BE THE HUNTER

# Security Analytics Chalk Talk

## Optimizing Investigation

## Optimizing Investigation

## Security Analytics database

The Security Analytics Core products contain a proprietary database developed specifically for use within the Security Analytics suite of products. It bears very little resemblance to traditional relational databases, and it is not based on any off-the-shelf database technology. The purpose of this session is to help Security Analytics users understand the database and use it to it's fullest potential.

The following components of Security Analytics contain the Core Databases:

1. Concentrator
2. Archiver
3. Decoder
4. Log Decoder

## Core databases:

Packet DB: The packet database contains raw packet information. On a Decoder, the packet database contains raw packets as captured from the network. Log Decoders utilize the packet database to store raw log information. Each packet stored in the packet database is accessible by a Packet ID.

Meta DB: The meta database contains items of information that are extracted by a Decoder or Log Decoder from the raw data stream. Meta items can be generated by parsers, rules or feeds.

Session DB: The session database contains information that ties the packet and meta items together into sessions.

## Packet, Meta, and Session Storage

Each of the packet, meta, and session databases are configured through the /database/config folder on each SA Core service.

Each database has a configurable parameter to specify where the Core service will store data.

Configuration items for the packet database start with the prefix "packet", meta database configuration starts with the prefix "meta", and the session database configuration items start with the prefix "session".

## Index Storage:

The index configuration is stored in the /index/config folder on each Core service.

## Description of each database configuration node:

packet.dir, meta.dir, session.dir

This is the primary configuration entry for each database. It controls where in the filesystem the database is stored. This configuration entry understands a complex syntax for specifying many directories as storage locations.

Example:

```
/var/netwitness/decoder/packetdb=10t;/var/netwitness/decoder0/packetdb=20.5t
```

The size values are optional. If set they indicate the maximum total size of files stored there before databases roll over. If the size is not present the database will not automatically roll over, but it's size can be managed using other mechanisms.

The use of = or == is significant. The default behavior of the databases is to automatically create directories specified when the Core service starts. However, this behavior can be overridden by using the == syntax. If == is used, the service will not create any directories. If the directories do not exist when the service starts, the service will not successfully start processing. This gives the service resilience against filesystems that are missing or unmounted when the appliance boots.

## SDK Configuration Items that affect Database

There are some additional configuration items in each core service that affect the database, but do not actually affect how the database stores or retrieves data. These settings exist in the /sdk/config folder.

max.concurrent.queries

This setting controls how many query operations are allowed on the database simultaneously. Allowing more simultaneous query operations can improve overall responsiveness for more users, but if the query load of the Core device is very I/O bound, having a high max.concurrent.queries value can have a detrimental effect. The recommended value is the number of cores on the system, including hyper threading. Thus, for a Series 4 appliance with 16 cores, the value should be 32.

## max.pending.queries

This setting controls the backlog size for the database's query engine. Larger values allow the database to queue more operations for execution. A queued query does not make progress on it's execution, so it may be more useful to make the system produce errors when the queue is full, rather than allowing the queue to grow very large. However, on a system that is primarily performing batch operations such as reports, there may be no detrimental effect to having a large queue.

## cache.window.minutes

This setting controls a feature of the query engine that is intended to improve query responsiveness when there are a large number of simultaneous users.

## max.where.clause.cache

The where clause cache controls how much memory can be consumed by query operations that need to produce a large temporary data set to evaluate sorting or counting. If the where clause cache size is overflowed, the query will still work, but it will be much slower. If the where clause cache is too large, it is possible for queries to allocate so much memory that the service would be forced into swap or will run out of memory. Thus, this value multiplied by the **max.concurrent.queries** should always be much less than the size of physical RAM. This setting understands sizes in the form of a number followed by a unit, for example 1.5 GB.

## query.level.1.minutes, query.level.2.minutes, query.level.3.minutes

The Core database supports three query priority levels. Each core user is assigned to one of the priority levels. Thus, there are up to 3 groups of users that can be defined for the purposes of performance tuning. These settings control how long each user level is allowed to execute the queries. For example, lower privileged users may have a lower value so that they are not able to use all the Core device's resources with long-running queries.

## Per-user Configuration

There are settings that influence the actions users are allowed to perform on the database. These settings are stored in the configuration tree at /users/accounts/username/config, where username is the name of the user to which the settings apply.

query.prefix

A query prefix applies a filter to every query operation that the user performs. This is implemented by taking the query.prefix values and appending it the where clause of each query using the logical && (and) operator.

## query.level

The query.level setting assigns the query level that the user will have for every query they perform. These influence whether their queries are limited by the query.level.1.minutes, query.level.2.minutes, or query.level.3.minutes.

## session.threshold

The session.threshold setting assigns a maximum session threshold for the user. If set, this threshold value is assigned to all values calls that the user performs.

## Rollover

The database operates as a first-in, first-out (FIFO) queue. New data is always appended to the database, and the oldest data is automatically removed as needed. Data that is in the middle of the database is immutable, meaning it cannot be modified. There are two mechanisms to for rollover: synchronous and asynchronous.

## Synchronous Rollover

Synchronous rollover refers to rollover settings that are applied in response to a write operation on the database. That means data will be removed from the database in direct response to the need to write new data. Synchronous rollover is configured by setting size values on the configuration for packet.dir, meta.dir, session.dir, and index.dir.
Synchronous rollover on the packet, meta, and session databases can occur within any write operation.
Synchronous rollover on the index occurs when the index is saved.

## Asynchronous Rollover

Asynchronous rollover refers to database file removal that occurs when an explicit rollover command is issued to the database. Most commonly this type of rollover is scheduled to run periodically using the Core service's built-in scheduler. It can also be explicitly requested by the user.

The asynchronous rollover command is the sizeRoll message present on the /index and /database nodes of the configuration tree. The message on the /database node will do size rollover on packet meta, and session databases only, while the message on the /index node can do simultaneous rollover on both the index and the packet, meta, and session databases.

As an example, here is a typical sizeRoll scheduler entry for an archiver:

```
pathname=/index minutes=5 msg=sizeRoll params="type=meta,session,packet maxSize=25TB"
```

This scheduler entry specifies that every 5 minutes, the database will ensure that the max size of the meta, session, packet, and index does not exceed 25 terabytes.

## Index Configuration

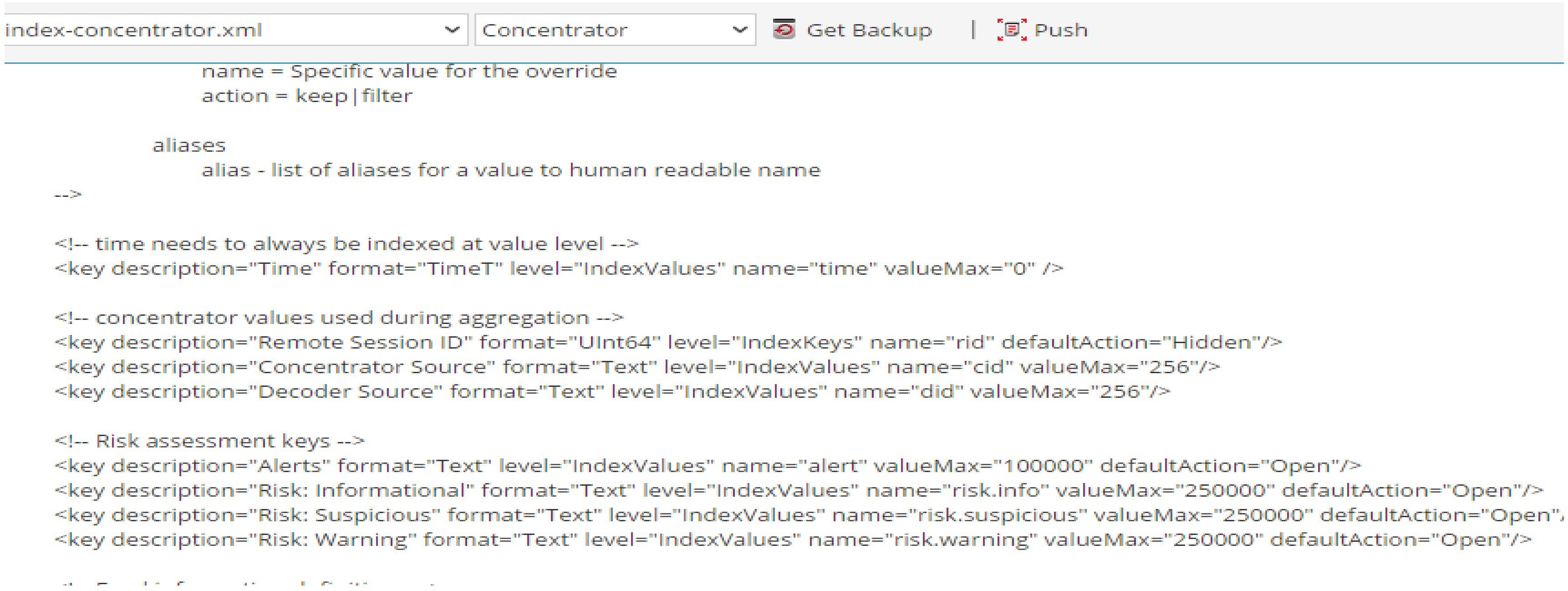
Default configuration file:

/etc/netwitness/ng/index-concentrator.xml

Customized configuration file:

/etc/netwitness/ng/index-concentrator-custom.xml

# Index config entries



```
                name = Specific value for the override
                action = keep|filter

            aliases
                alias - list of aliases for a value to human readable name
    -->

    <!-- time needs to always be indexed at value level -->
    <key description="Time" format="TimeT" level="IndexValues" name="time" valueMax="0" />

    <!-- concentrator values used during aggregation -->
    <key description="Remote Session ID" format="UInt64" level="IndexKeys" name="rid" defaultAction="Hidden"/>
    <key description="Concentrator Source" format="Text" level="IndexValues" name="cid" valueMax="256"/>
    <key description="Decoder Source" format="Text" level="IndexValues" name="did" valueMax="256"/>

    <!-- Risk assessment keys -->
    <key description="Alerts" format="Text" level="IndexValues" name="alert" valueMax="100000" defaultAction="Open"/>
    <key description="Risk: Informational" format="Text" level="IndexValues" name="risk.info" valueMax="250000" defaultAction="Open"/>
    <key description="Risk: Suspicious" format="Text" level="IndexValues" name="risk.suspicious" valueMax="250000" defaultAction="Open"
    <key description="Risk: Warning" format="Text" level="IndexValues" name="risk.warning" valueMax="250000" defaultAction="Open"/>
```

## Index Levels

### IndexNone

Not really an index at all and exist only to define and document meta key.

Used in Decoder to save performance for parsing.

### IndexKeys

Keeps track of sessions that contain meta items with meta key name.

Will not index any unique value in meta database.

Less storage space, memory & CPU time but low performance.

### IndexValues

This type of custom index keeps sessions that contain each individual unique value for the meta key. It is also known as a "full index".

This type of index is needed for efficient processing of most where clauses.

## Value Max

Limits the number of unique values that can enter the index.

Applies only to values added since last index save.

Varies from version to version but recommended ceiling is 5,000,000 For any meta key.

## Complex where clauses

- The amount of time it takes for the SA Core database to produce a result is dependent on the complexity of the query. Queries that align directly with the indexes present on the meta can be resolved quickly.
- A clause with a high cost takes longer to execute. Below, the query operations are ordered in terms of their relative cost, from lowest to highest.
  - exists, !exists
  - =, !=
  - <, >, <=, >=
  - Begins, ends, contains
  - regex

## Index Saves (Slices)

- The Core index is subdivided by save points, also known as slices. When the index is saved, all the data in the index is flushed to disk, and that portion of the index is marked as read-only. Saves serve two functions:

  1. Each save point represents a place where the index could be recovered in the case of a power failure.

  2. By periodically saving, we can ensure that the portion of the index that is actively being updated does not grow larger than RAM.

- Save points have the effect of partitioning the index into independent, non-overlapping segments. When a query must cross over multiple save points, it must re-execute parts of the query and merge the results together. This ultimately makes the query take longer to complete.

- By default a save is performed on the core index every 8 hours.

## Index Saves (Slices) cont'd

- By increasing the save interval, save points are created less frequently, and therefore fewer save points will exist. This has a positive effect on query performance, because it becomes less likely that queries will traverse slices, and when slices do have to be traversed, there are not as many to traverse.
- There are downsides to increasing the save interval though. First, the concentrator is more likely to hit the valueMax limit set on any of the indices. Second, the recovery time in the event of a forced shutdown or power failure is increased. And third, the aggregation rate may suffer if the index slice grows too large to fit in memory.

## Cache Window

- Consider this sequence of events:

    1. At 9:00 a.m., user "kevin" logs into a Concentrator and requests a report on the last 1 hour of collection time.

    2. The concentrator retrieves reports for the time range 8:00 am to 9:00 am

    3. At 9:02 a.m., user "scott" logs into the same Concentrator and also requests a report on the last 1 hour of collection time.

    4. The concentrator retrieves reports for the time range 8:02 am to 9:02 am

*Notice that even though both users were looking at time ranges that were close together, the work done by the concentrator to produce Kevin's reports could not be re-sent to Scott, since the time ranges are slightly different. Thus the Concentrator had to re-calculate most of the reports for Scott.*

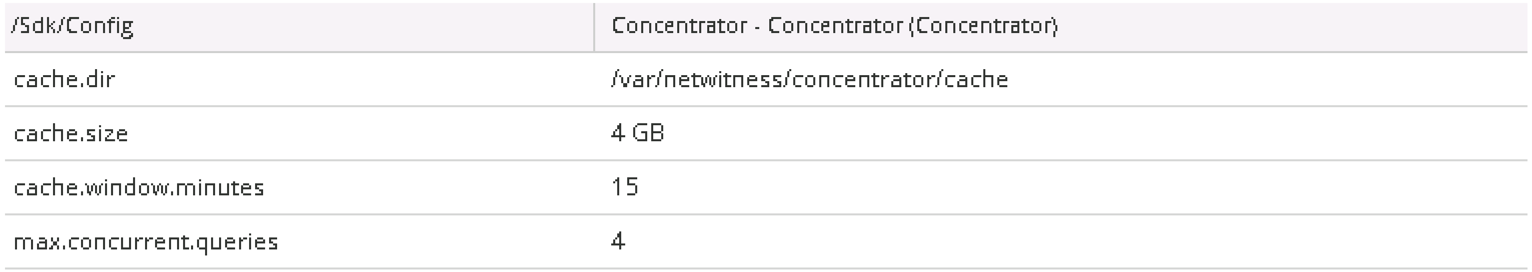
## Cache Window cont'd

- The setting /sdk/config/cache.window.minutes allows you to optimize this situation. When a user logs in, the point in time representing the most recent data for the collection only moves forward in increments of the number of minutes in this setting. For example, assume it is set to 10. If we re-evaluate the action in the previous slide, we get a new sequence of events.

    1. At 9:00 a.m., user "kevin" logs into a Concentrator and requests a report on the last 1 hour of collection time.
    2. The concentrator retrieves reports for the time range 8:00 a.m. to 9:00 a.m.
    3. At 9:02 a.m., user "scott" logs into the same Concentrator and also requests a report on the last 1 hour of collection time.
    4. The concentrator retrieves reports for the time range 8:00 a.m. to 9:00 a.m.
    5. At 9:10 a.m., user "scott" re-loads the reports for the last 1 hour of collection time.
    6. The concentrator retrieves reports for the time range 8:10 a.m. to 9:10 a.m.

## Cache Window cont'd

- The report returned at step 3 falls in the cache window, so it will be returned instantaneously. This will give Scott the impression that the Concentrator is very fast.
- Thus, larger cache.window settings improve perceived performance, at the cost of introducing small delays until the latest data is available to search.

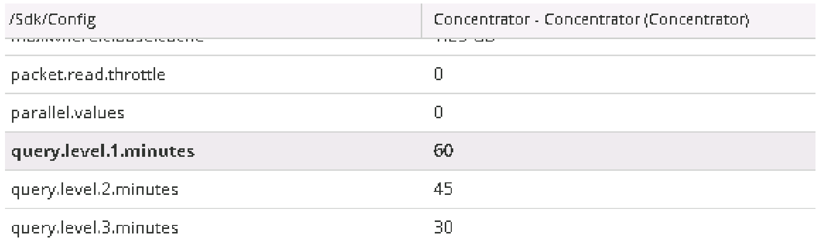


| /Sdk/Config | Concentrator - Concentrator (Concentrator) |
|---|---|
| cache.dir | /var/netwitness/concentrator/cache |
| cache.size | 4 GB |
| cache.window.minutes | 15 |
| max.concurrent.queries | 4 |

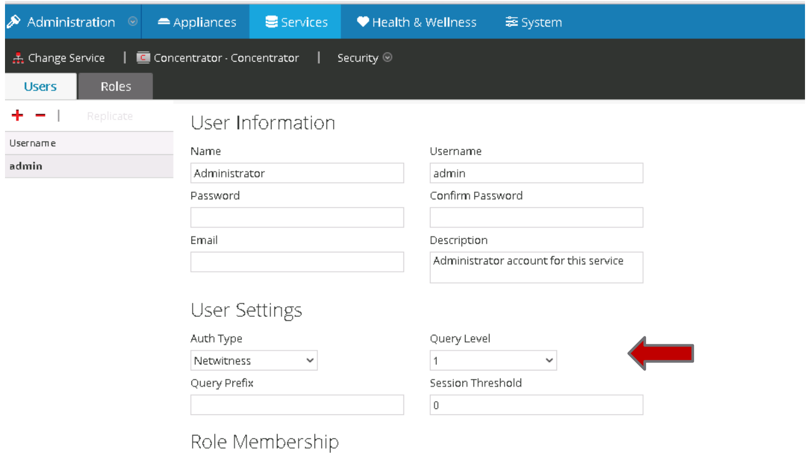
## Time Limits

- When a query is running on the SA Core database for a very long time, the Core service will dedicate more and more CPU time and RAM to that query in order to get it to complete faster. This can have a detrimental impact on other queries and aggregation. In order to prevent lower privileged users from utilizing more than their share of the Core service's resources, it is a good idea to put time limits on the queries run by normal users. This can be done from the users created from the security view of the concentrator from the SA UI.

# Time Limits cont'd



| /Sdk/Config | Concentrator - Concentrator (Concentrator) |
|---|---|
| packet.read.throttle | 0 |
| parallel.values | 0 |
| **query.level.1.minutes** | 60 |
| query.level.2.minutes | 45 |
| query.level.3.minutes | 30 |



| Username |
|---|
| **admin** |

User Information

| | |
|---|---|
| Name: Administrator | Username: admin |
| Password: | Confirm Password: |
| Email: | Description: Administrator account for this service |

User Settings

| | |
|---|---|
| Auth Type: Netwitness | Query Level: 1 |
| Query Prefix: | Session Threshold: 0 |

Role Membership

## Questions?

## Thank you for your time