in the string.
7. `email_address:/(.*)@acme.com/` (https://advantage.mandiant.com/dtm?tab=researchTools&query=ZW1haWxfYWRkcmVzczovKC4qKUBhY21ILmNvbS8%3D&dateRange=%2CTue+Nov+21+2023+23%3A59%3A59+GMT-0500+%28Eastern+Standard+Time%29)
   You could use the query to find any email address in the `abcde.com` domain. This query first scopes the search to the `email_address` topic, and then uses a regular expression that matches anything `(.*)` followed by the string `@acme.com` . Notice the use of `/(.*)` characters for a special regex meaning to Lucene.
8. `bin:nnnnnn*` or `bin:/nnnnnn.*/`
   You can use this query to search Research Tools for BINs.
9. `phone_number:(12223334444 12223334445 12223334446)`
   You can use this query to search Research Tools for phone numbers.Phone numbers use the `bin` analyzer, which only considers account numbers,. You can add additional characters but they'll be removed when they're searched.

Lucene Tips

The following are some important considerations when working with Lucene queries.

When to Use Lucene

Lucene can be incredibly useful for searching for terms that have a wide range of possible values. Its support for deep nesting boolean conditions makes it possible to run queries that are difficult to express any other way. Some potentially good use cases include the following:

- Finding specific ranges of BINs.
- Matching specific ranges of subdomains (using `must_end_with)` , for example.
- Searching for terms within a specific distance of each other using proximity searches.

When not to Use Lucene

When searching for a simple set of terms or keywords, Lucene is rarely a good fit. Lucene adds an extra layer of complexity that can make it especially difficult to figure out why something did or did not match.

Typically, Lucene should be treated as a last resort. It's often sufficient to use the Keyword Topic with different operators for most use cases.

Queries Operate on Terms

Most advanced query types such as wildcards, ranges, and regular expressions operate specifically on terms and cannot be used with phrases.

For example, the following query:

`"hello*" ["hello moon" TO "hello world"] "/hello world/"`

becomes the following:

`hello [hello moon TO hello world] "hello world"`

This result is almost never what's intended. Be careful when using phrases to avoid using features that only operate on terms.

For a simpler example, take the following query:

`"hello*"`

In this case, the phrase query gets Analyzed and the `*` gets removed and effectively becomes:

`hello`

Since this isn't a wildcard query, you won't find any Documents containing with terms like `helloworld`.

Phrases must contain Multiple Terms

Phrase queries that contain a single term are automatically decomposed to the equivalent term query.

For example, the following queries are equivalent:

`"hello" hello`

This can be surprising when using query modifiers like phrase proximity searching:

`"hello"~2`

Instead of performing a proximity search, this query would instead perform a fuzzy search.

Unexpected results from Analyzers

Lucene queries are Analyzed as part of being parsed. This process can lead to some surprising behaviors when combined with things like domain names, urls, paths, and other non-prose terms.

For example:

`"some-domain.com" some-domain.com`

Effectively becomes:

`"some domain.com" some domain.com`

Which especially in the later case could cause some surprising matches.

> For more information about Analyzers, see **Monitor Matching Methodology** **(https://docs.mandiant.com/home/dtm-monitor-matching-methodology)**.

Common Lucene Mistakes

- Forgetting to put quote around phrases, for example:
  `john smith` instead of `"john smith"`
  - The former would match any Document containing the terms `john` or `smith` anywhere in the Document. Because of the implicit OR between `john` and `smith` it would not require both to be present to match. This result can be especially surprising when there are long lists of match terms or match terms with short lengths.
- Forgetting to mark regular expressions, for example:
  `te[a-z]t instead of /te[a-z]t/`
- Forgetting to escape phrases causes the regular expressions to be considered literally, which will produce unexpected results because it effectively turns this query into the following:
  `te a-z t`
- Incorrect casing on the operators AND and OR, for example:
  `john and smith` instead of `john AND smith`
  - The operators AND and OR must be capitalized. Using the wrong case makes the condition part of the query, so in the former case the query would search for: `john OR and OR smith`
- Quoting single terms, such as `"hello*"` instead of `hello*` or `"/hello(.*)/"` instead of `/hello(.*)/`.
  - When quoting single terms special features like wildcards and regular expressions aren't evaluated. Instead they're treated like part of the phrase.
- Forgetting that pure negation matches everything. For example:
  `-wontfindthis`
  - Since there's no other condition to be met in this query, anything that doesn't match the negated term will be considered a match.
- Forgetting to use the correct syntax when including reserved characters in queries. For example:
  `(1+1):2` instead of `\(1\+1\)\:2`
  - Certain characters in Lucene have special meanings within queries. These queries are often used together to group terms or phrases for a certain type of segmentation (such as inclusive versus exclusive ranges). A common mistake is to forget the second character that's required to quit or escape from the segmented query behavior.

    > If a special character should be included as a search parameter, it must be enclosed within quotation marks.

Examples of Problematic Queries

- Consider the following Lucene query:
  `[a-zA-Z0-9]example.com`
  - This query was probably intended to find all instances of domains ending with "example.com" and starting with alphanumeric characters. What this query actually searches for:
    `za z0 9 example.com`
    - It should probably read:
      `domain:/[a-z0-9]+example\.com/`
- Consider the following Lucene query intended to search for a specific IP range:

`>192.168.0.*`
    - It should actually read:
      `ipv4_address:192.168.0.0/24`

- Consider the following Lucene query that is looking for CVE identifiers:
  `cve-*`
    - It should actually read:
      `cve:*`

- Consider the following Lucene query looking for a specific email address:
  `johnsmith@example.com`
    - It should actually read:
      `email_address:johnsmith@example.com`