Advanced Search Queries

The Carbon Black EDR console provides a check box interface to choose criteria for searches of processes, binaries, alerts, and threat reports. This chapter describes how to construct complex queries. The fields, field types, and examples in this chapter focus on queries to search for processes and binaries, but most of the syntax descriptions also apply to alerts and threat reports.

Sections

Query Syntax Details

Carbon Black EDR supports multiple types of operators and syntax that can form complex queries in the **Search** boxes on the Process Search, Binary Search, Threat Report Search, and Triage Alerts pages.

Searches are generally case-insensitive.

Terms, Phrases, and Operators

A term is a single keyword (without whitespace) that is searched in the Carbon Black EDR process or binary data store, or in the alerts or threat reports on your server. For example, a keyword could be: `svchost.exe.`

Terms can be combined by logical operators and nested to form complex queries; for example:

- and, AND, or whitespace — Boolean `AND` operator: `svchost.exe cmd.exe`, `svchost.exe and cmd.exe`
- or, OR — Boolean `OR` operator: `svchost.exe or cmd.exe`
- - — Boolean `NOT` operator: `-svchost.exe`
- nesting using parenthesis: `(svchost.exe or cmd.exe) powershell.exe”`
- Wildcard searches with *****; for example, `process_name:win*.exe`

Terms can be limited to a single field with *<field>:<term>* syntax; for example:

```
process_name:svchost.exe
```

Multiple terms are connected with `AND` if not otherwise specified.

Terms that are not preceded by fields are expanded to search all default fields.

Because terms are whitespace-delimited, use double quotes, or escape whitespaces with a single backslash, when required.

For example:

```
path:"microsoft office\office15\powerpnt.exe"
```

or

```
path:microsoft\ office\office15\powerpnt.exe
```

Terms can be combined to form phrases. A phrase is a set of terms that are separated by whitespace and enclosed in quotes. Whitespace between the terms of a quoted phrase is not treated as a logical `AND` operator. Instead, a phrase is searched as a single term.

For example: `“svchost.exe cmd.exe”`

Phrases can be combined and nested with other phrases and terms using logical operators.

For example: `"svchost.exe cmd.exe" or powershell.exe`

Restrictions on Terms

Whitespace

Whitespace is the default delimiter. A query with whitespace is “tokenized” and parsed as multiple terms.

For example:

This input: `microsoft office\office15\powerpnt.exe`

is interpreted as two terms: `microsoft AND office\office15\powerpnt.exe`

Use quotation marks to avoid automatic parsing into individual terms.

For example:

This input: `"microsoft office\office15\powerpnt.exe"`

Is interpreted as: `microsoft office\office15\powerpnt.exe`

Alternatively, you can escape whitespaces by using the backslash (\).

For example:

This input: `microsoft\ office\office15\powerpnt.exe`

Is interpreted as: `microsoft office\office15\powerpnt.exe`

See path for more information about how whitespaces and slashes affect path tokenization.

Parentheses

Parentheses are used as a delimiter for nested queries. A query with parentheses is parsed as a nested query, and if a proper nesting cannot be found, a syntax error is returned.

For example:

This input: `c:\program files (x86)\windows`

is interpreted as: `c:\program AND files AND x86 AND \windows`

Use quotation marks around the whole phrase to avoid automatic nesting. Otherwise, escape the parentheses (and whitespaces) using the backslash (\).

For example:

This input: `c:\program\ files\ \(x86\)\windows`

is interpreted as: `c:\program files (x86)\windows`

Negative Sign

The negative sign is used as logical `NOT` operator. Queries that begin with a negative sign are negated in the submitted query.

For example:

This input: `-system.exe`

is interpreted as: `not system.exe`

This input: `-alliance_score_srstrust:*`

is interpreted as: `Return all results that are not trusted by the alliance.`

You can use a phrase query to avoid automatic negation.

Double Quotes

Double quotes are used as a delimiter for phrase queries. A query in which double quotes should be taken literally must be escaped using backslash (\).

For example, the following query input:

```
cmdline:"\"c:\program files
\(x86\)\google\update\googleupdate.exe\" /svc"
```

is interpreted to match the following command line (with the command line including the quotes as shown):

```
"c:\program files (x86)\google\update\googleupdate.exe\" /svc
```

Leading Wildcards

The use of leading wildcards in a query is not recommended unless absolutely necessary, and is blocked by default. Leading wildcards carry a significant performance penalty for the search.

For example, the following query is not recommended:

```
filemod:*/system32/ntdll.dll
```

The same results would be returned by the following query, and the search would be much more efficient:

```
filemod:system32/ntdll.dll
```

Note

While process searches with leading wildcards are blocked by default beginning in Carbon Black EDR 6.2.3, you can change this either through the Advanced Settings page or the `cb.conf` file. For more information refer to the *VMware Carbon Black EDR Server Configuration Guide* "Managing High-Impact Queries".

Fields in Process and Binary Searches

This section contains a complete list of fields that are searchable in Carbon Black EDR process and binary searches. Some fields are valid in only one of the two, and some in both. Any binary-related field that the process search uses actually searches the executable file backing the process.

If a query specifies a term without specifying a field, the search is executed on all default fields. Default fields are indicated by `(def)`.

Note

Availability of SHA-256 hash data is dependent upon sensor capabilities. The macOS (OS X) sensor version 6.2.4, which is packaged with Carbon Black EDR Server version 6.3, sends SHA-256 hashes to the server. Check the VMware Carbon Black User Exchange or VMware Carbon Black Support for information about other sensors that can generate SHA-256 hashes.

For files that were originally discovered by a sensor that did not provide SHA-256 hashes, process information for new executions show SHA-256 hashes, but binary entries show SHA-256 as "(unknown)" until they appear as new files on a sensor that supports SHA-256. This applies to all SHA-256 related fields.

Field	Process Search	Binary Search	Field Type	Description
blocked_md5	x (def)	-	md5	MD5 of a process blocked due to a banning rule.
blocked_status	x	-	status	Status of a block attempt on a running process due to a banning rule, one of the following: a-ProcessTerminated b-NotTerminatedCBProcess c-NotTerminatedSystemProcess d-NotTerminatedCriticialSystemProcess e-NotTerminatedWhiltestedPath f-NotTerminatedOpenProcessError g-NotTerminatedTerminateError
childproc_count	x	-	count	Total count of child processes created by this process.
childproc_md5	x (def)	-	md5	MD5 of the executable backing the created child processes.
childproc_sha256	x (def)	-	sha256	SHA-256 of the executable backing the created child processes (if available).
childproc_name	x (def)	-	keyword	Filename of the child process executables.
cmdline	x (def)	-	cmdline	Full command line for this process.
comments	-	x (def)	text	Comment string from the class FileVersionInfo.
company_name	x	x (def)	text	Company name string from the class FileVersionInfo.
copied_mod_len	x	x	count	Number of bytes collected.
crossproc_count	x		count	Total count of cross process actions by an actor process.

Field	Process Search	Binary Search	Field Type	Description
crossproc_md5	x		md5	MD5 of an actor process that performed a cross process action on a target process.
crossproc_sha256	x		sha256	SHA-256 of an actor process that performed a cross process action on a target process (if available).
crossproc_name	x		keyword	Name of an actor process that performed a cross process action on a target process.
crossproc_type	x (def)		processopen remotethread processopentarget remotethreadtarget	• **processopen** (or process_open) finds processes which opened a handle into another process with a set of access rights. Sample results: OpenThread() API call requested THREAD_GET_CONTEXT, THREAD_SET_CONTEXT, THREAD_SUSPEND_RESUME access rights. • **remotethread** (or remote_thread) finds processes which injected a thread into another process. Sample results: CreateRemoteThread API used to inject code into target process. • **processopentarget** is similar to processopen above, but instead of finding the actor process returns the targeted process, i.e., the process which the handle is opened into. • **remotethreadtarget** is similar to remotethread above, but instead of finding the actor process returns the targeted process, i.e., the process which the thread was injected into.
digsig_issuer	x	x (def)	text	If digitally signed, the issuer.

Field	Process Search	Binary Search	Field Type	Description
digsig_prog_name	x	x (def)	text	If digitally signed, the program name.
digsig_publisher	x	x (def)	text	If digitally signed, the publisher.
digsig_result	x	x (def)	sign	If digitally signed, the result. Values are: • "Bad Signature" • "Invalid Signature" • "Expired" • "Invalid Chain" • "Untrusted Root" • "Signed" • "Unsigned" • "Explicit Distrust"
digsig_sign_time	x	x	datetime	If digitally signed, the time of signing.
digsig_subject	x	x (def)	text	If digitally signed, the subject.
domain	x (def)	-	domain	Network connection to this domain.
file_desc	x	x (def)	text	File description string from the class FileVersionInfo.
file_version	x	x (def)	text	File version string from the class FileVersionInfo.
filemod	x (def)	-	path	Path of a file modified by this process.
filemod_count	x	-	count	Total count of file modifications by this process.
filewrite_md5	x (def)	-	md5	MD5 of file written by this process.
filewrite_sha256	x (def)	-	md5	SHA-256 of file written by this process (if available).

Field	Process Search	Binary Search	Field Type	Description
group	x (def)	x (def)	keyword	Sensor group this sensor was assigned to at the time of process execution.
has_emet_config	x	-	bool	True or False - Indicates whether process has EMET mitigations configured/enabled.
has_emet_event	x	-	bool	True or False - Indicates whether process has EMET mitigation events.
host_count	-	x	integer	Count of hosts that have seen a binary.
host_type	x (def)	-	keyword	Type of the computer: workstation, server, or domain controller.
hostname	x (def)	x (def)	keyword	Hostname of the computer on which the process was executed.
internal_name	x	x (def)	text	Internal name string from the class FileVersionInfo.
ipaddr	x	-	ipaddr	Network connection to or from this IP address. Only a remote (destination) IP address is searchable regardless of incoming or outgoing.
ipv6addr	x	-	ipv6addr	**ipv6addr** Only a remote (destination) IP address is searchable regardless of incoming or outgoing.
ipport	x	-	integer	Network connection to this destination port.
is_64bit	x	x	bool	True if architecture is x64.
is_executable_image	x	x	bool	True if the binary is an EXE (versus DLL or SYS).

Field	Process Search	Binary Search	Field Type	Description
ja3	x	-	keyword	JA3 fingerprint of the client TLS hello packet.
ja3s	x	-	keyword	JA3S fingerprint of the server TLS hello packet.
last_server_update	x	-	datetime	Last activity in this process in the server's local time.
last_update	x	-	datetime	Last activity in this process in the computer's local time.
legal_copyright	x	x (def)	text	Legal copyright string from the class FileVersionInfo.
legal_trademark	x	x (def)	text	Legal trademark string from the class FileVersionInfo.
md5	x (def)	x (def)	md5	MD5 of the process, parent, child process, loaded module, or a written file.
sha256	x (def)	x (def)	sha256	SHA-256 of the process, parent, child process, loaded module, or a written file (if available).
modload	x (def)	-	path	Path of module loaded into this process.
modload_count	x	-	count	Total count of module loads by this process.
netconn_count	x	-	count	Total count of network connections by this process.
observed_filename	x	x (def)	path	Full path of the binary at the time of collection.
orig_mod_len	x	x	count	Size in bytes of the binary at time of collection.
original_filename	x	x (def)	text	Original name string from the class FileVersionInfo.

Field	Process Search	Binary Search	Field Type	Description
os_type	x	x	keyword	Type of the operating system: Windows, OSX or Linux.
parent_id	x	-	long	The internal Carbon Black EDR process guid for the parent process.
parent_md5	x (def)	-	md5	MD5 of the executable backing the parent process.
parent_sha256	x (def)	-	sha256	SHA-256 of the executable backing the parent process (if available).
parent_name	x (def)	-	keyword	Filename of the parent process executable.
path	x (def)	-	path	Full path to the executable backing this process.
private_build	x	x (def)	text	Private build string from the class FileVersionInfo.
process_id	x	-	long	The internal Carbon Black EDR process guid for the process.
process_md5	x (def)	-	md5	MD5 of the executable backing this process.
process_sha256	x (def)	-	sha256	SHA-256 of the executable backing this process (if available).
process_name	x (def)	-	keyword	Filename of the executable backing this process.
product_desc	x	x (def)	text	Product description string from the class FileVersionInfo.
product_name	x	x (def)	text	Product name string from the class FileVersionInfo.
product_version	x	x (def)	text	Product version string from the class FileVersionInfo.

Field	Process Search	Binary Search	Field Type	Description
regmod	x (def)	-	path	Path of a registry key modified by this process.
regmod_count	x	-	count	Total count of registry modifications by this process.
sensor_id	x	-	long	The internal Carbon Black EDR sensor guid of the computer on which this process was executed.
server_added_ timestamp	-	x	datetime	Time this binary was first seen by the server.
special_build	x	x (def)	text	Special build string from the class FileVersionInfo.
start	x	-	datetime	Start time of this process in the computer's local time.
tampered	x	x	bool	True if attempts were made to modify the sensor's binaries, disk artifacts, or configuration
username	x (def)	-	keyword	User context with which the process was executed.
watchlist_<id>	x	x	datetime	Time that this process or binary matched the watchlist query with <id>.

Fields in Alert and Threat Report Searches

Different sets of fields are searchable on the **Triage Alerts** and **Threat Report Search** pages. As with process and binary searches, if no field is specified for a term, the search is executed on all default fields. In the tables below, default fields are indicated by (def).

Field	Field Type	Description
alert_severity	float	Overall score of the alert (combines report score, feed rating, sensor criticality). For more information refer to the *VMware Carbon Black EDR Server Configuration Guide* “Threat Intelligence Feed Scores”.
alert_type	keyword	Type of the alert: one of "watchlist.hit.ingress.binary", "wathclist.hit.ingress.process", "watchlist.hit.query.process", "watchlist.hit.query.binary", "watchlist.hit.ingress.host"
assigned_to	keyword (def)	Name of the Carbon Black EDR administrator who changed the alert status.
create_time	datetime	Date and time this feed report was created.
created_time	datetime	Creation time of the alert.
description	text (def)	Description of the feed report, whitespace tokenized so each term is individually searchable.
domain	domain (def)	A domain IOC value in the feed report.
feed_category	text (def)	Category of this report/feed, whitespace tokenized.
feed_id	int	Numeric value of the feed id (-1 for watchlists).
feed_name	keyword (def)	Name of the feed that triggered the alert. All user-created watchlists have the feed name "My Watchlists" as a special case.
group	keyword	Sensor group name of the endpoint on which the process/binary that triggered the alert was observed.
hostname	keyword (def)	Hostname of endpoint that the process/binary that triggered the alert was observed on.
ioc_value	keyword (def)	Value (IP address, MD5, or SHA-256) of the IOC that caused the alert to be triggered.

Field	Field Type	Description
ipaddr	ipaddr	An IP address IOC value in the feed report.
ipv6addr	ipv6addr	An IPv6 address IOC value in the feed report.
is_ignored	bool	Indicates whether the report has been marked to be ignored on this server.
md5	md5 (def)	MD5 of the process that triggered the alert, or an MD5 IOC value in the feed report.
observed_filename	keyword (def)	Full path name of the process triggered the alert (not tokenized).
process_name	keyword (def)	Filename of the process that triggered the alert.
process_path	path (def)	Full path to the executable backing the process.
report_id	keyword	Name or unique identifier of the threat report that is part of the field.
report_score	float	Report score of the feed that triggered the alert. For more information refer to the *VMware Carbon Black EDR Server Configuration Guide* "Threat Intelligence Feed Scores".
resolved_time	datetime	Time this alert was triaged by a resolution action.
sha256	sha256 (def)	SHA-256 of the process that triggered the alert (if available), or a SHA-256 IOC value in the feed report.
status	keyword	Status of the alert: one of "resolved", "unresolved", "in progress", "false positive".
tags	text (def)	Tags related to this report/feed, whitespace tokenized.
title	text	Text title of the feed report, whitespace tokenized.
update_time	datetime	Date and time this feed report was last updated.

Field	Field Type	Description
username	keyword (def)	Username in whose context the process that triggered the alert event was executed.
watchlist_id	int (def)	Numeric value of the watchlist id (not applicable to feeds).
watchlist_name	keyword (def)	Name of the watchlist or the report (for feeds).

Field Types

domain

Domains are split into labels for query purposes. For example, "`example.com`" is split into "`example`" and "`com`".

If provided in a query, "dot" separator characters (.) between labels are maintained to enable position-dependent domain searches.

This has the following results:

- *Leading dot after the label, no trailing dot* – Returns results for matching labels that are at the *end* of the domain name.
- *Trailing dot after the label, no leading dot* – Returns results for matching labels that are at the *beginning* of the domain name.
- *Leading and trailing dots surrounding the label* – Returns results for matching labels that are in the middle of the domain name (i.e., not the first or last label).
- *Two labels with a dot between them* – Treated as a search for the entire phrase, and so returns results for domains that include the entire string.
- *No dot separators* – Returns results for any domain that includes the query string anywhere in the domain name.

The following table provides examples of these different domain searches:

Search	If domain is foo.com	If domain is foo.com.au
domain:com	**match**	**match**
domain:.com	**match**	no match
domain:.com.	no match	**match**
domain:com.	no match	no match
domain:example.	**match**	**match**
domain:example.com	**match**	no match

ipaddr

IP addresses are searched with a CIDR notation:

`(ip)/(netmask)`

If the netmask is omitted, it is presumed to be 32.

For example:

`ipaddr:192.168.0.0/16` or `ipaddr:10.0.1.1`

ipv6addr

IPv6 addresses are searched with a CIDR notation:

`(ip)/(netmask)`

If the netmask is omitted, it is assumed to be 32.

For example:

`ipv6addr:fe00:b9:266:2011:28dc:43d4:3298:12e2` or
`ipv6addr:fe00:b9:266:2011::0/50`

text

Text fields are tokenized on whitespace and punctuation. Searches are case-insensitive.

For example, the string from the product_name field:

`Microsoft Visual Studio 2010`

is interpreted as `microsoft AND visual AND studio AND 2010`.

Searches for any of these strings will match on the binary. Phrase queries for any two consecutive terms also match on the binary.

For example:

```
product_name: "visual studio"
```

count

An integer value. If it exists, the values are from `0` to `MAXINT`. It supports two types of search syntaxes:

- `X:` Matches all fields with precisely `X`. For example, `modload_count:34` for processes with exactly 34 modloads.
- `[X TO Y]:` Matches all fields with counts `>= X` and `<= Y`. For example, `modload_count:[1 TO 10]` for processes with 1 to 10 modloads.

In both cases, either `X` or `Y` can be replaced by the wildcard *. For example:

`netconn_count:*` for any process where the `netconn_count` field exists.
`netconn_count:[10 TO *]` for any process with more than 10 network connections.

datetime

Datetime fields have five types of search syntaxes:

- `YYYY-MM-DD` matches all entries on this day, for example, `start:2016-12-01` for all processes started on Dec 1, 2016.
- `YYYY-MM-DDThh:mm:dd` matches all entries within the next 24 hours from this date and time, for example, `start:2016-12-01T22:15:00` for all processes started between Dec 1, 2016 at 22:15:00 to Dec 2, 2016 at 22:14:59.
- `[YYYY-MM-DD TO YYYY-MM-DD]` matches all entries between, for example, `start:[2016-12-01 TO 2016-12-31]` for all processes started in Dec 2016.
- `[YYYY-MM-DDThh:mm:ss TO YYYY-MM-DDThh:mm:ss]` matches all entries between, for example, `start:[2016-12-01T22:15:00 TO 2016-12-01:23:14:59]` for all processes started in Dec 1, 2016 within the given time frame.
- `-Xh` relative time calculations matches all entries with a time between `NOW-10h` and `NOW`. Support units supported are h: hours, m: minutes, s: seconds as observed on the host, for example, `start:-24h` for all processes started in the last 24 hours.

As with counts, `YYYYMMDD` can be replaced the wildcard *, for example, `start:[2016-01-01 TO *]` for any process started after 1 Jan 2016.

keyword

Keywords are `text` fields with no tokenization. The term that is searched for must exactly match the value in the field, for example, `process_name:svchost.exe`.

Queries containing wildcards can be submitted with keyword queries.

For example:

```
process_name:ms*.exe.
```

md5

md5 fields are keyword fields with an md5 hash value.

The term searched for must exactly match the value in the field.

For example:

```
process_md5:6d7c8a951af6ad6835c029b3cb88d333.
```

sha256

sha256 fields are keyword fields with a SHA-256 hash value.

The term searched for must exactly match the value in the field.

For example:

```
process_sha256:BCB8F25FE404CDBFCB0927048F668D7958E590357930CF620F
74B59839AF2A9C.
```

ja3

ja3 fields are keyword fields with a ja3 hash value. You can search for the hash value. The term searched for must exactly match the value in the field.

For example:

```
ja3:669181128F1B9B03303D77C6F2EEFD128
```

ja3s

ja3s fields are keyword fields with a ja3s hash value. You can search for the hash value. The term searched for must exactly match the value in the field.

For example:

```
ja3s:679183361F1C6F13201C62F6F2CFED111
```

path

Path fields are special text fields. They are tokenized by path hierarchy as follows:

```
path:c:\windows.
```

For a given path, all subpaths are tokenized. For example:

```
c:\windows\system32\boot\winload.exe
```

is tokenized as:

```
c:\windows\system32\boot\winload.exe
windows\system32\boot\winload.exe
system32\boot\winload.exe
boot\winload.exe
winload.exe
```

Wildcard Searches

For queries involving path segments that are not tokenized, wildcard searches can be submitted.

For example, you can enter:

```
path:system*
```

for any path that has `system` as sub-path in it.

Modload Path Searches

When performing a loadable module filename (modload) search (as shown in path), leading forward and back slashes are tokenized. You do not have to remove the leading slash for modload path searches, although it is recommended.

For example:

```
\boot\winload.exe
```

should be entered as:

```
boot\winload.exe
```

Regmod Path Searches

When performing a Windows registry (regmod) search, a few important search caveats exist:

- If a regmod search term contains `controlset001` or `controlset002`, the search term is normalized and tokenized as `currentcontrolset`. As a result, you should search by replacing `controlsetXXX` with `currentcontrolset`.

  For example:

  ```
  registry\machine\system\controlset001\services\xkzc
  ```

  should be entered as:

  ```
  regmod:registry\machine\system\currentcontrolset\services\xkzc
  ```

- The leading backslash on regmod search terms are not tokenized. For regmod searches, be sure to omit this character when submitting search terms.

  For example:

`\registry\machine\system\controlset001\services\xkzc`

should become:

`regmod:registry\machine\system\currentcontrolset\services\xkzc`

bool

Boolean fields have only two possible values: the string `true` or `false`. Searches are case-insensitive.

sign

Signature fields can be one of the eight possible values:

- `Signed`
- `Unsigned`
- `Bad Signature`
- `Invalid Signature`
- `Expired`
- `Invalid Chain`
- `Untrusted Root`
- `Explicit Distrust`

Values with whitespace must be enclosed in quotes.

For example:

`digsig_result:Signed` or `digsig_result:"Invalid Chain"`

cmdline

When a process launches on an endpoint, the command line for that process is sent to the Carbon Black EDR server. If the server stored the whole command line as one item and allowed open ended queries of it, query performance would be extremely poor to the point of making search unusable. Instead, the server breaks each command line up into smaller component “tokens” to be stored for use when you enter a command line query.

Tokenization requires that decisions be made about which components of a command become their own token and which components are treated as delimiters between tokens. These decisions involve trade-offs since the same character may be used in different ways in a command. The following section describes how tokenization is done for Carbon Black Hosted EDR instances and Carbon Black EDR 6.3.0 servers (and later). If you are upgrading, see also Tokenization Changes on Server Upgrade.

Tokenization Rules

Characters Removed Before Tokenization

With enhanced tokenization, the following characters are converted to white spaces and emoved before the command-line is tokenized:

```
\ " ' ( ) [ ] { } , = < > & | ;
```

Several frequently used characters are intentionally not removed before tokenization. These include:

- Percent (`%`) and dollar (`$`), often used for variables
- Dash (−), period (`.`), and underscore (_), often found as parts of file names
- These additional characters: `^ @ # ! ?`

Parsing Forward Slashes

The forward slash (`/`) character is handled differently depending upon its position. If it is the start of the entire command line, it is assumed to be part of the path. If it is at the start of any other token in the command line, it is assumed to be a command line switch.

There is one situation in which this parsing rule may not produce the results you want. It is not efficient for the command line parser to distinguish between a command line switch and a Unix-style absolute path. Therefore, Linux and Mac absolute paths passed on the command line are tokenized as if the beginning of the path were a command line switch. So a command line of `/bin/ls /tmp/somefile` will produce the tokens `bin`, `ls`, `/tmp` and `somefile`, incorrectly considering `/tmp` a command line switch.

Parsing Colons

The colon (:) character is handled differently depending upon its position and whether it is repeated. If it is the end of a token, it is assumed to be something the user would want to search for like a drive letter, so it is included. If there are multiple colons at the end of a token or if the colons are not at the end of a token, they are converted to white space for tokenization purposes.

File Extension Tokens

File extension tokens allow searching for either just the file extension or the entire command or file name. In other words, “word.exe” in a command line becomes two tokens: “.exe” and “word.exe”.

Wildcards

There is support for the '?' and '*' characters as wildcards when used as a non-leading character in a query, allowing you to search for any single character or multiple variable characters within a token, respectively.

Note

Wildcards **should not** be used as leading characters in a search.

Tokenization Changes on Server Upgrade

This section is relevant to on-premise users upgrading from a pre-6.3.0 version of Carbon Black EDR. If 6.3.0 is your first version of Carbon Black EDR or if you are using a Carbon Black Hosted EDR instance, you do not need to review this section.

Beginning with version 6.1.0, Carbon Black EDR included tokenization option that improved command-line searches. This is standard for Carbon Black Hosted EDR instances, and beginning with version 6.3.0, it is also standard for Carbon Black EDR installations. It adds the following specific improvements, which are described in more detail below:

- More special characters are removed before tokenization.
- Forward slash “/” is interpreted as a command line switch or a path character depending upon position.
- Colon “:” is interpreted as part of a drive letter token or converted to white space depending upon position and repetition.
- File extensions are stored as a separate token as well as part of a file or path name.
- Wildcards are supported in non-leading positions within a query.

These changes result in simpler queries, better and faster search results, and reduced storage requirements for tokenized command lines.

Note

If you upgraded from a pre-6.3.0 release and configured Watchlists that use command line queries, these might require a re-write to take advantage of the new tokenization. Review your Watchlist entries to make sure they return the intended results.

Example: Enhanced vs. Legacy Tokenization

The following example shows how the enhanced tokenization in version 6.3.0 differs from the previous version. It can help you convert some older queries to the new standard:

```
"C:\Windows\system32\rundll32.exe" /d
srrstr.dll,ExecuteScheduledSPPC
```

Using **legacy** tokenization, the command was broken into the following tokens:

```
"c:

windows
```

```
system32

rundll32.exe”

d

srrstr.dll,executescheduledsppc
```

The **enhanced** tokenization in version 6.3.0 breaks the same command into the following tokens:

```
c:

windows

system32

rundll32.exe

.exe

/d

srrstr.dll

.dll

executescheduledsppc
```

Examples of new search capabilities due to this tokenization include:

- You can search for .exe or .dll as part of the command line query.
- Because of more complex parsing of the forward slash, you can explicitly search for a ‘/d’ command line argument and not worry about false positives from just searching for the letter ‘d’.
- You can use a wildcard and search for ‘"execute*’ if you want to find a specific term passed to the command line.
- You do not have to include extraneous single or double quote marks to find a drive letter or command path.

Retention Maximization and cmdline Searches

On the Edit Group page for a sensor group, you can specify **Retention Maximization** options that help control the information that is recorded on the server to manage bandwidth and processing costs (for more information refer to the *VMware Carbon Black EDR Server Configuration Guide* - “Advanced Settings”). As part of this feature, the process cmdline field for parent processes store also store the cmdlines of their child processes (childprocs) that are affected by a retention setting. This is done because these childprocs do not have process documents of their own to store this information and so the expanded parent cmdline provides a way to search cmdlines for processes no longer recorded separately.

A side-effect of including the cmdlines of these childprocs in the parent’s cmdline info is that a cmdline search intended to match only the parent process’s cmdline will also match against the children. This can result in the parent process getting falsely tagged as a feed hit based on matching a childproc that was not judged to be interesting enough to

justify the creation of a complete process doc. Keep this in mind when choosing **Retention Maximization** settings.

Searching with Multiple (Bulk) Criteria

You can search for multiple IOCs by using bulk search criteria in both the Process Search and Binary Search pages. While you could just enter a chain of “ORed” terms, Carbon Black EDR provides special interfaces for bulk searches that do this for you when given a list of terms. You can type or paste multiple terms into a bulk search text box, following these syntax requirements:

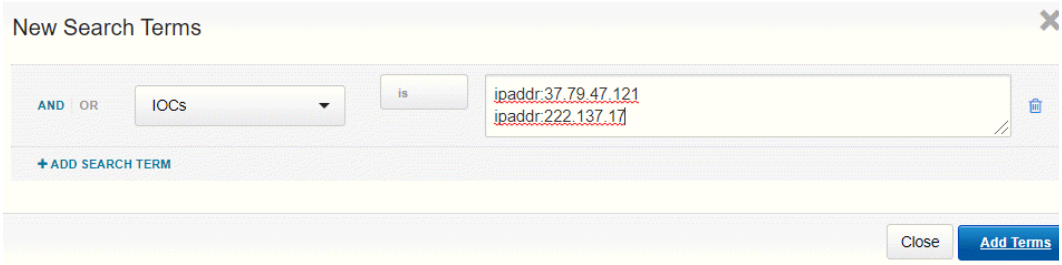
- Each term must be on its own line.
- No punctuation is required or allowed (for example, no comma-separated lists or parentheses).
- You must use the “ipaddr:” prefix to successfully use a list of IP addresses in a bulk search.
- For most other types of data, such as md5s, prefixes are optional but more efficient. See Fields in Process and Binary Searches for a table of search criteria types and their prefixes.

If a bulk search is initiated using terms without prefixes, the search is treated as a generic text search and will match the terms listed to any field. In the case of IP addresses without the “ipaddr” prefix, the search will fail because the terms are dealt with as individual numbers rather than four-part addresses.

Bulk IOC searches can be added to other search criteria or used as the only criteria for a search.

To do a bulk IOC search on the Process Search page:

1. On the Process Search page, unless you have already entered some terms to include in your search, click the **Reset Search** button under the search box to start with a fresh search.
2. Click **Add Search Terms**. Click the **Choose Criteria** dropdown menu and click **Bulk IOC > IOCs**.
3. In the text box, type or paste the list of IOCs to search for, making sure they meet the syntax requirements described in this section. For example:

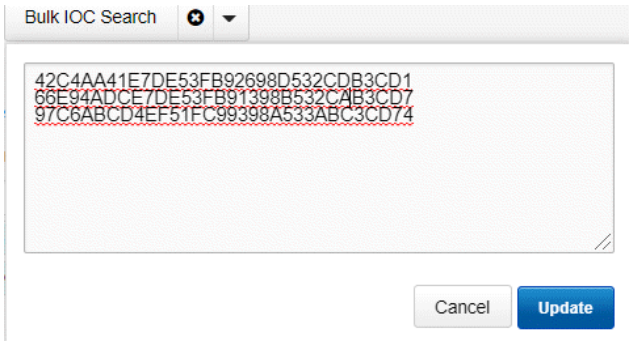


4. For most search criteria, you are probably interested in records that match one of the items on your list; however, you also can choose to get results that do not match your terms. Use the **is / is not** toggle in the dialog to make this choice.
5. To include additional search criteria, click the **Add Search Term** link.
6. When you have finished defining your search, click the **Add Terms** button.

   Your search is initiated and the results (if any) are shown in the table on the Process Search page. If necessary, you can continue to refine your search by using the search facet tables or you can manually enter terms.

To do a bulk IOC search on the Binary Search page:

1. On the Binary Search page, unless you have already entered some terms to include in your search, click the **Reset Search Terms** button to start with a fresh search.
2. Click the **Add Criteria** dropdown menu and, under **Bulk search**, select **IOCs**.
3. In the text box, type or paste the list of IOCs to search for, making sure they meet the syntax requirements described in this section.



4. Click **Update** to apply the search terms.

   Your search is initiated and any results are shown in the table on the Binary Search page. If necessary, you can continue to refine your search using the search facet tables or by manually entering terms.

Searching with Binary Joins

Some binary search fields can be used as part of a process search query. (for more information, see Fields in Process and Binary Searches.)

In this case, the results returned are process instances that are backed by binaries that match the binary search criteria. This is called a *joined search*. For example, consider submitting the following query on the **Process Search** page:

```
digsig_result:Unsigned
```

This query returns all process instances that are backed by an unsigned MD5. By default, join searches are performed against the MD5 of the standalone process executable (`process_md5`). However, joined searches can also be performed against the MD5 of the following related events:

- `filewrites = <binary field>_filewrite`
- `parent processes = <binary_field>_parent`
- `child processes = <binary_field>_child`
- `modloads = <binary_field>_modload`

Specify the search by adding the following suffixes to the end of the binary search field:

- `filewrite`
- `parent`
- `child`
- `modload`

For example:

```
digsig_result_modload:Unsigned
```

This query returns all process instances that have loaded an unsigned module.

Note

Process searches involving large binary joins are blocked by default beginning in Carbon Black EDR 6.2.3. For more information refer to the *VMware Carbon Black EDR Server Configuration Guide* - “Managing High-Impact Queries” to modify this behavior.

Example Searches

Process Search Examples

Example Query Strings	Result
domain:www.carbonblack.com	Returns all processes with network connections to or from domains matching the given FQDN.
domain:.com	Returns all processes with network connections to or from domains matching `*.com`
domain:.com.	Returns all processes with network connections to or from domains matching the form `*.com.*`
domain:www.	Returns all processes with network connections to or from domains matching the form `www.*`
domain:microsoft	Returns all processes with network connections to or from domains matching `*.microsoft OR *.microsoft.* OR microsoft.*`
ipaddr:127.0.0.1	Returns all processes with network connections to or from IP address `127.0.0.1`
ipaddr:192.168.1.0/24	Returns all processes with network connections to or from IP addresses in the network subnet `192.168.1.0/24`
ipv6addr:fe00:b9:266:2011:28dc:43d4:3298:12e2	Returns all processes with network connections to or from IPv6 address `fe00:b9:266:2011:28dc:43d4:3298:12e2`
ipv6addr:fe00:b9:266:2011::0/50	Returns all processes with network connections to or from IPv6 addresses in the range of network subnet `fe00:b9:266:2011::0/50`
modload:kernel32.dll	Returns all processes that loaded a module `kernel32.dll` (accepts path hierarchies).

Example Query Strings	Result
modload:c:\windows\system32\sxs.dll	Returns all processes that loaded a module matching path and file `sxs.dll` (accepts path hierarchies).
path:c:\windows\system32\notepad.exe	Also returns all processes with the matching path (accepts path hierarchies).
regmod:\registry\machine\system\currentcontrolset\control\deviceclasses* **Notes:** Substitute “controlset001” or “controlset002” with “currentcontrolset”, as shown in this example query string. The regmod event in the process document still uses the original string, but searches must always use "currentcontrolset". regmod searches must include the complete path string or use wildcards. Searches for partial regmod paths without wildcards never yield results.	Returns all processes that modified a registry entry with the matching path (accepts path hierarchies).
path:excel.exe	Returns all processes with the matching path (accepts path hierarchies).
cmdline:backup	Returns all processes with matching command line arguments.
hostname:win-5ikqdnf9go1	Returns all processes executed on the host with matching hostname.
group:"default group"	Returns all processes executed on hosts with matching group name (use of quotes are required when submitting two-word group names).
host_type:workstation	Returns all processes executed on hosts with matching type (use of quotes are required when submitting two-word host types).

Example Query Strings	Result
username:system	Returns all processes executed with the matching user context.
process_name:java.exe	Returns all processes with matching names.
parent_name:explorer.exe	Returns all processes executed by a parent process with matching names.
childproc_name:cmd.exe	Returns all processes that executed a child process with matching names.
md5:5a18f00ab9330ac7539675f3f326cf11	Returns all processes, modified files, or loaded modules with matching MD5 hash values.
process_md5:5a18f00ab9330ac7539675f3f326cf11	Returns all processes with matching MD5 hash values.
parent_md5:5a18f00ab9330ac7539675f3f326cf11	Returns all processes that have a parent process with the given MD5 hash value.
filewrite_md5:5a18f00ab9330ac7539675f3f326cf11	Returns all processes that modified a file or module with matching MD5 hash values.
childproc_md5:5a18f00ab9330ac7539675f3f326cf11	Returns all processes that executed a child process with matching MD5 hash values.
<type>_count:*	Returns all processes that have xxx_count field > 0, where type is one of modload, filemod, regmod, netconn, or childproc.
<type>_count:10	Returns all processes that have xxx_count field = 10, where type is one of modload, filemod, regmod, netconn, or childproc.
<type>_count:[10 TO 20]	Returns all processes that have xxx_count field >= 10 and <= 20, where type is one of modload, filemod, regmod, netconn, or childproc.
<type>_count:[10 TO *]	Returns all processes that have xxx_count field >= 10, where type is one of modload, filemod, regmod, netconn, or childproc.

Example Query Strings	Result
<type>_count:[* TO 10]	Returns all processes that have xxx_count field < 10, where type is one of modload, filemod, regmod, netconn, or childproc.
start:2011-12-31	Returns all processes with a start date of 2011-12-31 (as observed on the host).
start:[* TO 2011-12-31]	Returns all processes with a start date earlier than or equal to 2011-12-31 (as observed on the host).
start:[* TO 2011-12-31T22:15:00]	Returns all processes with a start date earlier than or equal to 2011-12-31 at 22:15:00 (as observed on the host).
start:[2011-12-31 TO *]	Returns all processes with a start date later than or equal to 2011-12-31 (as observed on the host).
start:[2011-12-31T09:45:00 TO *]	Returns all processes with a start date later than or equal to 2011-12-31 at 09:45:00 (as observed on the host).
start:*	Returns processes with any start date (as observed on the host).
start:[* TO *]	Returns processes with any start date (as observed on the host).
start:-10h	Returns all processes with a start time between NOW-10h and NOW. Units supported are, h: hours, m: minutes, s: seconds (as observed on the host).
last_update:2011-12-31	Returns all processes last updated on date 2011-12-31 (as observed on the host).
last_update:[* TO 2011-12-31]	Returns all processes last updated on a date earlier than or equal to 2011-12-31 (as observed on the host).
last_update:[* TO 2011-12-31T22:15:00]	Returns all processes last updated on a date earlier than or equal to 2011-12-31 at 22:15:00 (as observed on the host).

Example Query Strings	Result
last_update:[2011-12-31 TO *]	Returns all processes last updated on a date later than or equal to 2011-12-31 (as observed on the host).
last_server_update:[2011-12-31T09:45:00 TO *]	Returns all processes last updated on a date later than or equal to 2011-12-31 at 09:45:00 (as observed at the server).
last_server_update:*	Returns processes with any update date (as observed on the server).
last_server_update:[* TO *]	Returns processes with any update date (as observed on the server) within the range provided.
last_server_update:-10h	Returns all processes last updated between NOW-10h and NOW. Units supported are h: hours, m: minutes, s: seconds (as observed on the server).
process_id:<guid>	Returns the process with the given process id, where `<guid>` is a signed 64-bit integer.
parent_id:<guid>	Returns the process with the given parent process id, where `<guid>` is a signed 64-bit integer.
sensor_id:<guid>	Returns processes executed on host with given sensor id, where `<guid>` is an unsigned 64-bit integer.

Binary Search Examples

Example Query Strings	Result
`md5:5a18f00ab9330ac7539675f326cf11`	Returns all binaries with matching MD5 hash values.
`digsig_publisher:Oracle`	Returns all binaries with a digital signature publisher field with a matching name.
`digsig_issues:VeriSign`	Returns all binaries with a digital signature issuer field with a matching name.
`digsig_subject:Oracle`	Returns all binaries with a digital signature subject field with a matching name.
`digsig_prog_name:Java`	Returns all binaries with a digital signature program name field with a matching name.
`digsig_result:Expired`	Returns all binaries with a digital signature status of `<status>`.
`digsig_sign_time:2011-12-31`	Returns all binaries with a digital signature date of 2011-12-31.
`digsig_sign_time:[* TO 2011-12-31]`	Returns all binaries with a digital signature date earlier than or equal to 2011-12-31.
`digsig_sign_time:[2011-12-31 TO *]`	Returns all binaries with a digital signature date later than or equal to 2011-12-31.
`digsig_sign_time:*`	Returns binaries with any digital signature date.
`digsig_sign_time:[* TO *]`	Returns binaries with any digital signature date within the range provided.
`digsig_sign_time:-10h`	Returns all binaries with a start time between NOW-10h and NOW. Units supported are h: hours, m: minutes, s: seconds.

Example Query Strings	Result
`<type>_version:7.0.170.2`	Returns all binaries with matching version, where `<type>` is product or file.
`product_name:Java`	Returns all binaries with matching product name.
`company_name:Oracle`	Returns all binaries with matching company name.
`internal_name:java`	Returns all binaries with matching internal name.
`original_filename:mtxoci.dll`	Returns all binaries with matching filename.
`observed_filename:c:\windows\system32\mtxoci.dll`	Returns all binaries that have been observed to run on or were loaded with the given path.
`<type>_mod_len:[* TO 10]`	Returns all binaries that have `<type>_mod_len` (module length in bytes) field < 4096, where type is original or copied.
`<type>_desc:"database support"`	Returns all binaries that have `<type>_desc` field with matching text, where type is file or product.
`legal_<type>:Microsoft`	Returns all binaries with matching `legal_<type>` field text, where type is trademark or copyright.
`<type>_build:"Public version"`	Returns all binaries with matching `<type>_build` field text, where type is special or private.
`is_executable_image:True or False`	Boolean search (case insensitive) returning all binaries that are executable or not executable.
`is_64bit_:True or False`	Boolean search (case insensitive) returning all binaries that are 64-bit or not 64-bit.
`watchlist_4:[2014-04-01 TO 2014-09-31]`	Returns all binaries that matched watchlist 4 during the time period shown.

Threat Intelligence Search Examples

Any document matching a threat intelligence feed is tagged with an `alliance_score_<feed>` field, where the value is a score from -100 to 100.

For more information refer to the *VMware Carbon Black EDR Server Configuration Guide* "Threat Intelligence Feeds".

<feed> is the "short name" of the threat intelligence feed, such as **nvd** or **isight**.

For any threat intelligence feed, you can click the **View Hits** button to discover the feed's short name. For more information refer to the *VMware Carbon Black EDR Server Configuration Guide* - "Threat Intelligence Feeds".

Example Query Strings	Result
`alliance_score_<feed>:*`	Returns all binaries that have *<feed>* score > 0.
`alliance_score__score_<feed>:10`	Returns all binaries that have *<feed>* score = 10.
`alliance_score__score_<feed>:[10 TO 20]`	Returns all binaries that have *<feed>* score >= 10 and <= 20.
`alliance_score__score_<feed>:[10 TO *]`	Returns all binaries that have *<feed>* score >= 10.
`alliance_score__score_<feed>:[* TO 10]`	Returns all binaries that have *<feed>* score < 10.